Oracle® Streams
Advanced Queuing User's Guide
11g Release 2 (11.2)
E11013-01

July 2009
Contents

Preface.............................................................................................................................................................. xvi

Audience............................................................................................................................................................ xvii
Documentation Accessibility ......................................................................................................................... xvii
Related Documents ........................................................................................................................................ xviii
Conventions ....................................................................................................................................................... xviii

What's New in Oracle® Database? ................................................................................................................ xix

Notification Enhancements ........................................................................................................................... xix
Better Diagnosability and Manageability ..................................................................................................... xx
Transition from Job Queue Processes to Database Scheduler ................................................................. xx
Messaging Gateway Enhancements ........................................................................................................... xx

1 Introduction to Oracle® Database

What Is Queuing? ............................................................................................................................................ 1-1
Oracle® Database Leverages Oracle Database ........................................................................................... 1-2
Oracle® Database in Integrated Application Environments ....................................................................... 1-5
Oracle® Database Client/Server Communication ....................................................................................... 1-6
Multiconsumer Dequeuing of the Same Message ....................................................................................... 1-7
Oracle® Database Implementation of Workflows .................................................................................... 1-9
Oracle® Database Implementation of Publish/Subscribe ....................................................................... 1-10
Buffered Messaging ..................................................................................................................................... 1-12
Asynchronous Notifications ....................................................................................................................... 1-16
Views on Registration ................................................................................................................................. 1-18
Event-Based Notification .......................................................................................................................... 1-18
Notification Grouping by Time .................................................................................................................. 1-18
Enqueue Features ......................................................................................................................................... 1-19
Dequeue Features ......................................................................................................................................... 1-21
Propagation Features ................................................................................................................................... 1-27
Message Format Transformation ................................................................................................................ 1-34
Other Oracle® Database Features ............................................................................................................. 1-34
Interfaces to Oracle® Database ................................................................................................................ 1-38
Oracle® Database Demonstrations ........................................................................................................... 1-38

2 Basic Components

Object Name ................................................................................................................................................... 2-1
Type Name........................................................................................................................................2-2
AQ Agent Type ..............................................................................................................................2-2
AQ Recipient List Type ................................................................................................................2-3
AQ Agent List Type ......................................................................................................................2-3
AQ Subscriber List Type ...............................................................................................................2-3
AQ Registration Information List Type ..........................................................................................2-3
AQ Post Information List Type ......................................................................................................2-3
AQ Registration Information Type .................................................................................................2-3
AQ Notification Descriptor Type .....................................................................................................2-5
AQ Message Properties Type ........................................................................................................2-5
AQ Post Information Type ...............................................................................................................2-6
AQ$_NTFN_MSGID_ARRAY Type .......................................................................................................2-6
Enumerated Constants in the Oracle® Database Administrative Interface ...........................................2-6
Enumerated Constants in the Oracle® Database Operational Interface ..............................................2-7
AQ Background Processes ..............................................................................................................2-7
  Queue Monitor Processes ............................................................................................................2-8
  Job Queue Processes ....................................................................................................................2-8

3  Oracle® Database: Programmatic Interfaces

Programmatic Interfaces for Accessing Oracle® Database ..............................................................3-1
Using PL/SQL to Access Oracle® Database ..................................................................................3-2
Using OCI to Access Oracle® Database .........................................................................................3-3
Using OCCI to Access Oracle® Database .....................................................................................3-3
Using Visual Basic (OO4O) to Access Oracle® Database .............................................................3-3
Using Oracle Java Message Service (OJMS) to Access Oracle® Database .................................3-4
  Accessing Standard and Oracle JMS Applications ..................................................................3-5
Using Oracle® Database XML Servlet to Access Oracle® Database ............................................3-5
Comparing Oracle® Database Programmatic Interfaces ...................................................................3-6
  Oracle® Database Administrative Interfaces .............................................................................3-6
  Oracle® Database Operational Interfaces ..................................................................................3-7

4  Managing Oracle® Database

Oracle® Database Compatibility Parameters ...................................................................................4-1
Queue Security and Access Control ...............................................................................................4-2
  Oracle® Database Security .........................................................................................................4-2
    Administrator Role ..................................................................................................................4-2
    User Role ..................................................................................................................................4-2
  Access to Oracle® Database Object Types .................................................................................4-3
Queue Security ..............................................................................................................................4-3
Queue Privileges and Access Control ............................................................................................4-3
  OCI Applications and Queue Access .......................................................................................4-3
  Security Required for Propagation ..........................................................................................4-4
Queue Table Export-Import ...........................................................................................................4-4
  Exporting Queue Table Data ....................................................................................................4-4
  Importing Queue Table Data ....................................................................................................4-5
  Data Pump Export and Import ...................................................................................................4-6
Oracle Enterprise Manager Support ..............................................................................................4-6
Using Oracle® Database with XA ................................................................. 4-6
Restrictions on Queue Management ............................................................. 4-7
  Subscribers ................................................................................................. 4-7
  DML Not Supported on Queue Tables or Associated IOTs .................. 4-7
  Propagation from Object Queues with REF Payload Attributes ......... 4-7
  Collection Types in Message Payloads ................................................... 4-7
  Synonyms on Queue Tables and Queues ............................................... 4-8
  Synonyms on Object Types ..................................................................... 4-8
  Tablespace Point-in-Time Recovery ....................................................... 4-8
  Virtual Private Database ......................................................................... 4-8
Managing Propagation ............................................................................... 4-8
  EXECUTE Privileges Required for Propagation ...................................... 4-8
  Propagation from Object Queues ............................................................ 4-9
  Optimizing Propagation .......................................................................... 4-9
  Handling Failures in Propagation ............................................................ 4-10

5 Oracle® Database Performance and Scalability

  Persistent Messaging Performance Overview ........................................ 5-1
    Oracle® Database and Oracle Real Application Clusters .................... 5-1
    Oracle® Database in a Shared Server Environment ............................... 5-2
  Persistent Messaging Basic Tuning Tips ................................................ 5-2
    Memory Requirements ......................................................................... 5-2
    Using Storage Parameters .................................................................... 5-2
    I/O Configuration .................................................................................. 5-3
    Running Enqueue and Dequeue Processes Concurrently in a Single Queue Table ......................................................................................................................... 5-3
    Running Enqueue and Dequeue Processes Serially in a Single Queue Table ................................................................. 5-3
    Creating Indexes on a Queue Table ...................................................... 5-3
    Other Tips ............................................................................................ 5-3
  Propagation Tuning Tips .......................................................................... 5-4
  Buffered Messaging Tuning .................................................................... 5-4
  Performance Views .................................................................................. 5-5

6 Internet Access to Oracle® Database

  Overview of Oracle® Database Operations over the Internet .................. 6-1
    Oracle® Database Internet Operations Architecture ............................ 6-1
    Internet Message Payloads ..................................................................... 6-2
    Configuring the Web Server to Authenticate Users Sending POST Requests ................................................................. 6-3
    Client Requests Using HTTP .................................................................. 6-3
      User Sessions and Transactions ......................................................... 6-3
    Oracle® Database Servlet Responses Using HTTP ............................... 6-3
    Oracle® Database Propagation Using HTTP and HTTPS ...................... 6-4
  Deploying the Oracle® Database XML Servlet ..................................... 6-4
  Internet Data Access Presentation (IDAP) .............................................. 6-7
    SOAP Message Structure ..................................................................... 6-7
    SOAP Envelope .................................................................................... 6-7
    SOAP Header ....................................................................................... 6-7
SOAP Body ....................................................................................................................... 6-7
SOAP Method Invocation ................................................................................................. 6-8
HTTP Headers .................................................................................................................. 6-8
Method Invocation Body ................................................................................................. 6-8
Results from a Method Request ...................................................................................... 6-9
Request and Response IDAP Documents ...................................................................... 6-9
IDAP Client Requests for Enqueue ................................................................................. 6-9
IDAP Client Requests for Dequeue ................................................................................ 6-11
IDAP Client Requests for Registration ......................................................................... 6-13
IDAP Client Requests to Commit a Transaction .............................................................. 6-13
IDAP Client Requests to Roll Back a Transaction .......................................................... 6-14
IDAP Server Response to an Enqueue Request .............................................................. 6-14
IDAP Server Response to a Dequeue Request .............................................................. 6-14
IDAP Server Response to a Register Request ............................................................... 6-15
IDAP Commit Response .................................................................................................. 6-15
IDAP Rollback Response ............................................................................................... 6-15
IDAP Notification ............................................................................................................ 6-15
IDAP Response in Case of Error ...................................................................................... 6-15
Notification of Messages by e-mail .................................................................................. 6-16

7 Troubleshooting Oracle® Database

Debugging Oracle® Database Propagation Problems ....................................................... 7-1
Oracle® Database Error Messages .................................................................................. 7-2

8 Oracle® Database Administrative Interface

Managing Queue Tables ................................................................................................. 8-1
Creating a Queue Table ................................................................................................... 8-1
Altering a Queue Table ................................................................................................... 8-8
Dropping a Queue Table ................................................................................................. 8-9
Purging a Queue Table .................................................................................................... 8-9
Migrating a Queue Table ................................................................................................. 8-11
Managing Queues ........................................................................................................... 8-12
Creating a Queue ............................................................................................................ 8-12
Altering a Queue ............................................................................................................. 8-15
Starting a Queue ............................................................................................................. 8-15
Stopping a Queue ........................................................................................................... 8-16
Dropping a Queue ............................................................................................................ 8-16
Managing Transformations ........................................................................................... 8-16
Creating a Transformation ............................................................................................. 8-17
Modifying a Transformation ......................................................................................... 8-17
Dropping a Transformation ............................................................................................ 8-18
Granting and Revoking Privileges ............................................................................... 8-18
Granting Oracle® Database System Privileges .............................................................. 8-18
Revoking Oracle® Database System Privileges .............................................................. 8-19
Granting Queue Privileges ............................................................................................. 8-19
Revoking Queue Privileges ............................................................................................. 8-20
Managing Subscribers .................................................................................................... 8-20
Adding a Subscriber ................................................................. 8-20
 Altering a Subscriber .............................................................. 8-22
 Removing a Subscriber............................................................. 8-23
Managing Propagations .............................................................. 8-23
 Scheduling a Queue Propagation ............................................. 8-24
 Verifying Propagation Queue Type ........................................... 8-26
 Altering a Propagation Schedule ............................................. 8-26
 Enabling a Propagation Schedule ............................................ 8-27
 Disabling a Propagation Schedule .......................................... 8-27
 Unscheduling a Queue Propagation ........................................ 8-28
Managing Oracle® Database Agents .......................................... 8-28
 Creating an Oracle® Database Agent ....................................... 8-29
 Altering an Oracle® Database Agent ....................................... 8-29
 Dropping an Oracle® Database Agent ................................. 8-29
 Enabling Database Access ........................................................ 8-29
Disabling Database Access ...................................................... 8-30
Adding an Alias to the LDAP Server ....................................... 8-30
 Deleting an Alias from the LDAP Server .................................. 8-30

9 Oracle® Database & Messaging Gateway Views

DBA_QUEUE_TABLES: All Queue Tables in Database .................. 9-2
USER_QUEUE_TABLES: Queue Tables in User Schema ................ 9-3
ALL_QUEUE_TABLES: Queue Tables Queue Accessible to the Current User 9-3
DBA_QUEUES: All Queues in Database ...................................... 9-3
USER_QUEUES: Queues In User Schema ................................... 9-3
ALl_QUEUES: Queues for Which User Has Any Privilege ................. 9-3
DBA_QUEUE_SCHEDULES: All Propagation Schedules .................. 9-3
USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema .......... 9-3
QUEUE_PRIVILEGES: Queues for Which User Has Queue Privilege ........ 9-3
AQ$Queue_Table_Name: Messages in Queue Table ....................... 9-4
AQ$Queue_Table_Name_S: Queue Subscribers ............................ 9-6
AQ$Queue_Table_Name_R: Queue Subscribers and Their Rules .......... 9-7
DBA_QUEUE_SUBSCRIBERS: All Queue Subscribers in Database ........ 9-7
USER_QUEUE_SUBSCRIBERS: Queue Subscribers in User Schema ........ 9-7
ALL_QUEUE_SUBSCRIBERS: Subscribers for Queues Where User Has Queue Privileges .... 9-7
DBA_TRANSFORMATIONS: All Transformations ......................... 9-7
DBA_ATTRIBUTE_TRANSFORMATIONS: All Transformation Functions ........ 9-7
USER_TRANSFORMATIONS: User Transformations ....................... 9-8
USER_ATTRIBUTE_TRANSFORMATIONS: User Transformation Functions ........ 9-8
DBA_SUBSCR_REGISTRATIONS: All Subscription Registrations .......... 9-8
USER_SUBSCR_REGISTRATIONS: User Subscription Registrations .......... 9-8
AQ$INTERNET_USERS: Oracle® Database Agents Registered for Internet Access .......... 9-8
(G)V$SAQ: Number of Messages in Different States in Database ................. 9-8
(G)V$BUFFERED_QUEUES: All Buffered Queues in the Instance ....... 9-9
(G)V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance .... 9-9
(G)V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance ....... 9-9
(G)V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance .......... 9-9
10 Oracle® Database Operations Using PL/SQL

Using Secure Queues ................................................................. 10-1
Enqueuing Messages ............................................................... 10-2
Enqueuing an Array of Messages .............................................. 10-10
Listening to One or More Queues ............................................. 10-11
Dequeuing Messages ............................................................... 10-13
Dequeuing an Array of Messages .............................................. 10-20
Registering for Notification ..................................................... 10-22
Unregistering for Notification .................................................. 10-23
Posting for Subscriber Notification ........................................ 10-23
Adding an Agent to the LDAP Server ....................................... 10-24
Removing an Agent from the LDAP Server .............................. 10-25

11 Introducing Oracle JMS

General Features of JMS and Oracle JMS .................................... 11-1
  JMS Connection and Session ................................................ 11-1
  ConnectionFactory Objects ................................................... 11-2
  Using AQmssFactory to Obtain ConnectionFactory Objects ........ 11-2
  Using JNDI to Look Up ConnectionFactory Objects .................. 11-2

(G)V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues .................................................. 9-9
(G)V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance ...................................................... 9-9
(G)V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance ...................................................... 9-9
(G)V$PROPAGATION_SENDER: Buffer Queue Propagation Schedules on the Sending (Source) Side .................................................................... 9-10
(G)V$PROPAGATION_RECEIVER: Buffer Queue Propagation Schedules on the Receiving (Destination) Side ........................................ 9-10
(G)V$SUBSCR_REGISTRATION_STATS: Diagnosability of Notifications ................................................................. 9-10
V$METRICGROUP: Information about the Metric Group ............... 9-10
(G)V$STREAMSMETRIC: Streams Metrics for the Most Recent Interval ................................................................. 9-10
(G)V$STREAMSMETRIC_HISTORY: Streams Metrics Over Past Hour .............................................................................. 9-11
(G)V$QUEUEMETRIC: Queue Metrics for the Most Recent Interval .............................................................................. 9-11
(G)V$QUEUEMETRIC_HISTORY: Queue Metrics Over Past Hour .............................................................................. 9-11
DBA_HIST_STREAMSMETRIC: Streams Metric History ................. 9-11
DBA_HIST_QUEUEMETRIC: Queue Metric History .................... 9-12
MGW_GATEWAY: Configuration and Status Information ................ 9-12
MGW_AGENT_OPTIONS: Supplemental Options and Properties .... 9-14
MGW_LINKS: Names and Types of Messaging System Links .......... 9-14
MGW_MQSERIES_LINKS: WebSphere MQ Messaging System Links ................................................................. 9-14
MGW_TIBRV_LINKS: TIB/Rendezvous Messaging System Links .... 9-15
MGW_FOREIGN_QUEUES: Foreign Queues ................................. 9-16
MGW_JOBS: Messaging Gateway Propagation Jobs .................... 9-16
MGW_SUBSCRIBERS: Information for Subscribers ..................... 9-17
MGW_SCHEDULES: Information about Schedules ....................... 9-18
JMS Connection .................................................................................................................. 11-3
JMS Session ....................................................................................................................... 11-5
JMS Destination .................................................................................................................. 11-6
Using a JMS Session to Obtain Destination Objects ....................................................... 11-6
Using JNDI to Look Up Destination Objects .................................................................. 11-7
JMS Destination Methods ............................................................................................... 11-7
System-Level Access Control in JMS .............................................................................. 11-7
Destination-Level Access Control in JMS ......................................................................... 11-8
Retention and Message History in JMS ............................................................................ 11-8
Supporting Oracle Real Application Clusters in JMS ....................................................... 11-8
Supporting Statistics Views in JMS .................................................................................. 11-9

Structured Payload/Message Types in JMS ................................................................... 11-9
JMS Message Headers ....................................................................................................... 11-9
JMS Message Properties ................................................................................................... 11-10
JMS Message Bodies ........................................................................................................ 11-12
  StreamMessage ........................................................................................................... 11-12
  BytesMessage .......................................................................................................... 11-12
  MapMessage ............................................................................................................. 11-13
  TextMessage ............................................................................................................ 11-13
  ObjectMessage ....................................................................................................... 11-13
  AdtMessage ............................................................................................................ 11-13
Using Message Properties with Different Message Types ............................................... 11-14
Buffered Messaging with Oracle JMS ............................................................................. 11-15

JMS Point-to-Point Model Features ................................................................................ 11-16

JMS Publish/Subscribe Model Features .......................................................................... 11-17
JMS Publish/Subscribe Overview ................................................................................... 11-18
DurableSubscriber .......................................................................................................... 11-18
RemoteSubscriber ......................................................................................................... 11-19
TopicPublisher ............................................................................................................... 11-19
Recipient Lists ............................................................................................................... 11-19
TopicReceiver ............................................................................................................. 11-19
TopicBrowser .............................................................................................................. 11-20
Setting Up JMS Publish/Subscribe Operations ............................................................. 11-20

JMS MessageProducer Features ...................................................................................... 11-21
Priority and Ordering of Messages .................................................................................. 11-21
Specifying a Message Delay ............................................................................................ 11-22
Specifying a Message Expiration .................................................................................... 11-22
Message Grouping ........................................................................................................ 11-22

JMS Message Consumer Features .................................................................................. 11-22
Receiving Messages ....................................................................................................... 11-23
Message Navigation in Receive ...................................................................................... 11-23
Browsing Messages ........................................................................................................ 11-24
Remove No Data .......................................................................................................... 11-24
Retry with Delay Interval ............................................................................................... 11-24
Asynchronously Receiving Messages Using MessageListener .................................... 11-25
Exception Queues .......................................................................................................... 11-25

JMS Propagation ............................................................................................................... 11-26
RemoteSubscriber................................................................. 11-26
Scheduling Propagation......................................................... 11-26
Enhanced Propagation Scheduling Capabilities ....................... 11-27
Exception Handling During Propagation............................. 11-28
Message Transformation with JMS AQ............................. 11-29
J2EE Compliance ............................................................... 11-29

12 Oracle JMS Basic Operations

EXECUTE Privilege on DBMS_AQIN ........................................ 12-1
Registering a ConnectionFactory .......................................... 12-1
  Registering Through the Database Using JDBC Connection Parameters ........................................ 12-1
  Registering Through the Database Using a JDBC URL ................................................................. 12-2
  Registering Through LDAP Using JDBC Connection Parameters .................................................. 12-3
  Registering Through LDAP Using a JDBC URL ............................................................................ 12-4
Unregistering a Queue/Topic ConnectionFactory .................. 12-5
  Unregistering Through the Database ............................................................ 12-5
Getting a QueueConnectionFactory or TopicConnectionFactory ..................................................... 12-6
  Getting a QueueConnectionFactory with JDBC URL ............................................................... 12-6
  Getting a QueueConnectionFactory with JDBC Connection Parameters .................................. 12-7
  Getting a TopicConnectionFactory with JDBC URL ............................................................... 12-7
  Getting a TopicConnectionFactory with JDBC Connection Parameters .................................. 12-8
  Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP ................................ 12-8
Getting a Queue or Topic in LDAP ......................................... 12-9
Creating a Queue Table ....................................................... 12-9
Creating a Queue ................................................................. 12-10
  Creating a Point-to-Point Queue ......................................................................................... 12-10
  Creating a Publish/Subscribe Topic .................................................................................. 12-11
Granting and Revoking Privileges ........................................ 12-12
  Granting Oracle® Database System Privileges ........................................................................ 12-13
  Revoking Oracle® Database System Privileges ................................................................... 12-13
  Granting Publish/Subscribe Topic Privileges ........................................................................ 12-14
  Revoking Publish/Subscribe Topic Privileges ................................................................... 12-14
  Granting Point-to-Point Queue Privileges ........................................................................ 12-14
  Revoking Point-to-Point Queue Privileges ........................................................................ 12-15
Managing Destinations .......................................................... 12-16
  Starting a Destination ..................................................................................................... 12-16
  Stopping a Destination .................................................................................................. 12-16
  Altering a Destination .................................................................................................. 12-17
  Dropping a Destination .................................................................................................. 12-17
Propagation Schedules .......................................................... 12-17
  Scheduling a Propagation .............................................................................................. 12-18
  Enabling a Propagation Schedule ................................................................................... 12-18
  Altering a Propagation Schedule .................................................................................... 12-19
  Disabling a Propagation Schedule ................................................................................ 12-19
  Un scheduling a Propagation .......................................................................................... 12-20
13 Oracle JMS Point-to-Point

Creating a QueueSession ................................................................. 13-4
Creating a QueueConnection with Default ConnectionFactory Parameters .......... 13-2
Creating a QueueConnection with an Open JDBC Connection ............................ 13-2
Creating a QueueConnection with an Open OracleOCIConnectionPool ................. 13-3
Creating a QueueConnection with Username/Password ..................................... 13-2
Creating a QueueConnection with Open JDBC Connection ............................... 13-3
Creating a QueueReceiver for Oracle Object Type Messages ............................ 13-7
Creating a QueueBrowser for Standard JMS Type Messages .............................. 13-5
Creating a QueueBrowser for Oracle Object Type Messages ............................. 13-7
Creating a QueueBrowser for Oracle Object Type Messages, Locking Messages .... 13-8
Creating a QueueBrowser for Standard JMS Type Messages, Locking Messages ...... 13-7
Creating a QueueBrowser for Standard JMS Type Messages ............................ 13-9
Creating a QueueBrowser for Standard JMS Messages, Locking Messages .......... 13-7
Creating a Connection with Username/Password ........................................... 13-1
Creating a QueueReceiver for Oracle Object Type Messages, Locking Messages .... 13-9

14 Oracle JMS Publish/Subscribe

Creating a TopicPublisher ................................................................ 14-4
Creating a TopicSession .................................................................. 14-4
Creating a TopicSession with Open JDBC Connection ................................. 14-4
Creating a TopicSession with Username/Password ........................................ 14-4
Creating a TopicConnection with Username/Password .................................... 14-2
Creating a TopicConnection with Open JDBC Connection ............................... 14-3
Creating a TopicConnection with an Open OracleOCIConnectionPool ............... 14-3
Creating a TopicConnection with Username/Password ..................................... 14-4
Creating a TopicConnection with Open JDBC Connection ............................... 14-4
Creating a TopicConnection with Open OracleOCIConnectionPool ................... 14-4
Creating a Remote Subscriber for JMS Messages ......................................... 14-13
Creating a Remote Subscriber for Oracle Object Type Messages ...................... 14-14
Creating a Remote Subscriber for Oracle Object Type Messages, Locking Messages 14-15
Specifying Transformations for Topic Subscribers ......................................... 14-12
Specifying Transformations for Remote Subscribers ...................................... 14-15
Unsubscribing a Durable Subscription for a Local Subscriber ........................... 14-16
Unsubscribing a Durable Subscription for a Remote Subscriber ....................... 14-17
Creating a Remote Subscriber for a Topic of Standard JMS Type Messages ......... 14-17
Creating a Remote Subscriber for a Topic of Oracle Object Type Messages ......... 14-18
Creating a TopicPublisher for Oracle Object Type Messages, Locking Messages 14-19
15 Oracle JMS Shared Interfaces

Oracle® Database JMS Operational Interface: Shared Interfaces ........................................... 15-1

Starting a JMS Connection ................................................................................................. 15-2
Getting a JMS Connection ................................................................................................. 15-2
Committing All Operations in a Session ............................................................................ 15-2
Rolling Back All Operations in a Session .......................................................................... 15-2
Getting the JDBC Connection from a Session .................................................................... 15-2
Getting the OracleOCIConnectionPool from a JMS Connection .................................... 15-2
Creating a BytesMessage .................................................................................................. 15-3
Creating a MapMessage .................................................................................................. 15-3
Creating a StreamMessage .............................................................................................. 15-3
Creating an ObjectMessage ............................................................................................. 15-3
Creating a TextMessage ................................................................................................... 15-3
Creating a JMS Message ................................................................................................. 15-4
Creating an AdtMessage .................................................................................................... 15-4
Setting JMS Correlation Identifier .................................................................................. 15-4

Specifying JMS Message Properties ................................................................................ 15-4

Setting a Boolean Message Property ................................................................................ 15-5
Setting a String Message Property ................................................................................... 15-5
Setting an Integer Message Property ................................................................................ 15-5
Setting a Double Message Property .................................................................................. 15-6
Setting a Float Message Property ..................................................................................... 15-6
Setting a Byte Message Property ...................................................................................... 15-6
Setting a Long Message Property .................................................................................... 15-6
Setting a Short Message Property .................................................................................... 15-7
Setting an Object Message Property ................................................................................ 15-7

Setting Default TimeToLive for All Messages Sent by a MessageProducer .................... 15-7

Setting Default Priority for All Messages Sent by a MessageProducer .......................... 15-8

Creating an AQjms Agent ............................................................................................... 15-8

Receiving a Message Synchronously .............................................................................. 15-8

Using a Message Consumer by Specifying Timeout ....................................................... 15-8
Using a Message Consumer Without Waiting ..................................................................... 15-10
Receiving Messages from a Destination Using a Transformation .................................... 15-10

Specifying the Navigation Mode for Receiving Messages ............................................. 15-11

Receiving a Message Asynchronously .......................................................................... 15-12

Specifying a Message Listener at the Message Consumer ............................................ 15-12
Specifying a Message Listener at the Session ................................................................ 15-13

Getting Message ID ......................................................................................................... 15-13

Getting the Correlation Identifier .................................................................................... 15-13
Getting the Message Identifier ........................................................................................ 15-13

Getting JMS Message Properties .................................................................................... 15-13

Getting a Boolean Message Property ................................................................................ 15-14
Getting a String Message Property ................................................................................... 15-14
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Oracle JMS Types Examples</td>
<td>16-1</td>
</tr>
<tr>
<td>How to Run the Oracle® Database JMS Type Examples</td>
<td>16-1</td>
</tr>
<tr>
<td>Setting Up the Examples</td>
<td>16-1</td>
</tr>
<tr>
<td>JMS BytesMessage Examples</td>
<td>16-5</td>
</tr>
<tr>
<td>JMS StreamMessage Examples</td>
<td>16-10</td>
</tr>
<tr>
<td>JMS MapMessage Examples</td>
<td>16-15</td>
</tr>
<tr>
<td>More Oracle® Database JMS Examples</td>
<td>16-21</td>
</tr>
<tr>
<td>17 Introducing Oracle Messaging Gateway</td>
<td>17-1</td>
</tr>
<tr>
<td>Introducing Oracle Messaging Gateway</td>
<td>17-1</td>
</tr>
<tr>
<td>Oracle Messaging Gateway Features</td>
<td>17-1</td>
</tr>
<tr>
<td>Oracle Messaging Gateway Architecture</td>
<td>17-3</td>
</tr>
<tr>
<td>Administration Package DBMS_MGWADM</td>
<td>17-3</td>
</tr>
<tr>
<td>Oracle Messaging Gateway Agent</td>
<td>17-4</td>
</tr>
<tr>
<td>Oracle Database</td>
<td>17-4</td>
</tr>
<tr>
<td>Non-Oracle Messaging Systems</td>
<td>17-4</td>
</tr>
<tr>
<td>Propagation Processing Overview</td>
<td>17-4</td>
</tr>
<tr>
<td>Oracle Streams AQ Buffered Messages and Messaging Gateway</td>
<td>17-5</td>
</tr>
<tr>
<td>18 Getting Started with Oracle Messaging Gateway</td>
<td>18-1</td>
</tr>
<tr>
<td>Oracle Messaging Gateway Prerequisites</td>
<td>18-1</td>
</tr>
<tr>
<td>Loading and Setting Up Oracle Messaging Gateway</td>
<td>18-1</td>
</tr>
<tr>
<td>Loading Database Objects into the Database</td>
<td>18-2</td>
</tr>
<tr>
<td>Modifying listener.ora for the External Procedure</td>
<td>18-2</td>
</tr>
<tr>
<td>Modifying tnsnames.ora for the External Procedure</td>
<td>18-3</td>
</tr>
</tbody>
</table>
19 Working with Oracle Messaging Gateway

Configuring the Oracle Messaging Gateway Agent .............................................................. 19-1
Creating a Messaging Gateway Agent ................................................................................. 19-2
Removing a Messaging Gateway Agent .............................................................................. 19-2
Database Connection ........................................................................................................... 19-2
Resource Limits ................................................................................................................. 19-3

Starting and Shutting Down the Oracle Messaging Gateway Agent ..................................... 19-3
Starting the Oracle Messaging Gateway Agent ................................................................... 19-3
Shutting Down the Oracle Messaging Gateway Agent ......................................................... 19-3
Oracle Messaging Gateway Agent Scheduler Job .............................................................. 19-4
Running the Oracle Messaging Gateway Agent on RAC ..................................................... 19-5

Configuring Messaging System Links .................................................................................. 19-5
Creating a WebSphere MQ Base Java Link ......................................................................... 19-6
Creating a WebSphere MQ JMS Link ............................................................................... 19-7
Creating a WebSphere MQ Link to Use SSL ...................................................................... 19-9
Creating a TIB/Rendezvous Link ....................................................................................... 19-11
Altering a Messaging System Link .................................................................................... 19-11
Removing a Messaging System Link .................................................................................. 19-12
Views for Messaging System Links ................................................................................... 19-12

Configuring Non-Oracle Messaging System Queues ............................................................ 19-12
Registering a Non-Oracle Queue .......................................................................................... 19-13
Registering a WebSphere MQ Base Java Queue ................................................................. 19-13
Registering a WebSphere MQ JMS Queue or Topic .......................................................... 19-13
Registering a TIB/Rendezvous Subject .............................................................................. 19-14
Unregistering a Non-Oracle Queue .................................................................................... 19-14
View for Registered Non-Oracle Queues .......................................................................... 19-14

Configuring Oracle Messaging Gateway Propagation Jobs ................................................. 19-14
Propagation Job Overview ................................................................................................. 19-15
Creating an Oracle Messaging Gateway Propagation Job ................................................... 19-16
Enabling and Disabling a Propagation Job ................................................................. 19-16
Resetting a Propagation Job ...................................................................................... 19-17
Altering a Propagation Job ....................................................................................... 19-17
Removing a Propagation Job .................................................................................... 19-17

Propagation Jobs, Subscribers, and Schedules .......................................................... 19-18
Propagation Job, Subscriber, Schedule Interface Interoperability ............................. 19-19
Propagation Job, Subscriber, Schedule Views ........................................................... 19-20
Single Consumer Queue As Propagation Source ..................................................... 19-20

Configuration Properties .......................................................................................... 19-20
WebSphere MQ System Properties ............................................................................ 19-20
TIB/Rendezvous System Properties ......................................................................... 19-22
Optional Link Configuration Properties .................................................................... 19-23
Optional Foreign Queue Configuration Properties .................................................. 19-25
Optional Job Configuration Properties ...................................................................... 19-26

20 Oracle Messaging Gateway Message Conversion

Converting Oracle Messaging Gateway Non-JMS Messages ....................................... 20-1
Overview of the Non-JMS Message Conversion Process .......................................... 20-1
Oracle Messaging Gateway Canonical Types .......................................................... 20-2
Message Header Conversion ..................................................................................... 20-2
Handling Arbitrary Payload Types Using Message Transformations ....................... 20-2
Handling Logical Change Records .......................................................................... 20-4

Message Conversion for WebSphere MQ ................................................................. 20-6
WebSphere MQ Message Header Mappings ............................................................. 20-6
WebSphere MQ Outbound Propagation .................................................................... 20-9
WebSphere MQ Inbound Propagation ....................................................................... 20-9

Message Conversion for TIB/Rendezvous ............................................................... 20-10
AQ Message Property Mapping for TIB/Rendezvous ............................................. 20-12
TIB/Rendezvous Outbound Propagation ................................................................. 20-12
TIB/Rendezvous Inbound Propagation ................................................................... 20-13

JMS Messages ........................................................................................................... 20-14
JMS Outbound Propagation ..................................................................................... 20-15
JMS Inbound Propagation ....................................................................................... 20-15

21 Monitoring Oracle Messaging Gateway

Oracle Messaging Gateway Log Files ........................................................................ 21-1
Sample Oracle Messaging Gateway Log File ........................................................... 21-1
Interpreting Exception Messages in an Oracle Messaging Gateway Log File .......... 21-3

Monitoring the Oracle Messaging Gateway Agent Status ..................................... 21-3
MGW_GATEWAY View ............................................................................................. 21-3
Oracle Messaging Gateway Irrecoverable Error Messages .................................. 21-4
Other Oracle Messaging Gateway Error Conditions ............................................. 21-7

Monitoring Oracle Messaging Gateway Propagation ............................................. 21-8

Oracle Messaging Gateway Agent Error Messages .............................................. 21-9
Preface

This guide describes features of application development and integration using Oracle Streams Advanced Queuing (AQ). This information applies to versions of the Oracle Database server that run on all platforms, unless otherwise specified.

This Preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for programmers who develop applications that use Oracle® Database.

To use this document, you need knowledge of an application development language and object-relational database management concepts.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.
Accessibility of Links to External Web Sites in Documentation
This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services
To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Related Documents
For more information, see these Oracle resources:
- Oracle Database Advanced Application Developer’s Guide
- Oracle Database PL/SQL Language Reference
- Oracle Streams Advanced Queuing Java API Reference
- Oracle Database PL/SQL Packages and Types Reference
- Oracle Streams Concepts and Administration
- Oracle XML DB Developer’s Guide

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

Conventions
The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
What's New in Oracle® Database?

This chapter describes new features of the Oracle Database 11g Relase 1 (11.1) and provides pointers to additional information. There are no new features for 11g Release 2 (11.2).

- Notification Enhancements
- Better Diagnosability and Manageability
- Transition from Job Queue Processes to Database Scheduler
- Messaging Gateway Enhancements

Notification Enhancements

The following notification enhancements are introduced:

- Scalability for Streams Notifications
- Notification Grouping By Time

Scalability for Streams Notifications

AQ Event Notification Infrastructure provides asynchronous communication of database events from the suppliers/publishers of their events to the consumers/registrations. The event monitor sends these notifications. In order to meet the demands of increased notification use, the notification server in 11g is enhanced to a parallel notification server consisting of a coordinator and a set of subordinate processes. The parallel notification server offers a capability to process a greater volume of notifications, faster notification processing and lower shared memory use for staging notification events.

See Also: "Asynchronous Notifications" on page 1-16

Notification Grouping By Time

Sometimes a very large number of events occur in the database and it is important that applications not be overwhelmed with notifications. The preferred strategy is for notifications to be grouped and delivered at application-specified intervals and in application-specified formats. Oracle Streams AQ provides the infrastructure for notification grouping by time for AQ and DBCHANGE namespaces. Users have the option of specifying the grouping time interval and the predefined format in which to be notified at the end of those grouping intervals. Users can also specify when to start sending grouping notifications and how many times to send grouping notifications.

See Also: "Asynchronous Notifications" on page 1-16
Better Diagnosability and Manageability

The following sections describe diagnosability:

- New performance views and AWR support
- Dictionary View on Subscription Registrations
- Queue Table Level Export and Import

New performance views and AWR support
This release adds new performance views for persistent messaging statistics and notification statistics. The Automatic Workload Repository (AWR) has also been enhanced for displaying queues with the most persistent messaging operations, allowing for easier diagnosability of AQ performance problems.

Dictionary View on Subscription Registrations
New dictionary views are provided to simplify subscription management for Oracle Streams Advanced Queuing. `DBA_SUBSCR_REGISTRATIONS` and `USER_SUBSCR_REGISTRATIONS` identify registered subscriptions, as well as detail information on the subscriptions. Runtime statistics for notifications are available with the `V$SUBSCR_REGISTRATION_STATS` view.

Queue Table Level Export and Import
Export import of queues is now fully supported at queue table level granularity. The user only needs to export the queue table. All the queues in the queue table, primary object grants, related objects like views, IOTs, rules are automatically exported.

Transition from Job Queue Processes to Database Scheduler
EMON PL/SQL notifications are executed by background jobs. In this release these jobs are `DBMS_SCHEDULER` jobs and are no longer conducted by `DBMS_JOBS`.

The `init.ora` parameter `job_queue_processes` does not need to be set for PL/SQL notifications or AQ propagations.

AQ propagation is now likewise handled by `DBMS_SCHEDULER` jobs rather than `DBMS_JOBS`. Additionally, propagation takes advantage of the event based scheduling features of `DBMS_SCHEDULER` for better scalability.

Messaging Gateway Enhancements
The following Messaging Gateway enhancements are introduced:

- Enhanced Messaging Gateway Agent in a Real Application Clusters (RAC) Environment
- Multiple Messaging Gateway Agents
- Simplified Messaging Gateway Propagation Job Configuration

Enhanced Messaging Gateway Agent in a Real Application Clusters (RAC) Environment
The Oracle Scheduler will be used to start Messaging Gateway agents. Messaging Gateway will leverage the Oracle Scheduler RAC service feature so that a Messaging Gateway agent is associated with a database service. If the instance on which a
Messaging Gateway agent is running fails or is shutdown, the Oracle Scheduler will automatically restart the agent on another instance supporting that service.

**See Also:**
- "Configuring Oracle Messaging Gateway in a RAC Environment” on page 18-5
- "Running the Oracle Messaging Gateway Agent on RAC" on page 19-5

**Multiple Messaging Gateway Agents**
Messaging Gateway is enhanced to enable multiple agents per instance and database. With this enhancement, you can now statically partition propagation jobs based on functionality, organizations, or workload and assign them to different MGW agents hosted by different database instances on different machines. This not only enables MGW to scale, but also enables propagation job grouping and isolation, which is important when MGW is used in a complicated application integration environment.

**See Also:**
- "Getting Started with Oracle Messaging Gateway” on page 18-1
- "Working with Oracle Messaging Gateway” on page 19-1

**Simplified Messaging Gateway Propagation Job Configuration**
An enhanced PL/SQL API consolidates the propagation subscriber and the propagation schedule into a new propagation job. It is now easier to create and schedule a propagation job for the messaging gateway.

**See Also:**
- "Getting Started with Oracle Messaging Gateway” on page 18-1
- "Working with Oracle Messaging Gateway” on page 19-1
This chapter discusses Oracle Streams Advanced Queuing (AQ) and the requirements for complex information handling in an integrated environment.

This chapter contains the following topics:

- What Is Queuing?
- Oracle® Database Leverages Oracle Database
- Oracle® Database in Integrated Application Environments
- Buffered Messaging
- Asynchronous Notifications
- Enqueue Features
- Dequeue Features
- Propagation Features
- Message Format Transformation
- Other Oracle® Database Features
- Interfaces to Oracle® Database
- Oracle® Database Demonstrations

What Is Queuing?

When Web-based business applications communicate with each other, producer applications enqueue messages and consumer applications dequeue messages. At the most basic level of queuing, one producer enqueues one or more messages into one queue. Each message is dequeued and processed once by one of the consumers. A message stays in the queue until a consumer dequeues it or the message expires. A producer may stipulate a delay before the message is available to be consumed, and a time after which the message expires. Likewise, a consumer may wait when trying to dequeue a message if no message is available. An agent program or application may act as both a producer and a consumer.

Producers can enqueue messages in any sequence. Messages are not necessarily dequeued in the order in which they are enqueued. Messages can be enqueued without being dequeued.

At a slightly higher level of complexity, many producers enqueue messages into a queue, all of which are processed by one consumer. Or many producers enqueue messages, each message being processed by a different consumer depending on type and correlation identifier.
Enqueued messages are said to be propagated when they are reproduced on another queue, which can be in the same database or in a remote database.

Applications often use data in different formats. A transformation defines a mapping from one data type to another. The transformation is represented by a SQL function that takes the source data type as input and returns an object of the target data type. You can arrange transformations to occur when a message is enqueued, when it is dequeued, or when it is propagated to a remote subscriber.

Oracle® Database Leverages Oracle Database

Oracle® Database provides database-integrated message queuing functionality. It is built on top of Oracle Streams and leverages the functions of Oracle Database so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services and HTTP(S).

Because Oracle® Database is implemented in database tables, all operational benefits of high availability, scalability, and reliability are also applicable to queue data. Standard database features such as recovery, restart, and security are supported by Oracle® Database. You can use database development and management tools such as Oracle Enterprise Manager to monitor queues. Like other database tables, queue tables can be imported and exported.

Messages can be queried using standard SQL. This means that you can use SQL to access the message properties, the message history, and the payload. With SQL access you can also do auditing and tracking. All available SQL technology, such as indexes, can be used to optimize access to messages.

Note: Oracle® Database does not support data manipulation language (DML) operations on a queue table or an associated index-organized table (IOT), if any. The only supported means of modifying queue tables is through the supplied APIs. Queue tables and IOTs can become inconsistent and therefore effectively ruined, if DML operations are performed on them.

System-Level Access Control

Oracle® Database supports system-level access control for all queuing operations, allowing an application designer or DBA to designate users as queue administrators. A queue administrator can invoke Oracle® Database administrative and operational interfaces on any queue in the database. This simplifies administrative work because all administrative scripts for the queues in a database can be managed under one schema.

Queue-Level Access Control

Oracle® Database supports queue-level access control for enqueue and dequeue operations. This feature allows the application designer to protect queues created in one schema from applications running in other schemas. The application designer can grant only minimal access privileges to applications that run outside the queue schema.

Performance

Requests for service must be decoupled from supply of services to increase efficiency and enable complex scheduling. Oracle® Database exhibits high performance as measured by:
- Number of messages enqueued and dequeued each second
- Time to evaluate a complex query on a message warehouse
- Time to recover and restart the messaging process after a failure

**Scalability**
Queuing systems must be scalable. Oracle® Database exhibits high performance when the number of programs using the application increases, when the number of messages increases, and when the size of the message warehouse increases.

**Persistence for Security**
Messages that constitute requests for service must be stored persistently and processed exactly once for deferred execution to work correctly in the presence of network, computer, and application failures. Oracle® Database is able to meet requirements in the following situations:
- Applications do not have the resources to handle multiple unprocessed messages arriving simultaneously from external clients or from programs internal to the application.
- Communication links between databases are not available all the time or are reserved for other purposes. If the system falls short in its capacity to deal with these messages immediately, then the application must be able to store the messages until they can be processed.
- External clients or internal programs are not ready to receive messages that have been processed.

**Persistence for Scheduling**
Queuing systems must deal with priorities, and those priorities can change:
- Messages arriving later can be of higher priority than messages arriving earlier.
- Messages may wait for later messages before actions are taken.
- The same message may be accessed by different processes.
- Messages in a specific queue can become more important, and so must be processed with less delay or interference from messages in other queues.
- Messages sent to some destinations can have a higher priority than others.

**Persistence for Accessing and Analyzing Metadata**
Queuing systems must preserve message metadata, which can be as important as the payload data. For example, the time that a message is received or dispatched can be crucial for business and legal reasons. With the persistence features of Oracle® Database, you can analyze periods of greatest demand or evaluate the lag between receiving and completing an order.

**Object Type Support**
Oracle® Database supports enqueue, dequeue, and propagation operations where the queue type is an abstract datatype, ADT. It also supports enqueue and dequeue operations if the types are inherited types of a base ADT. Propagation between two queues where the types are inherited from a base ADT is not supported.

Oracle® Database also supports ANYDATA queues, which enable applications to enqueue different message types in a single queue.
If you plan to enqueue, propagate, or dequeue user-defined type messages, then each type used in these messages must exist at every database where the message can be enqueued in a queue. Some environments use directed networks to route messages through intermediate databases before they reach their destination. In such environments, the type must exist at each intermediate database, even if the messages of this type are never enqueued or dequeued at a particular intermediate database.

In addition, the following requirements must be met for such types:

- Type name must be the same at each database.
- Type must be in the same schema at each database.
- Shape of the type must match exactly at each database.
- Type cannot use inheritance or type evolution at any database.
- Type cannot contain varrays, nested tables, LOBs, rowids, or urowids.

The object identifier need not match at each database.

**Structured and XMLType Payloads**

You can use object types to structure and manage message payloads. Relational database systems in general have a richer typing system than messaging systems. Because Oracle Database is an object-relational database system, it supports traditional relational and user-defined types. Many powerful features are enabled as a result of having strongly typed content, such as content whose format is defined by an external type system. These include:

- Content-based routing
  Oracle® Database can examine the content and automatically route the message to another queue based on the content.
- Content-based subscription
  A publish and subscribe system is built on top of a messaging system so that you can create subscriptions based on content.
- Querying
  The ability to run queries on the content of the message enables message warehousing.

You can create queues that use the new opaque type, `XMLType`. These queues can be used to transmit and store messages that are XML documents. Using `XMLType`, you can do the following:

- Store any type of message in a queue
- Store documents internally as `CLOB` objects
- Store more than one type of payload in a queue
- Query `XMLType` columns using the operator `ExistsNode()`
- Specify the operators in subscriber rules or dequeue conditions

**Integration with Oracle Internet Directory**

You can register system events, user events, and notifications on queues with Oracle Internet Directory. System events are database startup, database shutdown, and system error events. User events include user log on and user log off, DDL statements (create, drop, alter), and DML statement triggers. Notifications on queues include OCI notifications, PL/SQL notifications, and e-mail notifications.
You can also create aliases for Oracle® Database agents in Oracle Internet Directory. These aliases can be specified while performing Oracle® Database enqueue, dequeue, and notification operations. This is useful when you do not want to expose an internal agent name.

**Support for Oracle Real Application Clusters**

Real Application Clusters can be used to improve Oracle® Database performance by allowing different queues to be managed by different instances. You do this by specifying different instance affinities (preferences) for the queue tables that store the queues. This allows queue operations (enqueue and dequeue) on different queues to occur in parallel.

If compatibility is set to Oracle8i release 8.1.5 or higher, then an application can specify the instance affinity for a queue table. When Oracle® Database is used with Real Application Clusters and multiple instances, this information is used to partition the queue tables between instances for queue-monitor scheduling as well as for propagation. The queue table is monitored by the queue monitors of the instance specified by the user. If the owner of the queue table is terminated, then the secondary instance or some available instance takes over the ownership for the queue table.

If an instance affinity is not specified, then the queue tables are arbitrarily partitioned among the available instances. This can result in pinging between the application accessing the queue table and the queue monitor monitoring it. Specifying the instance affinity prevents this, but does not prevent the application from accessing the queue table and its queues from other instances.

**Oracle® Database in Integrated Application Environments**

Oracle® Database provides the message management and communication needed for application integration. In an integrated environment, messages travel between the Oracle Database server, applications, and users, as shown in Figure 1–1.

Messages are exchanged between a client and the Oracle Database server or between two Oracle Database servers using Oracle Net Services. Oracle Net Services also propagates messages from one Oracle Database queue to another. Or, as shown in Figure 1–1, you can perform Oracle® Database operations over the Internet using HTTP(S). In this case, the client, a user or Internet application, produces structured XML messages. During propagation over the Internet, Oracle Database servers communicate using structured XML also.

Application integration also involves the integration of heterogeneous messaging systems. Oracle® Database seamlessly integrates with existing non-Oracle Database messaging systems like IBM WebSphere MQ through Messaging Gateway, thus allowing existing WebSphere MQ-based applications to be integrated into an Oracle® Database environment.

This section contains these topics:

- Oracle® Database Client/Server Communication
- Multiconsumer Dequeueing of the Same Message
- Oracle® Database Implementation of Workflows
- Oracle® Database Implementation of Publish/Subscribe
Oracle® Database Client/Server Communication

Client/Server applications usually run in a synchronous manner. Figure 1–2 demonstrates the asynchronous alternative using Oracle® Database. In this example Application B (a server) provides service to Application A (a client) using a request/response queue.

Application A enqueues a request into the request queue. In a different transaction, Application B dequeues and processes the request. Application B enqueues the result in the response queue, and in yet another transaction, Application A dequeues it.

The client need not wait to establish a connection with the server, and the server dequeues the message at its own pace. When the server is finished processing the message, there is no need for the client to be waiting to receive the result. A process of double-deferral frees both client and server.
Multiconsumer Dequeuing of the Same Message

A message can only be enqueued into one queue at a time. If a producer had to insert the same message into several queues in order to reach different consumers, then this would require management of a very large number of queues. To allow multiple consumers to dequeue the same message, Oracle® Database provides for queue subscribers and message recipients.

To allow for subscriber and recipient lists, the queue must reside in a queue table that is created with the multiple consumer option. Each message remains in the queue until it is consumed by all its intended consumers.

Queue Subscribers

Multiple consumers, which can be either applications or other queues, can be associated with a queue as subscribers. This causes all messages enqueued in the queue to be made available to be consumed by each of the queue subscribers. The subscribers to the queue can be changed dynamically without any change to the messages or message producers.

You cannot add subscriptions to single-consumer queues or exception queues. A consumer that is added as a subscriber to a queue is only able to dequeue messages that are enqueued after the subscriber is added. No two subscribers can have the same values for name, address, and protocol. At least one of these attributes must be different for two subscribers.

It cannot be known which subscriber will dequeue which message first, second, and so on, because there is no priority among subscribers. More formally, the order of dequeuing by subscribers is indeterminate.

Subscribers can also be rule-based. Similar in syntax to the WHERE clause of a SQL query, rules are expressed in terms of attributes that represent message properties or message content. These subscriber rules are evaluated against incoming messages, and those rules that match are used to determine message recipients.
In Figure 1–3, Application B and Application C each need messages produced by Application A, so a multiconsumer queue is specially configured with Application B and Application C as queue subscribers. Each receives every message placed in the queue.

**Figure 1–3 Communication Using a Multiconsumer Queue**

![Diagram of Communication Using a Multiconsumer Queue]

**Message Recipients**

A message producer can submit a list of recipients at the time a message is enqueued. This allows for a unique set of recipients for each message in the queue. The recipient list associated with the message overrides the subscriber list associated with the queue, if there is one. The recipients need not be in the subscriber list. However, recipients can be selected from among the subscribers.

A recipient can be specified only by its name, in which case the recipient must dequeue the message from the queue in which the message was enqueued. It can be specified by its name and an address with a protocol value of 0. The address should be the name of another queue in the same database or another installation of Oracle Database (identified by the database link), in which case the message is propagated to the specified queue and can be dequeued by a consumer with the specified name. If the recipient's name is NULL, then the message is propagated to the specified queue in the address and can be dequeued by the subscribers of the queue specified in the address. If the protocol field is nonzero, then the name and address are not interpreted by the system and the message can be dequeued by a special consumer.

Subscribing to a queue is like subscribing to a magazine: each subscriber is able to dequeue all the messages placed into a specific queue, just as each magazine subscriber has access to all its articles. Being a recipient, on the other hand, is like getting a letter: each recipient is a designated target of a particular message.

**Figure 1–4** shows how Oracle® Database can accommodate both kinds of consumers. Application A enqueues messages. Application B and Application C are subscribers. But messages can also be explicitly directed toward recipients like Application D, which may or may not be subscribers to the queue. The list of such recipients for a given message is specified in the enqueue call for that message. It overrides the list of subscribers for that queue.
Oracle® Database Implementation of Workflows

*Figure 1–5* illustrates the use of Oracle® Database for implementing a *workflow*, also known as a chained application transaction:

1. Application A begins a workflow by enqueuing Message 1.
2. Application B dequeues it, performs whatever activity is required, and enqueues Message 2.
3. Application C dequeues Message 2 and generates Message 3.
4. Application D, the final step in the workflow, dequeues it.

**Note:** Multiple producers can simultaneously enqueue messages aimed at different targeted recipients.
Applications at the top and bottom of the illustration are linked by arrows with three unlabeled queues in the middle. At top left Application A, Producer is linked with the leftmost queue by a downward-pointing arrow labeled Enqueue (Message 1). At bottom left, Application B, Consumer and Producer is linked to the leftmost queue with a downward-pointing arrow labeled Dequeue (Message 1) and to the middle queue with an upward-pointing arrow labeled Enqueue (Message 2). At top right, Application C, Producer and Consumer is linked to the middle queue by an upward-pointing arrow labeled Dequeue (Message 2) and to the rightmost queue with a downward-pointing arrow labeled Enqueue (Message 3). At bottom right, Application D, Consumer is linked to the rightmost queue with a downward-pointing arrow labeled Dequeue (Message 3).

**********************************************************************************************

**Note:** The contents of the messages 1, 2 and 3 can be the same or different. Even when they are different, messages can contain parts of the contents of previous messages.

The queues are used to buffer the flow of information between different processing stages of the business process. By specifying delay interval and expiration time for a message, a window of execution can be provided for each of the applications.

From a workflow perspective, knowledge of the volume and timing of message flows is a business asset quite apart from the value of the payload data. Oracle® Database helps you gain this knowledge by supporting the optional retention of messages for analysis of historical patterns and prediction of future trends.

**Oracle® Database Implementation of Publish/Subscribe**

A point-to-point message is aimed at a specific target. Senders and receivers decide on a common queue in which to exchange messages. Each message is consumed by only
one receiver. Figure 1–6 shows that each application has its own message queue, known as a single-consumer queue.

**Figure 1–6  Point-to-Point Messaging**

A **publish/subscribe** message can be consumed by multiple receivers, as shown in Figure 1–7. Publish/subscribe messaging has a wide dissemination mode called **broadcast** and a more narrowly aimed mode called **multicast**.

Broadcasting is like a radio station not knowing exactly who the audience is for a given program. The dequeuers are subscribers to multiconsumer queues. In contrast, multicast is like a magazine publisher who knows who the subscribers are. Multicast is also referred to as point-to-multipoint, because a single publisher sends messages to multiple receivers, called recipients, who may or may not be subscribers to the queues that serve as exchange mechanisms.

**Figure 1–7  Publish/Subscribe Mode**

Publish/subscribe describes a situation in which a publisher application enqueues messages to a queue anonymously (no recipients specified). The messages are then delivered to subscriber applications based on **rules** specified by each application. The rules can be defined on message properties, message data content, or both.

You can implement a publish/subscribe model of communication using Oracle® Database as follows:

1. Set up one or more queues to hold messages. These queues should represent an area or subject of interest. For example, a queue can be used to represent billed orders.

2. Set up a set of rule-based subscribers. Each subscriber can specify a rule which represents a specification for the messages that the subscriber wishes to receive. A null rule indicates that the subscriber wishes to receive all messages.

3. Publisher applications publish messages to the queue by invoking an enqueue call.

4. Subscriber applications can receive messages with a dequeue call. This retrieves messages that match the subscription criteria.

5. Subscriber applications can also use a listen call to monitor multiple queues for subscriptions on different queues. This is a more scalable solution in cases where a subscriber application has subscribed to many queues and wishes to receive messages that arrive in any of the queues.
6. Subscriber applications can also use the Oracle Call Interface (OCI) notification mechanism. This allows a push mode of message delivery. The subscriber application registers the queues (and subscriptions specified as subscribing agent) from which to receive messages. This registers a callback to be invoked when messages matching the subscriptions arrive.

Figure 1–8 illustrates the use of Oracle® Database for implementing a publish/subscribe relationship between publisher Application A and subscriber Applications B, C, and D:

- Application B subscribes with rule "priority = 1".
- Application C subscribes with rule "priority > 1".
- Application D subscribes with rule "priority = 3".

If Application A enqueues three messages with priorities 1, 2, and 3 respectively, then the messages will be delivered as follows:

- Application B receives a single message (priority 1).
- Application C receives two messages (priority 2, 3).
- Application D receives a single message (priority 3).

Buffered Messaging

Buffered messaging, a new feature in Oracle® Database 10g Release 2 (10.2), combines the rich functionality that this product has always offered with a much faster queuing
Buffered messaging is ideal for applications that do not require the reliability and transaction support of Oracle® Database persistent messaging. Buffered messaging is faster than persistent messaging, because its messages reside in shared memory. They are usually written to disk only when the total memory consumption of buffered messages approaches the available shared memory limit.

**Note:** The portion of a queue that stores buffered messages in memory is sometimes referred to as a buffered queue.

Message retention is not supported for buffered messaging. When using buffered messaging, Oracle recommends that you do one of the following:

- **Set parameter streams_pool_size**
  This parameter controls the size of shared memory available to Oracle® Database. If unspecified, up to 10% of the shared pool size may be allocated for the Oracle® Database pool from the database cache.

- **Turn on SGA autotuning**
  Oracle will automatically allocate the appropriate amount of memory from the SGA for Oracle® Database, based on Oracle® Database usage as well as usage of other components that use the SGA. Examples of such other components are buffer cache and library cache. If streams_pool_size is specified, it is used as the lower bound.

  **See Also:** "Setting Initialization Parameters Relevant to Streams" in *Oracle Streams Concepts and Administration*

This section contains the following topics:

- **Enqueuing Buffered Messages**
- **Dequeuing Buffered Messages**
- **Propagating Buffered Messages**
- **Flow Control**
- **Buffered Messaging with Real Application Clusters (RAC)**
- **Buffered Messaging Restrictions**
- **Error Handling**

**Enqueuing Buffered Messages**
Buffered and persistent messages use the same single-consumer or multiconsumer queues and the same administrative and operational interfaces. They are distinguished from each other by a delivery mode parameter, set by the application when enqueuing the message to an Oracle® Database queue.

  **See Also:** "Enqueuing Messages" on page 10-2

Recipient lists are supported for buffered messaging enqueue.

Buffered messaging is supported in all queue tables created with compatibility 8.1 or higher. Transaction grouping queues and array enqueues are not supported for buffered messages in this release. You can still use the array enqueue procedure to enqueue buffered messages, but the array size must be set to one.
Buffered messages can be queried using the AQ$Queue_Table_Name view. They appear with states IN--MEMORY or SPILLED.

**See Also:** "AQ$Queue_Table_Name: Messages in Queue Table" on page 9-4

The queue type for buffered messaging can be ADT, XML, ANYDATA, or RAW. For ADT types with LOB attributes, only buffered messages with null LOB attributes can be enqueued.

All ordering schemes available for persistent messages are also available for buffered messages, but only within each message class. Ordering among persistent and buffered messages enqueued in the same session is not currently supported.

**See Also:** "Priority and Ordering of Messages in Enqueuing" on page 1-19

Both enqueue and dequeue buffered messaging operations must be with IMMEDIATE visibility mode. Thus they cannot be part of another transaction. You cannot specify delay when enqueuing buffered messages.

**Dequeuing Buffered Messages**

Rule-based subscriptions are supported with buffered messaging. The procedure for adding subscribers is enhanced to allow an application to express interest in persistent messages only, buffered messages only, or both.

**See Also:** "Adding a Subscriber" on page 8-20

Array dequeue is not supported for buffered messaging, but you can still use the array dequeue procedure by setting array size to one message.

Dequeuing applications can choose to dequeue persistent messages only, buffered messages only, or both types. Visibility must be set to IMMEDIATE for dequeuing buffered messages. All of the following dequeue options are supported:

- Dequeue modes BROWSE, LOCK, REMOVE, and REMOVE_NO_DATA
- Navigation modes FIRST_MESSAGE and NEXT_MESSAGE
- Correlation identifier
- Dequeue condition
- Message identifier

**See Also:** "Dequeue Options" on page 10-13

**Propagating Buffered Messages**

Propagation of buffered messages is supported. A single propagation schedule serves both persistent and buffered messages. The DBA_QUEUE_SCHEDULES view displays statistics and error information.

**See Also:** "DBA_QUEUE_SCHEDULES: All Propagation Schedules" on page 9-3

Oracle Streams AQ deletes buffered messages once they are propagated to the remote sites. If the receiving site fails before these messages are consumed, then these
messages will be lost. The source site will not be able to re-send them. Duplicate delivery of messages is also possible.

**See Also:** "Buffered Messaging with Real Application Clusters (RAC)" on page 1-15

**Flow Control**

Oracle® Database implements a flow control system that prevents applications from flooding the shared memory with messages. If the number of unread messages enqueued by a message sender exceeds a system-determined limit, then message sender is blocked until one of the subscribers has read some of its messages. A message sender is identified by `sender_id.name` in the enqueue options. A sender blocked due to flow control on a queue does not affect other message senders.

Even with flow control, slow consumers of a multiconsumer queue can cause the number of messages stored in memory to grow without limit. If this happens, older messages are spilled to disk and removed from the Oracle® Database pool to free up memory. This ensures that the cost of disk access is paid by the slower consumers, and faster subscribers can proceed unhindered.

**Buffered Messaging with Real Application Clusters (RAC)**

An application can enqueue and dequeue buffered messages from any RAC instance as long as it uses password-based authentication to connect to the database. The structures required for buffered messaging are implemented on one RAC instance. The instance where the buffered messaging structures are implemented is the `OWNER_INSTANCE` of the queue table containing the queue. Enqueue and dequeue requests received at other instances are forwarded to the `OWNER_INSTANCE` over the interconnect. The `REMOTE_LISTENER` parameter in `listener.ora` must also be set to enable forwarding of buffered messaging requests to correct instance.

**See Also:**
- "ALL_QUEUE_TABLES: Queue Tables Queue Accessible to the Current User" on page 9-3 for more information on `OWNER_INSTANCE`
- "REMOTE_LISTENER" in *Oracle Database Reference* for more information on setting the `REMOTE_LISTENER` parameter

A service name is associated with each queue in RAC and displayed in the `DBA_QUEUES` and `USER_QUEUES` views. This service name always points to the instance with the most efficient access for buffered messaging, minimizing pinging between instances. OCI clients can use the service name for buffered messaging operations.

**See Also:** "DBA_QUEUES: All Queues in Database" on page 9-3 or "USER_QUEUES: Queues In User Schema" on page 9-3

Oracle recommends that you use buffered messaging with queue-to-queue propagation. This results in transparent failover when propagating messages to a destination RAC system. You do not need to re-point your database links if the primary Oracle® Database RAC instance fails.

**See Also:** "Support for Oracle Real Application Clusters" on page 1-5
Buffered Messaging Restrictions
The following Oracle® Database features are not currently supported for buffered messaging:

- Message retention
- Message delay
- Transaction grouping
- Array enqueue
- Array dequeue
- Message export and import
- Posting for subscriber notification
- Messaging Gateway

Error Handling
Retry count and retry delay are not supported for buffered messages. Message expiration is supported. When a buffered message has been in the queue beyond its expiration period, it is moved into the exception queue as a persistent message.

Asynchronous Notifications
Asynchronous notification allows clients to receive notifications of messages of interest. The client can use these notifications to monitor multiple subscriptions. The client need not be connected to the database to receive notifications regarding its subscriptions. Asynchronous notification is supported for buffered messages. The delivery mode of the message is available in the message descriptor of the notification descriptor.

Note: In releases before Oracle Database 10g Release 2 (10.2), the Oracle® Database notification feature was not supported for queues with names longer than 30 characters. This restriction no longer applies. The 24-character limit on names of user-generated queues still applies. See "Creating a Queue" on page 8-12.

The client specifies a callback function which is run for each message. Asynchronous notification cannot be used to invoke an executable, but it is possible for the callback function to invoke a stored procedure.

Clients can receive notifications procedurally using PL/SQL, Java Message Service (JMS), or OCI callback functions, or clients can receive notifications through e-mail or HTTP post. Clients can also specify the presentation for notifications as either RAW or XML.

For JMS queues, the dequeue is accomplished as part of the notification; explicit dequeue is not required. For RAW queues, clients can specify payload delivery; but they still must dequeue the message in REMOVE_NO_DATA mode. For all other persistent queues, the notification contains only the message properties; clients explicitly dequeue to receive the message.

Payload Delivery for RAW Queues
For RAW queues, Oracle® Database clients can now specify that the message payload be delivered along with its notification.
Reliable Notification
In earlier releases of Oracle® Database, message notifications were stored in shared memory and were lost if the instance failed. Clients can now specify persistent message notification. If a RAC instance fails, its notifications are delivered by another RAC node. If a standalone instance fails, its notifications are delivered when the instance restarts.

Note: Notification reliability refers only to server failures. If Oracle® Database is unable to deliver client notifications for any other reason, then the notifications are purged along with the client registration.

Designated Port Notification
Oracle® Database clients can now use the OCI subscription handle attribute OCI_ATTR_SUBSCR_PORTNO to designate the port at which notifications are delivered. This is especially useful for clients on a computer behind a firewall. The port for the listener thread can be designated before the first registration, using an attribute in the environment handle. The thread is started the first time an OCISubscriptionRegister is called. If the client attempts to start another thread on a different port using a different environment handle, then Oracle® Database returns an error.

Note: Designated port notification and IP address notification apply only to OCI clients.

See Also: "Publish-Subscribe Registration Functions in OCI" in Oracle Call Interface Programmer’s Guide

IPv6 Compliance and Designated IP Support
Oracle Streams AQ supports IPv6 and Oracle Streams AQ clients can use the OCI subscription handle attribute OCI_ATTR_SUBSCR_IPADDR to designate the IP address at which notifications are delivered. This is especially useful for clients on a computer that has multiple network interface cards or IP addresses. The IP address for the listener thread can be designated before the first registration using an attribute in the environment handle. The thread is started the first time an OCISubscriptionRegister is called. If the client attempts to start another thread on a different IP address using a different environment handle, Oracle Streams AQ returns an error. If no IP address is specified, Oracle Streams AQ will deliver notifications on all IP addresses of the computer the client is on.

Registration Timeout
In earlier releases of Oracle® Database, registrations for notification persisted until explicitly removed by the client or purged in case of extended client failure. In Oracle® Database 10g Release 2 (10.2) clients can register for a specified time, after which the registration is automatically purged.

When the registration is purged, Oracle® Database sends a notification to the client, so the client can invoke its callback and take any necessary action.

See Also: "AQ Registration Information Type” on page 2-3

Note: Notification reliability refers only to server failures. If Oracle® Database is unable to deliver client notifications for any other reason, then the notifications are purged along with the client registration.

Designated Port Notification
Oracle® Database clients can now use the OCI subscription handle attribute OCI_ATTR_SUBSCR_PORTNO to designate the port at which notifications are delivered. This is especially useful for clients on a computer behind a firewall. The port for the listener thread can be designated before the first registration, using an attribute in the environment handle. The thread is started the first time an OCISubscriptionRegister is called. If the client attempts to start another thread on a different port using a different environment handle, then Oracle® Database returns an error.

Note: Designated port notification and IP address notification apply only to OCI clients.

See Also: "Publish-Subscribe Registration Functions in OCI" in Oracle Call Interface Programmer’s Guide

IPv6 Compliance and Designated IP Support
Oracle Streams AQ supports IPv6 and Oracle Streams AQ clients can use the OCI subscription handle attribute OCI_ATTR_SUBSCR_IPADDR to designate the IP address at which notifications are delivered. This is especially useful for clients on a computer that has multiple network interface cards or IP addresses. The IP address for the listener thread can be designated before the first registration using an attribute in the environment handle. The thread is started the first time an OCISubscriptionRegister is called. If the client attempts to start another thread on a different IP address using a different environment handle, Oracle Streams AQ returns an error. If no IP address is specified, Oracle Streams AQ will deliver notifications on all IP addresses of the computer the client is on.

Registration Timeout
In earlier releases of Oracle® Database, registrations for notification persisted until explicitly removed by the client or purged in case of extended client failure. In Oracle® Database 10g Release 2 (10.2) clients can register for a specified time, after which the registration is automatically purged.

When the registration is purged, Oracle® Database sends a notification to the client, so the client can invoke its callback and take any necessary action.

See Also: "AQ Registration Information Type” on page 2-3

Note: Notification reliability refers only to server failures. If Oracle® Database is unable to deliver client notifications for any other reason, then the notifications are purged along with the client registration.
Purge on Notification
Clients can also register to receive only the first notification, after which the registration is automatically purged.

An example where purge on notification is useful is a client waiting for enqueues to start. In this case, only the first notification is useful; subsequent notifications provide no additional information. Previously, this client would be required to unregister once enqueuing started; now the registration can be configured to go away automatically.

Buffered Message Notification
Clients can register for notification of buffered messages. The registration requests apply to both buffered and persistent messages. The message properties delivered with the PL/SQL or OCI notification specify whether the message is buffered or persistent.

See Also:
- "Registering for Notification" on page 10-22 for more information on PL/SQL notification
- Appendix C, "OCI Examples", which appears only in the HTML version of this guide, for an example of OCI notification

Reliable notification is not supported.

Views on Registration
The dictionary views DBA_SUBSCR_REGISTRATIONS and USER_SUBSCR_REGISTRATIONS display the various registrations in the system. The diagnostic view GV$SUBSCR_REGISTRATION_STATS may be used to monitor notification statistics and performance.

Event-Based Notification
Event-based notifications are processed by a set of coordinator (EMNC) and subordinate processes (EXXX). The event notification load is distributed amongst these processes. These processes work on the system notifications in parallel, offering a capability to process a larger volume of notifications, a faster response time and lower shared memory use for staging notifications.

Notification Grouping by Time
Notification applications may register to receive a single notification for all events that occur within a specified time interval. Notification Clients may specify a start time for the notifications. Additionally, they must specify a time as the grouping class and the time interval as the grouping value. A repeat count may be used to limit the number of notifications delivered.

Clients can receive two types of grouping events, Summary or Last. A summary notification is a list of Message Identifiers of all the messages for the subscription. If last was specified as a grouping type, notification would have information about the last message in the notification interval. A count of the number of messages in the interval is also sent.
The registration interfaces in PLSQL and OCI allow for specification of the START_TIME, REPEAT_COUNT, GROUPING_CLASS, GROUPING_VALUE, GROUPING_TYPE in the AQ$_REGISTRATION_INFO and the OCI subscription Handle.

The notification descriptor received by the AQ notification client provides information about the group of message identifiers and the number of notifications in the group.

See Also:
- Oracle Database PL/SQL Packages and Types Reference
- Oracle Call Interface Programmer’s Guide

Enqueue Features

The following features apply to enqueuing messages:
- Enqueue an Array of Messages
- Correlation Identifiers
- Priority and Ordering of Messages in Enqueuing
- Message Grouping
- Sender Identification
- Time Specification and Scheduling

Enqueue an Array of Messages

When enqueuing messages into a queue, you can operate on an array of messages simultaneously, instead of one message at a time. This can improve the performance of enqueue operations. When enqueuing an array of messages into a queue, each message shares the same enqueue options, but each message can have different message properties. You can perform array enqueue operations using PL/SQL or OCI.

Array enqueuing is not supported for buffered messages in this release.

Correlation Identifiers

You can assign an identifier to each message, thus providing a means to retrieve specific messages at a later time.

Priority and Ordering of Messages in Enqueuing

You can specify the priority of an enqueued message and its exact position in the queue. This means that users can specify the order in which messages are consumed in three ways:

- A priority can be assigned to each message.
- A sort order specifies which properties are used to order all messages in a queue. This is set when the queue table is created and cannot be changed. You can choose to sort messages by priority, enqueue time, or commit time. The commit-time option, a new feature in Oracle® Database 10g Release 2 (10.2), orders messages by an approximate CSCN calculated for each transaction.

Commit-time ordering is useful when transactions are interdependent or when browsing the messages in a queue must yield consistent results.
Enqueue Features

See Also:

- "Commit-Time Queues" in Oracle Streams Concepts and Administration
- "Creating a Queue Table" on page 8-1 for more information on sort order

- A sequence deviation positions a message in relation to other messages.

Note: The sequence deviation feature is deprecated in 10g Release 2 (10.2).

If several consumers act on the same queue, then each consumer gets the first message that is available for immediate consumption. A message that is in the process of being consumed by another consumer is skipped.

Priority ordering of messages is achieved by specifying priority, enqueue time as the sort order. If priority ordering is chosen, then each message is assigned a priority at enqueue time by the enqueuing agent. At dequeue time, the messages are dequeued in the order of the priorities assigned. If two messages have the same priority, then the order in which they are dequeued is determined by the enqueue time. A first-in, first-out (FIFO) priority queue can also be created by specifying enqueue time, priority as the sort order of the messages.

Message Grouping

Messages belonging to one queue can be grouped to form a set that can only be consumed by one user at a time. This requires that the queue be created in a queue table that is enabled for message grouping. All messages belonging to a group must be created in the same transaction, and all messages created in one transaction belong to the same group.

This feature allows users to segment complex messages into simple messages. For example, messages directed to a queue containing invoices can be constructed as a group of messages starting with a header message, followed by messages representing details, followed by a trailer message.

Message grouping is also useful if the message payload contains complex large objects such as images and video that can be segmented into smaller objects.

Group message properties priority, delay, and expiration are determined solely by the message properties specified for the first message in a group, irrespective of which properties are specified for subsequent messages in the group.

The message grouping property is preserved across propagation. However, the destination queue where messages are propagated must also be enabled for transactional grouping. There are also some restrictions you must keep in mind if the message grouping property is to be preserved while dequeuing messages from a queue enabled for transactional grouping.

Sender Identification

Applications can mark the messages they send with a custom identification. Oracle® Database also automatically identifies the queue from which a message was dequeued. This allows applications to track the pathway of a propagated message or a string message within the same database.
Time Specification and Scheduling
Messages can be enqueued with an expiration that specifies the interval of time the message is available for dequeuing. The default for expiration is never. When a message expires, it is moved to an exception queue. Expiration processing requires that the queue monitor be running.

Dequeue Features
The following features apply to dequeuing messages:

- Concurrent Dequeues
- Dequeue Methods
- Dequeue Modes
- Dequeue an Array of Messages
- Message States
- Navigation of Messages in Dequeuing
- Waiting for Messages
- Retries with Delays
- Optional Transaction Protection
- Exception Queues

Concurrent Dequeues
When there are multiple processes dequeuing from a single-consumer queue or dequeuing for a single consumer on the multiconsumer queue, different processes skip the messages that are being worked on by a concurrent process. This allows multiple processes to work concurrently on different messages for the same consumer.

Dequeue Methods
A message can be dequeued using one of the following dequeue methods:

- Specifying a correlation identifier
  A correlation identifier is a user-defined message property. Multiple messages with the same correlation identifier can be present in a queue, which means that the ordering (enqueue order) between messages might not be preserved on dequeue calls.

- Specifying a message identifier
  A message identifier is a system-assigned value (of RAW datatype). Only one message with a given message identifier can be present in the queue.

- Specifying a dequeue condition
  A dequeue condition is expressed in terms of message properties or message content and is similar in syntax to the WHERE clause of a SQL query. Messages in the queue are evaluated against the condition, and messages that satisfy the given condition are returned. When a dequeue condition is used, the order of the messages dequeued is indeterminate, and the sort order of the queue is not honored.

- Default dequeue
  A default dequeue retrieves the first available message.
Dequeue Modes

A dequeue request can browse a message, remove it, or remove it with no data. If a message is browsed, then it remains available for further processing. If a message is removed or removed with no data, then it is no longer available for dequeue requests. Depending on the queue properties, a removed message can be retained in the queue table. A message is retained in the queue table after it has been consumed only if a retention time is specified for its queue.

The browse mode has three risks. First, there is no guarantee that the message can be dequeued again after it is browsed, because a dequeue call from a concurrent user might have removed the message. To prevent a viewed message from being dequeued by a concurrent user, you should view the message in the locked mode.

Second, your dequeue position in browse mode is automatically changed to the beginning of the queue if a nonzero wait time is specified and the navigating position reaches the end of the queue. If you repeat a dequeue call in the browse mode with the NEXT_MESSAGE navigation option and a nonzero wait time, then you can end up dequeuing the same message over and over again. Oracle recommends that you use a nonzero wait time for the first dequeue call on a queue in a session, and then use a zero wait time with the NEXT_MESSAGE navigation option for subsequent dequeue calls. If a dequeue call gets an “end of queue” error message, then the dequeue position can be explicitly set by the dequeue call to the beginning of the queue using the FIRST_MESSAGE navigation option, following which the messages in the queue can be browsed again.

Third, if the sort order of the queue is ENQ_TIME, PRIORITY, or a combination of these two, then results may not be repeatable from one browse to the next. If you must have consistent browse results, then you should use a commit-time queue.

See Also:
- "Commit-Time Queues“ in Oracle Streams Concepts and Administration
- "Creating a Queue Table“ on page 8-1

When a message is dequeued using REMOVE_NODATA mode, the payload of the message is not retrieved. This mode can be useful when the user has already examined the message payload, possibly by means of a previous BROWSE dequeue.

Dequeue an Array of Messages

When dequeuing messages from a queue, you can operate on an array of messages simultaneously, instead of one message at a time. This can improve the performance of dequeue operations. If you are dequeuing from a transactional queue, you can dequeue all the messages for a transaction with a single call, which makes application programming easier.

When dequeuing an array of messages from a queue, each message shares the same dequeue options, but each message can have different message properties. You can perform array enqueue and array dequeue operations using PL/SQL or OCI.

Array dequeuing is not supported for buffered messages in this release.
Message States

Multiple processes or operating system threads can use the same consumer name to dequeue concurrently from a queue. In that case Oracle® Database provides the first unlocked message that is at the head of the queue and is intended for the consumer. Unless the message identifier of a specific message is specified during dequeue, consumers can dequeue messages that are in the READY state.

A message is considered PROCESSED only when all intended consumers have successfully dequeued the message. A message is considered EXPIRED if one or more consumers did not dequeue the message before the EXPIRATION time. When a message has expired, it is moved to an exception queue.

Expired messages from multiconsumer queues cannot be dequeued by the intended recipients of the message. However, they can be dequeued in the REMOVE mode exactly once by specifying a NULL consumer name in the dequeue options.

---

Note: If the multiconsumer exception queue was created in a queue table with the compatible parameter set to 8.0, then expired messages can be dequeued only by specifying a message identifier.

Queues created in a queue table with compatible set to 8.0 (referred to in this guide as 8.0-style queues) are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

---

Beginning with Oracle® Database release 8.1.6, only the queue monitor removes messages from multiconsumer queues. This allows dequeuers to complete the dequeue operation by not locking the message in the queue table. Because the queue monitor removes messages that have been processed by all consumers from multiconsumer queues approximately once every minute, users can see a delay between when the messages have been completely processed and when they are physically removed from the queue.

Navigation of Messages in Dequeuing

You have several options for selecting a message from a queue. You can select the first message with the FIRST_MESSAGE navigation option. Alternatively, once you have selected a message and established its position in the queue, you can then retrieve the next message with the NEXT_MESSAGE navigation option.

The FIRST_MESSAGE navigation option performs a SELECT on the queue. The NEXT_MESSAGE navigation option fetches from the results of the SELECT run in the FIRST_MESSAGE navigation. Thus performance is optimized because subsequent dequeues need not run the entire SELECT again.

If the queue is enabled for transactional grouping, then the navigation options work in a slightly different way. If FIRST_MESSAGE is requested, then the dequeue position is still reset to the beginning of the queue. But if NEXT_MESSAGE is requested, then the position is set to the next message in the same transaction. Transactional grouping also offers a NEXT_TRANSACTION option. It sets the dequeue position to the first message of the next transaction.

Transactional grouping has no effect if you dequeue by specifying a correlation identifier or message identifier, or if you dequeue some of the messages of a transaction and then commit.
If you reach the end of the queue while using the NEXT_MESSAGE or NEXT_TRANSACTION option, and you have specified a nonzero wait time, then the navigating position is automatically changed to the beginning of the queue. If a zero wait time is specified, then you can get an exception when the end of the queue is reached.

**Waiting for Messages**

Oracle® Database allows applications to block on one or more queues waiting for the arrival of either a newly enqueued message or a message that becomes ready. You can use the DEQUEUE operation to wait for the arrival of a message in a single queue or the LISTEN operation to wait for the arrival of a message in more than one queue.

When the blocking DEQUEUE call returns, it returns the message properties and the message payload. When the blocking LISTEN call returns, it discloses only the name of the queue where a message has arrived. A subsequent DEQUEUE operation is needed to dequeue the message.

When there are messages for multiple agents in the agent list, LISTEN returns with the first agent for whom there is a message. To prevent one agent from starving other agents for messages, the application can change the order of the agents in the agent list.

**Note:** Applications can also perform a blocking dequeue on exception queues to wait for arrival of EXPIRED messages.

Applications can optionally specify a timeout of zero or more seconds to indicate the time that Oracle® Database must wait for the arrival of a message. The default is to wait forever until a message arrives in the queue. This removes the burden of continually polling for messages from the application, and it saves CPU and network resources because the application remains blocked until a new message is enqueued or becomes READY after its DELAY time.

An application that is blocked on a dequeue is either awakened directly by the enqueuer if the new message has no DELAY or is awakened by the queue monitor process when the DELAY or EXPIRATION time has passed. If an application is waiting for the arrival of a message in a remote queue, then the Oracle® Database propagator wakes up the blocked dequeuer after a message has been propagated.

**Retries with Delays**

If the transaction dequeuing a message from a queue fails, then it is regarded as an unsuccessful attempt to consume the message. Oracle® Database records the number of failed attempts to consume the message in the message history. Applications can query the RETRY_COUNT column of the queue table view to find out the number of unsuccessful attempts on a message. In addition, Oracle® Database allows the application to specify, at the queue level, the maximum number of retries for messages in the queue. The default value for maximum retries is 5. If the number of failed attempts to remove a message exceeds this number, then the message is moved to the exception queue and is no longer available to applications.
A bad condition can cause the transaction receiving a message to end. Oracle® Database allows users to hide the bad message for a specified retry delay interval, during which it is in the WAITING state. After the retry delay, the failed message is again available for dequeue. The Oracle® Database time manager enforces the retry delay property. The default value for retry delay is 0.

If multiple sessions are dequeuing messages from a queue simultaneously, then RETRY_COUNT information might not always be updated correctly. If session one dequeues a message and rolls back the transaction, then Oracle Streams AQ notes that the RETRY_COUNT information for this message must be updated. However RETRY_COUNT cannot be incremented until session one completes the rollback. If session two attempts to dequeue the same message after session one has completed the rollback but before it has incremented RETRY_COUNT, then the dequeue by session two succeeds. When session one attempts to increment RETRY_COUNT, it finds that the message is locked by session two and RETRY_COUNT is not incremented. A trace file is then generated in the USER_DUMP_DESTINATION for the instance with the following message:

Error on rollback: ORA-25263: no message in queue schema.qname with message ID ...

Note: Maximum retries and retry delay are not available with 8.0-style multiconsumer queues.

Queues created in a queue table with compatible set to 8.0 (referred to in this guide as 8.0-style queues) are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

Optional Transaction Protection

Enqueue and dequeue requests are usually part of a transaction that contains the requests, thereby providing the wanted transactional action. You can, however, specify that a specific request is a transaction by itself, making the result of that request immediately visible to other transactions. This means that messages can be made visible to the external world when the enqueue or dequeue statement is applied or after the transaction is committed.

Note: Transaction protection is not supported for buffered messaging.

Exception Queues

An exception queue is a repository for expired or unserviceable messages. Applications cannot directly enqueue into exception queues. Also, a multiconsumer exception queue cannot have subscribers associated with it. However, an application that intends to handle these expired or unserviceable messages can dequeue them exactly once from the exception queue using remove mode. The consumer name
specified while dequeuing should be null. Messages can also be dequeued from the exception queue by specifying the message identifier.

---

**Note:** Expired or unserviceable buffered messages are moved to an exception queue as persistent messages.

Messages intended for single-consumer queues, or for 8.0-style multiconsumer queues, can only be dequeued by their message identifiers once the messages have been moved to an exception queue.

Queues created in a queue table with `compatible` set to 8.0 (referred to in this guide as 8.0-style queues) are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

---

After a message has been moved to an exception queue, there is no way to identify which queue the message resided in before moving to the exception queue. If this information is important, then the application must save this information in the message itself.

The exception queue is a message property that can be specified during enqueue time. If an exception queue is not specified, then a default exception queue is used. The default exception queue is automatically created when the queue table is created.

A message is moved to an exception queue under the following conditions:

- It was not dequeued within the specified expiration interval.
  
  For a message intended for multiple recipients, the message is moved to the exception queue if one or more of the intended recipients was not able to dequeue the message within the specified expiration interval. The default expiration interval is never, meaning the messages does not expire.

- The message was dequeued successfully, but the application that dequeued it rolled back the transaction because of an error that arose while processing the message. If the message has been dequeued but rolled back more than the number of times specified by the retry limit, then the message is moved to the exception queue.
  
  For a message intended for multiple recipients, a separate retry count is kept for each recipient. The message is moved to the exception queue only when retry counts for all recipients of the message have exceeded the specified retry limit.

  The default retry limit is five for single-consumer queues and 8.1-style multiconsumer queues. No retry limit is supported for 8.0-style multiconsumer queues, which are deprecated in Oracle® Database 10g Release 2 (10.2).

---

**Note:** If a dequeue transaction fails because the server process dies (including `ALTER SYSTEM KILL SESSION`) or `SHUTDOWN ABORT` on the instance, then `RETRY_COUNT` is not incremented.

---

- The statement processed by the client contains a dequeue that succeeded but the statement itself was undone later due to an exception.
  
  If the dequeue procedure succeeds but the PL/SQL procedure raises an exception, then Oracle® Database increments the retry count of the message returned by the dequeue procedure.
The client program successfully dequeued a message but terminated before committing the transaction.

Propagation Features

Messages can be propagated from one queue to another, allowing applications to communicate with each other without being connected to the same database or to the same queue. The destination queue can be located in the same database or in a remote database.

Propagation enables you to fan out messages to a large number of recipients without requiring them all to dequeue messages from a single queue. You can also use propagation to combine messages from different queues into a single queue. This is known as compositing or funneling messages.

A message is marked as processed in the source queue immediately after the message has been propagated, even if the consumer has not dequeued the message at the remote queue. Similarly, when a propagated message expires at the remote queue, the message is moved to the exception queue of the remote queue, and not to the exception queue of the local queue. Oracle Database does not currently propagate the exceptions to the source queue.

To enable propagation, one or more subscribers are defined for the queue from which messages are to be propagated and a schedule is defined for each destination where messages are to be propagated from the queue.

Oracle Database automatically checks if the type of the remote queue is structurally equivalent to the type of the local queue within the context of the character sets in which they are created. Messages enqueued in the source queue are then propagated and automatically available for dequeuing at the destination queue or queues.

When messages arrive at the destination queues, sessions based on the source queue schema name are used for enqueuing the newly arrived messages into the destination queues. This means that you must grant schemas of the source queues enqueue privileges to the destination queues.

Propagation runs as an Oracle Scheduler job. A background process, the JOB_QUEUE_PROCESS will run the job. Propagation scheduling may be a dedicated process, running continuously and without end, or it may be event driven, in which case it runs only if there is a message to be propagated.

Oracle Database offers two kinds of propagation:

- Queue-to-dblink propagation
- Queue-to-queue propagation

Queue-to-dblink propagation delivers messages or events from the source queue to all subscribing queues at the destination database identified by the dblink.

A single propagation schedule is used to propagate messages to all subscribing queues. Hence any changes made to this schedule will affect message delivery to all the subscribing queues.
Queue-to-queue propagation delivers messages or events from the source queue to a specific destination queue identified on the dblink. This allows the user to have fine-grained control on the propagation schedule for message delivery.

This new propagation mode also supports transparent failover when propagating to a destination RAC system. With queue-to-queue propagation, you are no longer required to re-point a database link if the owner instance of the queue fails on RAC.

Oracle® Database provides detailed statistics about the messages propagated and the schedule itself. This information can be used to tune propagation schedules for best performance.

Remote Consumers
Consumers of a message in multiconsumer queues can be local or remote. Local consumers dequeue messages from the same queues into which the producer enqueued the messages. Local consumers have a name but no address or protocol in their agent descriptions.

Remote consumers dequeue from queues that are different from the queues where the messages were enqueued. Remote consumers fall into three categories:

- The address refers to a queue in the same database.
  In this case the consumer dequeues the message from a different queue in the same database. These addresses are of the form [schema].queue_name. If the schema is not specified, then the schema of the current user is used.

- The address refers to a queue in a different database.
  In this case the database must be reachable using database links and the protocol must be either NULL or 0. These addresses are of the form [schema].queue_name@dblink. If the schema is not specified, then the schema of the current user is used. If the database link does not have a domain name specified, then the default domain as specified by the DB_DOMAIN init.ora parameter is used.

- The address refers to a destination that can be reached by a third party protocol.
  You must refer to the documentation of the third party software to determine how to specify the address and the protocol database link and schedule propagation.

Propagation to Remote Subscribers
Oracle® Database validates the database link specified in a propagation schedule when the schedule runs, but not when the schedule is created. It is possible, therefore, to create a queue-to-dblink or queue-to-queue propagation before creating its associated database link. Also, the propagation schedule is not disabled if you remove the database link.

Oracle Streams AQ offers two kinds of propagation:

A) **Queue-to-dblink propagation** - specified by providing a (source) queue and (destination) databaselink. Messages from the source queue for any queues at the destination specified by the dblink will be handled by this propagation.

In this scenario, we cannot have multiple propagations from a source queue, with dblinks connecting to the same database. Thus(q1, dblink1) and (q1, dblink2) cannot co-exist if both dblinks connect to the same database. On the other hand (q1, dblink1) and (q2, dblink1) OR (q1, dblink1) and (q2, dblink2) can co-exist as source queues are different.

B) **Queue-to-queue propagation** - specified by providing a (source) queue, (destination) dblink and (destination) queue. Messages from the source queue for the
indicated queue at the destination dblink will be handled by this propagation. Here, either (q1, dblink1, dq1), (q1, dblink1, dq2) OR (q1, dblink1, dq1), (q1, dblink2, dq2) succeeds. This strategy works because the destination queues are different even though source queue is the same and dblink connects to the same database.

In this scenario, we cannot have multiple propagations between a source queue, destination queue, even if using different dblinks: (q1, dblink1, q2) and (q1, dblink2, q2) cannot co-exist, if dblink1 and dblink2 are pointing to the same database.

**Priority and Ordering of Messages in Propagation**

The delay, expiration, and priority parameters apply identically to both local and remote consumers in both queue-to-dblink and queue-to-queue propagation. Oracle® Database accounts for any delay in propagation by adjusting the delay and expiration parameters accordingly. For example, if expiration is set to one hour, and the message is propagated after 15 minutes, then the expiration at the remote queue is set to 45 minutes.

**Inboxes and Outboxes**

Figure 1–9 illustrates applications on different databases communicating using Oracle® Database. Each application has an inbox for handling incoming messages and an outbox for handling outgoing messages. Whenever an application enqueues a message, it goes into its outbox regardless of the message destination. Similarly, an application dequeues messages from its inbox no matter where the message originates.
Propagation Scheduling

A queue-to-dblink propagation schedule is defined for a pair of source and destination database links. A queue-to-queue propagation schedule is defined for a pair of source and destination queues. If a queue has messages to be propagated to several queues, then a schedule must be defined for each of the destination queues. With queue-to-dblink propagation, all schedules for a particular remote database have the same frequency. With queue-to-queue propagation, the frequency of each schedule can be adjusted independently of the others.

A schedule indicates the time frame during which messages can be propagated from the source queue. This time frame can depend on a number of factors such as network traffic, load at the source database, and load at the destination database. If the duration is unspecified, then the time frame is an infinite single window. If a window must be repeated periodically, then a finite duration is specified along with a NEXT_TIME function that defines the periodic interval between successive windows.
When a schedule is created, a job is automatically submitted to the job queue facility to handle propagation.

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue. You can also temporarily disable a schedule instead of dropping it. All administrative calls can be made irrespective of whether the schedule is active or not. If a schedule is active, then it takes a few seconds for the calls to be processed.

**Propagation of Messages with LOBs**

Large Objects can be propagated using Oracle® Database using two methods:

- **Propagation from RAW queues**

  In RAW queues the message payload is stored as a BLOB. This allows users to store up to 32KB of data when using the PL/SQL interface and as much data as can be contiguously allocated by the client when using OCI. This method is supported by all releases after 8.0.4 inclusive.

- **Propagation from object queues with LOB attributes**

  The user can populate the LOB and read from the LOB using Oracle Database LOB handling routines. The LOB attributes can be BLOBs or CLOBs (not NCLOBs). If the attribute is a CLOB, then Oracle® Database automatically performs any necessary character set conversion between the source queue and the destination queue. This method is supported by all releases from 8.1.3 inclusive.

**Note:** Payloads containing LOBs require users to grant explicit Select, Insert and Update privileges on the queue table for doing enqueues and dequeues.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

**Propagation Statistics**

Detailed runtime information about propagation is gathered and stored in the `DBA_QUEUE_SCHEDULES` view for each propagation schedule. This information can be used by queue designers and administrators to fix problems or tune performance. Similarly, errors reported by the view can be used to diagnose and fix problems. The view also describes additional information such as the session ID of the session handling the propagation and the process name of the job queue process handling the propagation.

For each schedule, detailed propagation statistics are maintained:

- Total number of messages propagated in a schedule
- Total number of bytes propagated in a schedule
- Maximum number of messages propagated in a window
- Maximum number of bytes propagated in a window
- Average number of messages propagated in a window
- Average size of propagated messages
- Average time to propagated a message
Propagation Error Handling

Propagation has built-in support for handling failures and reporting errors. For example, if the specified database link is invalid, if the remote database is unavailable, or if the remote queue is not enabled for enqueuing, then the appropriate error message is reported. Propagation uses a linear backoff scheme for retrying propagation from a schedule that encountered a failure.

If a schedule continuously encounters failures, then the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, then the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts is made, after which the schedule is automatically disabled.

---

**Note:** Once a retry attempt slips to the next propagation window, it will always do so; the exponential backoff scheme no longer governs retry scheduling. If the date function specified in the `next_time` parameter of `DBMS_AQADM.SCHEDULE_PROPAGATION` results in a short interval between windows, then the number of unsuccessful retry attempts can quickly reach 16, disabling the schedule.

---

When a schedule is disabled automatically due to failures, the relevant information is written into the alert log. A check for scheduling failures indicates:

- How many successive failures were encountered
- The error message indicating the cause for the failure
- The time at which the last failure was encountered

By examining this information, a queue administrator can fix the failure and enable the schedule. If propagation is successful during a retry, then the number of failures is reset to 0.

In some situations that indicate application errors in queue-to-dblink propagations, Oracle® Database marks messages as **UNDELIVERABLE** and logs a message in `alert.log`. Examples of such errors are when the remote queue does not exist or when there is a type mismatch between the source queue and the remote queue. The trace files in the `background_dump_dest` directory can provide additional information about the error.

When a new job queue process starts, it clears the mismatched type errors so the types can be reverified. If you have capped the number of job queue processes and propagation remains busy, then you might not want to wait for the job queue process to terminate and restart. Queue types can be reverified at any time using `DBMS_AQADM.VERIFY_QUEUE_TYPES`.

---

**Note:** When a type mismatch is detected in queue-to-queue propagation, propagation stops and throws an error. In such situations you must query the `DBA_SCHEDULES` view to determine the last error that occurred during propagation to a particular destination. The message is not marked as **UNDELIVERABLE**.

---

Propagation with Real Application Clusters

Propagation has support built-in for Oracle Real Application Clusters. It is transparent to the user and the queue administrator. The job that handles propagation is submitted to the same instance as the owner of the queue table where the queue resides.
If there is a failure at an instance and the queue table that stores the source queue is migrated to a different instance, then the propagation job is also migrated to the new instance. This minimizes pinging between instances and thus offers better performance.

The destination can be identified by a database link or by destination queue name. Specifying the destination database results in queue-to-dblink propagation. If you propagate messages to several queues in another database, then all queue-to-dblink propagations to that database have the same frequency. Specifying the destination queue name results in queue-to-queue propagation, a new feature in Oracle® Database 10g Release 2 (10.2). If you propagate messages to several queues in another database, then queue-to-queue propagation enables you to adjust the frequency of each schedule independently of the others. You can even enable or disable individual propagations.

This new queue-to-queue propagation mode also supports transparent failover when propagating to a destination RAC system. With queue-to-queue propagation, you are no longer required to re-point a database link if the owner instance of the queue fails on RAC.

See Also: "Scheduling a Queue Propagation" on page 8-24 for more information on queue-to-queue propagation

Propagation has been designed to handle any number of concurrent schedules. The number of job queue processes is limited to a maximum of 1000, and some of these can be used to handle jobs unrelated to propagation. Hence, propagation has built-in support for multitasking and load balancing.

The propagation algorithms are designed such that multiple schedules can be handled by a single job queue process. The propagation load on a job queue process can be skewed based on the arrival rate of messages in the different source queues.

If one process is overburdened with several active schedules while another is less loaded with many passive schedules, then propagation automatically redistributes the schedules so they are loaded uniformly.

Third-Party Support

If the protocol number for a recipient is in the range 128 - 255, then the address of the recipient is not interpreted by Oracle® Database and the message is not propagated by the Oracle® Database system. Instead, a third-party propagator can dequeue the message by specifying a reserved consumer name in the dequeue operation. The reserved consumer names are of the form AQ$$_Pprotocol_number$. For example, the consumer name AQ$_P128$ can be used to dequeue messages for recipients with protocol number 128. The list of recipients for a message with the specific protocol number is returned in the recipient_list message property on dequeue.

Another way for Oracle® Database to propagate messages to and from third-party messaging systems is through Messaging Gateway. Messaging Gateway dequeues messages from an Oracle® Database queue and guarantees delivery to supported third-party messaging systems. Messaging Gateway can also dequeue messages from these systems and enqueue them to an Oracle® Database queue.

Propagation Using HTTP

In Oracle Database 10g you can set up Oracle® Database propagation over HTTP and HTTPS (HTTP over SSL). HTTP propagation uses the Internet access infrastructure and requires that the Oracle® Database servlet that connects to the destination database be deployed. The database link must be created with the connect string
indicating the Web server address and port and indicating HTTP as the protocol. The source database must be created for running Java and XML. Otherwise, the setup for HTTP propagation is more or less the same as Oracle Net Services propagation.

**Message Format Transformation**

Applications often use data in different formats. A **transformation** defines a mapping from one Oracle data type to another. The transformation is represented by a SQL function that takes the source data type as input and returns an object of the target data type. Only one-to-one message transformations are supported.

To transform a message during enqueue, specify a mapping in the enqueue options. To transform a message during dequeue, specify a mapping either in the dequeue options or when you add a subscriber. A dequeue mapping overrides a subscriber mapping. To transform a message during propagation, specify a mapping when you add a subscriber.

You can create transformations by creating a single PL/SQL function or by creating an expression for each target type attribute. The PL/SQL function returns an object of the target type or the constructor of the target type. This representation is preferable for simple transformations or those not easily broken down into independent transformations for each attribute.

Creating a separate expression specified for each attribute of the target type simplifies transformation mapping creation and management for individual attributes of the destination type. It is useful when the destination type has many attributes.

As Figure 1–10 shows, queuing, routing, and transformation are essential building blocks to an integrated application architecture. The figure shows how data from the Out queue of a CRM application is routed and transformed in the integration hub and then propagated to the In queue of the Web application. The transformation engine maps the message from the format of the Out queue to the format of the In queue.

**Figure 1–10 Transformations in Application Integration**

**XML Data Transformation**

You can transform XML data using the `extract()` method supported on `XMLType` to return an object of `XMLType` after applying the supplied XPath expression. You can also create a PL/SQL function that transforms the `XMLType` object by applying an XSLT transformation to it, using the package `XSLPROCESSOR`.

**Other Oracle® Database Features**

This section contains these topics:

- Queue Monitor Coordinator
- Integration with Oracle Internet Directory
- Integration with Oracle Enterprise Manager
- Retention and Message History
Other Oracle® Database Features

- Cleaning Up Message Queues
- Tracking and Event Journals
- Non-repudiation
- Internet Integration

Queue Monitor Coordinator

Before 10g Release 1 (10.1), the Oracle® Database time manager process was called queue monitor (QMNn), a background process controlled by setting the dynamic init.ora parameter AQ_TM_PROCESSES. Beginning with 10g Release 1 (10.1), time management and many other background processes are automatically controlled by a coordinator-slave architecture called Queue Monitor Coordinator (QMNC). QMNC dynamically spawns slaves named qXXX depending on the system load. The slaves provide mechanisms for:

- Message delay
- Message expiration
- Retry delay
- Garbage collection for the queue table
- Memory management tasks for buffered messages

Because the number of processes is determined automatically and tuned constantly, you are saved the trouble of setting it with AQ_TM_PROCESSES.

Although it is no longer necessary to set init.ora parameter AQ_TM_PROCESSES, it is still supported. If you do set it (up to a maximum of 10), then QMNC still autotunes the number of processes. But you are guaranteed at least the set number of processes for persistent queues. Processes for a buffered queue and other Oracle Streams tasks, however, are not affected by this parameter.

---

**Note:** If you want to disable the Queue Monitor Coordinator, then you must set AQ_TM_PROCESSES = 0 in your pfile or spfile. Oracle strongly recommends that you do NOT set AQ_TM_PROCESSES = 0. If you are using Oracle Streams, setting this parameter to zero (which Oracle Database respects no matter what) can cause serious problems.

---

Integration with Oracle Internet Directory

Oracle Internet Directory is a native LDAPv3 directory service built on Oracle Database that centralizes a wide variety of information, including e-mail addresses, telephone numbers, passwords, security certificates, and configuration data for many types of networked devices. You can look up enterprise-wide queuing information—queues, subscriptions, and events—from one location, the Oracle Internet Directory. Refer to the Oracle Fusion Middleware Administrator’s Guide for Oracle Internet Directory for more information.

Integration with Oracle Enterprise Manager

You can use Oracle Enterprise Manager to:

- Create and manage queues, queue tables, propagation schedules, and transformations
Monitor your Oracle® Database environment using its topology at the database and queue levels, and by viewing queue errors and queue and session statistics.

**Retention and Message History**
The systems administrator specifies the retention duration to retain messages after consumption. Oracle® Database stores information about the history of each message, preserving the queue and message properties of delay, expiration, and retention for messages destined for local or remote receivers. The information contains the enqueue and dequeue times and the identification of the transaction that executed each request. This allows users to keep a history of relevant messages. The history can be used for tracking, data warehouse, and data mining operations, as well as specific auditing functions.

Message retention is not supported for buffered messaging.

**Cleaning Up Message Queues**
The Oracle® Database retention feature can be used to automatically clean up messages after the user-specified duration after consumption.

If messages are accidentally inserted into a queue for the wrong subscriber, you can dequeue them with the subscriber name or by message identifier. This consumes the messages, which are cleaned up after their retention time expires.

To clean up messages for a particular subscriber, you can remove the subscriber and add the subscriber again. Removing the subscriber removes all the messages for that subscriber.

**Tracking and Event Journals**
Retained messages can be related to each other to form sequences. These sequences represent event journals, which are often constructed by applications. Oracle® Database is designed to let applications create event journals automatically.

**Non-repudiation**
Oracle® Database maintains the entire history of information about a message along with the message itself. This information serves as proof of sending and receiving of messages and can be used for non-repudiation of the sender and non-repudiation of the receiver.

The following information is kept at enqueue for non-repudiation of the enqueuer:
- Oracle® Database agent doing the enqueue
- Database user doing the enqueue
- Enqueue time
- Transaction ID of the transaction doing enqueue

The following information is kept at dequeue for non-repudiation of the dequeuer:
- Oracle® Database agent doing dequeue
- Database user doing dequeue
- Dequeue time
- Transaction ID of the transaction doing dequeue

After propagation, the `ORIGINAL_MSGID` field in the destination queue of the propagation corresponds to the message ID of the source message. This field can be
used to correlate the propagated messages. This is useful for non-repudiation of the
dequeue of propagated messages.

Stronger non-repudiation can be achieved by enqueuing the digital signature of the
sender at the time of enqueue with the message and by storing the digital signature of
the dequeue at the time of dequeue.

**Internet Integration**

You can access Oracle® Database over the Internet by using **Simple Object Access
Protocol (SOAP)**. **Internet Data Access Presentation (IDAP)** is the SOAP specification
for Oracle® Database operations. IDAP defines the XML message structure for the
body of the SOAP request.

An IDAP message encapsulates the Oracle® Database request and response in XML.
IDAP is used to perform Oracle® Database operations such as enqueue, dequeue, send
notifications, register for notifications, and propagation over the Internet standard
transports—HTTP(s) and e-mail. In addition, IDAP encapsulates transactions, security,
transformation, and the character set ID for requests.

You can create an alias to an Oracle® Database agent in Oracle Internet Directory and
then use the alias in IDAP documents sent over the Internet to perform Oracle®
Database operations. Using aliases prevents exposing the internal name of the Oracle®
Database agent.

**Figure 1–11** shows the architecture for performing Oracle® Database operations over
HTTP. The major components are:

- Oracle® Database client program
- Web server/servlet runner hosting the Oracle® Database servlet
- Oracle Database server

The Oracle® Database client program sends XML messages (conforming to IDAP) to
the Oracle® Database servlet, which understands the XML message and performs
Oracle® Database operations. Any HTTP client, a Web browser for example, can be
used. The Web server/servlet runner hosting the Oracle® Database servlet,
Apache/Jserv or Tomcat for example, interprets the incoming XML messages. The
Oracle® Database servlet connects to the Oracle Database server and performs
operations on user queues.

**Note:** This feature is certified to work with Apache, along with the
Tomcat or Jserv servlet execution engines. However, the code does not
prevent the servlet from working with other Web server and servlet
execution engines that support Java Servlet 2.0 or higher interfaces.

**Figure 1–11  Architecture for Performing Oracle® Database Operations Using HTTP**
Interfaces to Oracle® Database

You can access Oracle® Database functionality through the following interfaces:

- PL/SQL using DBMS_AQ, DBMS_AQADM, and DBMS_AQELM
- Visual Basic using Oracle Objects for OLE
- Java Message Service (JMS) using the oracle.jms Java package
- Internet access using HTTP(S)

**Note:** The oracle.AQ Java package was deprecated in Oracle® Database 10g Release 1 (10.1). Oracle recommends that you migrate existing Java AQ applications to Oracle JMS and use Oracle JMS to design your future Java AQ applications.

**See Also:**

- Oracle Database PL/SQL Packages and Types Reference
- Online Help for Oracle Objects for OLE

Oracle® Database Demonstrations

Oracle® Database demos can be installed from the Oracle Database Companion CD. Once they are installed, you can find them in the $ORACLE_HOME/rdbms/demo directory. Refer to aqxmlREADME.txt and aqjmsREADME.txt in the demo directory for more information.

Table 1–1 lists and briefly describes the PL/SQL and OCI demos. Table 1–2 lists and briefly describes the JMS demos. Table 1–3 lists and briefly describes the XML demos.

**Table 1–1  Oracle® Database Demonstrations**

<table>
<thead>
<tr>
<th>Demo and Locations</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>aqdemo00.sql</td>
<td>Create users, message types, and tables</td>
</tr>
<tr>
<td>aqdemo01.sql</td>
<td>Create queue tables, queues, subscribers, and propagation schedule</td>
</tr>
<tr>
<td>aqdemo02.sql</td>
<td>Enqueue messages into input queue</td>
</tr>
<tr>
<td>aqdemo03.sql</td>
<td>Install dequeue procedures</td>
</tr>
<tr>
<td>aqdemo04.sql</td>
<td>Perform blocking dequeues</td>
</tr>
<tr>
<td>aqdemo05.sql</td>
<td>Perform listen for multiple agents</td>
</tr>
<tr>
<td>aqdemo06.sql</td>
<td>Clean up users, queue tables, queues, and subscribers in aqdemo00.sql to aqdemo05.sql</td>
</tr>
<tr>
<td>aqdemo07.sql</td>
<td>Enqueue and dequeue to XMLElement queue using XPATH expressions</td>
</tr>
<tr>
<td>aqdemo08.sql</td>
<td>Demonstrates server-to-server email notifications with default XML presentation</td>
</tr>
<tr>
<td>aqdemo09.sql</td>
<td>Set up queues and subscribers for array enqueue and dequeue (for OCI array demos also)</td>
</tr>
<tr>
<td>aqdemo10.sql</td>
<td>Array enqueue 10 messages</td>
</tr>
<tr>
<td>aqdemo11.sql</td>
<td>Array dequeue 10 messages</td>
</tr>
</tbody>
</table>
### Table 1–1 (Cont.) Oracle® Database Demonstrations

<table>
<thead>
<tr>
<th>Demo and Locations</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>aqdemo12.sql</td>
<td>Clean up queues and subscribers for array enqueue and dequeue (for OCI array demos also)</td>
</tr>
<tr>
<td>ociaqdemo00.c</td>
<td>Enqueue messages</td>
</tr>
<tr>
<td>ociaqdemo01.c</td>
<td>Perform blocking dequeues</td>
</tr>
<tr>
<td>ociaqdemo02.c</td>
<td>Perform listen for multiple agents</td>
</tr>
<tr>
<td>ociaqarrayenq.c</td>
<td>Array enqueue 10 messages</td>
</tr>
<tr>
<td>ociaqarraydeq.c</td>
<td>Array dequeue 10 messages</td>
</tr>
</tbody>
</table>

### Table 1–2 Oracle® Database JMS Demonstrations

<table>
<thead>
<tr>
<th>Demo and Locations</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>aqjmsREADME.txt</td>
<td>Describes the Oracle® Database Java API and JMS demos</td>
</tr>
<tr>
<td>aqjmsdmo.sql</td>
<td>Set up Oracle® Database JMS demos</td>
</tr>
<tr>
<td>aqjmsdemo01.java</td>
<td>Enqueue text messages and dequeue based on message properties</td>
</tr>
<tr>
<td>aqjmsdemo02.java</td>
<td>Message listener demo (enqueue messages)</td>
</tr>
<tr>
<td>aqjmsdemo03.java</td>
<td>Message listener demo (set up listener and dequeue messages)</td>
</tr>
<tr>
<td>aqjmsdemo04.java</td>
<td>Oracle type payload: dequeue on payload content</td>
</tr>
<tr>
<td>aqjmsdemo05.java</td>
<td>Queue browser example</td>
</tr>
<tr>
<td>aqjmsdemo06.java</td>
<td>Schedule propagation between queues in the database</td>
</tr>
<tr>
<td>aqjmsdemo07.java</td>
<td>Send and receive an ADT message containing XML data</td>
</tr>
<tr>
<td>aqjmsdemo08.java</td>
<td>JMS 1.1 domain unification demo</td>
</tr>
<tr>
<td>aqjmsdemo09.java</td>
<td>JMS bulk array enqueue and dequeue</td>
</tr>
<tr>
<td>aqjmsdemo10.java</td>
<td>ANYDATA messaging with JMS message types and ADT messages</td>
</tr>
<tr>
<td>aqjmsdrp.sql</td>
<td>Clean up AQ JMS demos</td>
</tr>
<tr>
<td>aqoradmo.sql</td>
<td>Set up Oracle® Database Java API demos</td>
</tr>
<tr>
<td>aqorademo01.java</td>
<td>Enqueue and dequeue RAW messages</td>
</tr>
<tr>
<td>aqorademo02.java</td>
<td>Enqueue and dequeue object type messages using ORAData interface</td>
</tr>
<tr>
<td>aqoradrp.sql</td>
<td>Clean up AQ Java API demos</td>
</tr>
<tr>
<td>aqjmskprb01.java</td>
<td>Enqueues and dequeues a message within the database</td>
</tr>
<tr>
<td>aqjmskprb01a.sql</td>
<td>Set up kprb driver demo</td>
</tr>
<tr>
<td>aqjmskprb01b.sql</td>
<td>Defines Java program aqjmskprb01.java as stored procedure</td>
</tr>
<tr>
<td>aqjmskprb01c.sql</td>
<td>Runs aqjmskprb01.java as stored procedure</td>
</tr>
<tr>
<td>aqjmskprb01d.sql</td>
<td>Clean up AQ kprb driver demo</td>
</tr>
</tbody>
</table>

### Table 1–3 Oracle® Database XML Demonstrations

<table>
<thead>
<tr>
<th>Demo and Locations</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>aqxmlREADME.txt</td>
<td>Describes the Internet access demos</td>
</tr>
<tr>
<td>aqxmldmo.sql</td>
<td>Create users, queue tables, and queues</td>
</tr>
<tr>
<td>Demo and Locations</td>
<td>Topic</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
</tr>
<tr>
<td>aqxml01.xml</td>
<td>AQXmlSend: Enqueue three messages to an ADT single-consumer queue with piggyback commit</td>
</tr>
<tr>
<td>aqxml02.xml</td>
<td>AQXmlReceive: Dequeue messages from ADT single-consumer queue with piggyback commit</td>
</tr>
<tr>
<td>aqxml03.xml</td>
<td>AQXmlPublish: Enqueue two messages to an ADT multiconsumer queue</td>
</tr>
<tr>
<td>aqxml04.xml</td>
<td>AQXmlReceive: Dequeue messages from an ADT (with LOB) multiconsumer queue</td>
</tr>
<tr>
<td>aqxml05.xml</td>
<td>AQXmlCommit: Commit previous operation</td>
</tr>
<tr>
<td>aqxml06.xml</td>
<td>AQXmlSend: Enqueue a message to a JMS TEXT single-consumer queue with piggyback commit</td>
</tr>
<tr>
<td>aqxml07.xml</td>
<td>AQXmlReceive: Dequeue messages from a JMS TEXT single-consumer queue with piggyback commit</td>
</tr>
<tr>
<td>aqxml08.xml</td>
<td>AQXmlPublish: Enqueue a JMS MAP message with recipient into multiconsumer queue</td>
</tr>
<tr>
<td>aqxml09.xml</td>
<td>AQXmlReceive: Dequeue JMS MAP messages from a multiconsumer queue</td>
</tr>
<tr>
<td>aqxml10.xml</td>
<td>AQXmlRollback: Roll back previous operation</td>
</tr>
<tr>
<td>aqxmlhtp.sql</td>
<td>HTTP propagation</td>
</tr>
<tr>
<td>AQDemoServlet.java</td>
<td>Servlet to post Oracle® Database XML files (for Jserv)</td>
</tr>
<tr>
<td>AQPropServlet.java</td>
<td>Servlet for Oracle® Database HTTP propagation</td>
</tr>
<tr>
<td>aqxmldrp.sql</td>
<td>Clean up AQ XML demo</td>
</tr>
</tbody>
</table>
This chapter describes the Oracle Streams Advanced Queuing (AQ) basic components. This chapter contains the following topics:

- Object Name
- Type Name
- AQ Agent Type
- AQ Recipient List Type
- AQ Agent List Type
- AQ Subscriber List Type
- AQ Registration Information List Type
- AQ Post Information List Type
- AQ Registration Information Type
- AQ Notification Descriptor Type
- AQ Message Properties Type
- AQ Post Information Type
- AQ$_NTFN_MSGID_ARRAY Type
- Enumerated Constants in the Oracle® Database Administrative Interface
- Enumerated Constants in the Oracle® Database Operational Interface
- AQ Background Processes

See Also:

- Chapter 8, "Oracle® Database Administrative Interface"
- Chapter 10, "Oracle® Database Operations Using PL/SQL"

**Object Name**

```
object_name := VARCHAR2
object_name := [schema_name.]name
```

This component names database objects. This naming convention applies to queues, queue tables, and object types.

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, then the current schema is assumed. The name must follow the
reserved character guidelines in Oracle Database SQL Language Reference. The schema name, agent name, and the object type name can each be up to 30 bytes long. However, queue names and queue table names can be a maximum of 24 bytes.

**Type Name**

```sql
type_name := VARCHAR2

| type_name := object_type | "RAW"
```

This component defines queue types. The maximum number of attributes in the object type is limited to 900.

To store payloads of type RAW, Oracle® Database creates a queue table with a LOB column as the payload repository. The size of the payload is limited to 32K bytes of data. Because LOB columns are used for storing RAW payload, the Oracle® Database administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the `storage_clause` parameter during queue table creation time.

---

**Note:** Payloads containing LOBs require users to grant explicit `Select`, `Insert` and `Update` privileges on the queue table for doing enqueues and dequeues.

---

**AQ Agent Type**

```sql
TYPE AQ$_AGENT IS OBJECT {
  name VARCHAR2(30),
  address VARCHAR2(1024),
  protocol NUMBER;
}
```

This component identifies a **producer** or a **consumer** of a **message**.

All consumers that are added as subscribers to a multiconsumer queue must have unique values for the AQ$_AGENT parameters. Two subscribers cannot have the same values for the NAME, ADDRESS, and PROTOCOL attributes for the AQ$_AGENT type. At least one of the three attributes must be different for two subscribers.

You can add subscribers by repeatedly using the `DBMS_AQADM.ADD_SUBSCRIBER` procedure up to a maximum of 1024 subscribers for a multiconsumer queue.

This type has three attributes:

- **name**
  This attribute specifies the name of a producer or consumer of a message. It can be the name of an application or a name assigned by an application. A queue can itself be an agent, enqueuing or dequeuing from another queue. The name must follow the reserved character guidelines in Oracle Database SQL Language Reference.

- **address**
  This attribute is interpreted in the context of protocol. If protocol is 0 (default), then address is of the form `[schema.]queue[@dblink].`

- **protocol**
  This attribute specifies the protocol to interpret the address and propagate the message. The default value is 0.
AQ Recipient List Type

    TYPE AQ$_RECIPIENT_LIST_T IS TABLE OF aq$_agent
    INDEX BY BINARY_INTEGER;

This component identifies the list of agents that receive a message.

AQ Agent List Type

    TYPE AQ$_AGENT_LIST_T IS TABLE OF aq$_agent
    INDEX BY BINARY INTEGER;

This component identifies the list of agents for DBMS_AQ.LISTEN to listen for.

AQ Subscriber List Type

    TYPE AQ$_SUBSCRIBER_LIST_T IS TABLE OF aq$_agent
    INDEX BY BINARY INTEGER;

This component identifies the list of subscribers that subscribe to this queue.

AQ Registration Information List Type

    TYPE AQ$_REG_INFO_LIST AS VARRAY(1024) OF sys.aq$_reg_info;

This component identifies the list of registrations to a queue.

AQ Post Information List Type

    TYPE AQ$_POST_INFO_LIST AS VARRAY(1024) OF sys.aq$_post_info;

This component identifies the list of anonymous subscriptions to which messages are posted.

AQ Registration Information Type

    TYPE SYS.AQ$_REG_INFO IS OBJECT {
        name                              VARCHAR2(128),
        namespace                         NUMBER,
        callback                          VARCHAR2(4000),
        context                           RAW(2000)  DEFAULT NULL,
        qosflags                          NUMBER,
        timeout                           NUMBER
        ntfn_grouping_class               NUMBER,
        ntfn_grouping_value               NUMBER    DEFAULT 600,
        ntfn_grouping_type                NUMBER,
        ntfn_grouping_start_time          TIMESTAMP WITH TIME ZONE,
        ntfn_grouping_repeat_count       NUMBER);

This component identifies a producer or a consumer of a message. Its attributes are described in the following list. Attributes qosflags and timeout are part of Oracle® Database 10g Release 2 (10.2) notification enhancements.
Table 2-1  AQ$_REG_INFO Type Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the name of the subscription. The subscription name is of the form schema.queue if the registration is for a single consumer queue or schema.queue:consumer_name if the registration is for a multiconsumer queues.</td>
</tr>
<tr>
<td>namespace</td>
<td>Specifies the namespace of the subscription. To receive notification from Oracle Streams AQ queues, the namespace must be DBMS_AQ.NAMESPACE_AQ. To receive notifications from other applications through DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.</td>
</tr>
<tr>
<td>callback</td>
<td>Specifies the action to be performed on message notification. For HTTP notifications, use <a href="http://www.company.com:8080">http://www.company.com:8080</a>. For e-mail notifications, use mailto://xyz@company.com. For raw message payload for the PLSQDCALLBACK procedure, use plsql://schema.procedure?PR=0. For user-defined type message payload converted to XML for the PLSQDCALLBACK procedure, use plsql://schema.procedure?PR=1.</td>
</tr>
<tr>
<td>context</td>
<td>Specifies the context that is to be passed to the callback function</td>
</tr>
<tr>
<td>qosflags</td>
<td>Can be set to one or more of the following values to specify the notification quality of service:</td>
</tr>
<tr>
<td></td>
<td>■ NTFN_QOS_RELIABLE - This value specifies that reliable notification is required. Reliable notifications persist across instance and database restarts.</td>
</tr>
<tr>
<td></td>
<td>■ NTFN_QOS_PAYLOAD - This value specifies that payload delivery is required. It is supported only for client notification and only for RAW queues.</td>
</tr>
<tr>
<td></td>
<td>■ NTFN_QOS_PURGE_ON_NTFN - This value specifies that the registration is to be purged automatically when the first notification is delivered to this registration location.</td>
</tr>
<tr>
<td>ntfn_grouping_class</td>
<td>Currently, only the following flag can be set to specify criterion for grouping. The default value will be 0. If ntfn_grouping_class is 0, all other notification grouping attributes must be 0.</td>
</tr>
<tr>
<td></td>
<td>■ NTFN_GROUPING_CLASS_TIME - Notifications grouped by time, that is, the user specifies a time value and a single notification gets published at the end of that time.</td>
</tr>
<tr>
<td>ntfn_grouping_value</td>
<td>Time-period of grouping notifications specified in seconds, meaning the time after which grouping notification would be sent periodically until ntfn_grouping_repeat_count is exhausted.</td>
</tr>
<tr>
<td>ntfn_grouping_type</td>
<td>■ NTFN_GROUPING_TYPE_SUMMARY - Summary of all notifications that occurred in the time interval. (Default)</td>
</tr>
<tr>
<td></td>
<td>■ NTFN_GROUPING_TYPE_LAST - Last notification that occurred in the interval.</td>
</tr>
<tr>
<td>ntfn_grouping_start_time</td>
<td>Notification grouping start time. Notification grouping can start from a user-specified time that should a valid timestamp with time zone. If ntfn_grouping_start_time is not specified when using grouping, the default is to current timestamp with time zone.</td>
</tr>
</tbody>
</table>
This component specifies the Oracle® Database descriptor received by Oracle® Database PL/SQL callbacks upon notification. It has the following attributes:

### AQ Notification Descriptor Type

```sql
TYPE SYS.AQ$_DESCRIPTOR IS OBJECT (
    queue_name      VARCHAR2(61),
    consumer_name   VARCHAR2(30),
    msg_id          RAW(16),
    msg_prop        MSG_PROP_T,
    gen_desc        AQ$_NTFN_DESCRIPTOR,
   msgid_array     SYS.AQ$_NTFN_MSGID_ARRAY,
    ntfnsRecdInGrp  NUMBER);
```

Table 2–2  AQ$_DESCRIPTOR Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntfn_grouping_repeat_count</td>
<td>Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. The ntfn_grouping_repeat_count, if not specified, will default to</td>
</tr>
<tr>
<td></td>
<td>- NTFN_GROUPING_FOREVER - Keep sending grouping notifications forever.</td>
</tr>
</tbody>
</table>

### AQ Message Properties Type

The message properties type `msg_prop_t` has the following components:

```sql
TYPE AQ$_MSG_PROP_T IS OBJECT{
    priority         number,
    delay            number,
    expiration       number,
    correlation      varchar2(128),
    attempts         number,
    recipient_list   aq$_recipient_list_t,
    exception_queue  varchar2(51),
    enqueue_time     date,
    state            number,
    sender_id        aq$_agent,
    original_misgid  raw(16),
    delivery_mode    number);
```
The timeout specifications type AQ$_NTFN_DESCRIPTOR has a single component:

```sql
TYPE AQ$_NTFN_DESCRIPTOR IS OBJECT(
    NTFN_FLAGS   number);
```

NTFN_FLAGS is set to 1 if the notifications are already removed after a stipulated timeout; otherwise the value is 0.

### AQ Post Information Type

```sql
TYPE SYS.AQ$_POST_INFO IS OBJECT (  
    name       VARCHAR2(128),
    namespace  NUMBER,
    payload    RAW(2000));
```

This component specifies anonymous subscriptions to which you want to post messages. It has three attributes:

- **name**
  This attribute specifies the name of the anonymous subscription to which you want to post.

- **namespace**
  This attribute specifies the namespace of the anonymous subscription. To receive notifications from other applications using `DBMS_AQ.POST` or `OCISubscriptionPost()`, the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`.

- **payload**
  This attribute specifies the payload to be posted to the anonymous subscription. The default is NULL.

### AQ$_NTFN_MSGID_ARRAY Type

```sql
TYPE SYS.AQ$_NTFN_MSGID_ARRAY  
    AS VARRAY(1073741824)OF RAW(16);
```

This component is for storing grouping notification data for AQ namespace, value $2^{30}$ which is the max varray size.

### Enumerated Constants in the Oracle® Database Administrative Interface

When enumerated constants such as INFINITE, TRANSACTIONAL, and NORMAL_QUEUE are selected as values, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with `DBMS_AQADM`. For example:

```sql
DBMS_AQADM.NORMAL_QUEUE
```

Table 2–3 lists the enumerated constants in the Oracle® Database administrative interface.
When using enumerated constants such as BROWSE, LOCKED, and REMOVE, the PL/SQL constants must be specified with the scope of the packages defining them. All types associated with the operational interfaces must be prepended with DBMS_AQ. For example:

DBMS_AQ.BROWSE

Table 2–4 lists the enumerated constants in the Oracle® Database operational interface.

### Enumerated Constants in the Oracle® Database Operational Interface

When using enumerated constants such as BROWSE, LOCKED, and REMOVE, the PL/SQL constants must be specified with the scope of the packages defining them. All types associated with the operational interfaces must be prepended with DBMS_AQ. For example:

DBMS_AQ.BROWSE

Table 2–4 lists the enumerated constants in the Oracle® Database operational interface.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>retention</td>
<td>0, 1, 2,...INFINITE</td>
</tr>
<tr>
<td>message_grouping</td>
<td>TRANSACTIONAL, NONE</td>
</tr>
<tr>
<td>queue_type</td>
<td>NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>BUFFERED, PERSISTENT, PERSISTENT_OR_BUFFERED</td>
</tr>
</tbody>
</table>

**Note:** Nonpersistent queues are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that you use buffered messaging instead.

### Enumerated Constants in the Oracle® Database Administrative Interface

When using enumerated constants such as BROWSE, LOCKED, and REMOVE, the PL/SQL constants must be specified with the scope of the packages defining them. All types associated with the operational interfaces must be prepended with DBMS_AQ. For example:

DBMS_AQ.BROWSE

Table 2–4 lists the enumerated constants in the Oracle® Database administrative interface.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>IMMEDIATE, ON_COMMIT</td>
</tr>
<tr>
<td>dequeue_mode</td>
<td>BROWSE, LOCKED, REMOVE, REMOVE_NODATA</td>
</tr>
<tr>
<td>navigation</td>
<td>FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION</td>
</tr>
<tr>
<td>state</td>
<td>WAITING, READY, PROCESSED, EXPIRED</td>
</tr>
<tr>
<td>wait</td>
<td>FOREVER, NO_WAIT</td>
</tr>
<tr>
<td>delay</td>
<td>NO_DELAY</td>
</tr>
<tr>
<td>expiration</td>
<td>NEVER</td>
</tr>
<tr>
<td>namespace</td>
<td>NAMESPACE_AQ, NAMESPACE_ANONYMOUS</td>
</tr>
<tr>
<td>delivery_mode</td>
<td>BUFFERED, PERSISTENT, PERSISTENT_OR_BUFFERED</td>
</tr>
<tr>
<td>quosflags</td>
<td>NTFN_QOS_RELIABLE, NTFN_QOS_PAYLOAD, NTFN_QOS_PURGE_ON_NTFN</td>
</tr>
<tr>
<td>ntfn_grouping_class</td>
<td>NFTN_GROUPING_CLASS_TIME</td>
</tr>
<tr>
<td>ntfn_grouping_type</td>
<td>NFTN_GROUPING_TYPE_SUMMARY, NFTN_GROUPING_TYPE_LAST</td>
</tr>
<tr>
<td>ntfn_grouping_repeat_count</td>
<td>NTFN_GROUPING_FOREVER</td>
</tr>
</tbody>
</table>

### AQ Background Processes

- **Queue Monitor Processes**
- **Job Queue Processes**
**AQ Background Processes**

A number of Streams AQ or Streams tasks are executed in the background. These include converting messages with `DELAY` specified into the `READY` state, expiring messages, moving messages to exception queues, spilling and recovering of buffered messages, and similar operations.

These are executed by a set of AQ background process. These include a coordinator process, name QMNC (link), which dynamically spawns subordinate processes Qxx as needed. The number of subordinate processes is determined automatically and tuned constantly.

It is no longer necessary to set `AQ_TM_PROCESSES` when Oracle Streams AQ or Streams is used. If a value is specified, that value is taken into account when starting the Qxx processes. However, the number of Qxx processes can be different from what was specified by `AQ_TM_PROCESSES`.

QMNC only runs when you use queues and create new queues. It affects Streams Replication and Messaging users.

No separate API is needed to disable or enable the background processes. This is controlled by setting `AQ_TM_PROCESSES` to zero or nonzero. Oracle recommends, however, that you leave the `AQ_TM_PROCESSES` parameter unspecified and let the system autotune.

---

**Note:** If you want to disable the Queue Monitor Coordinator, then you must set `AQ_TM_PROCESSES = 0` in your pfile or spfile. Oracle strongly recommends that you do NOT set `AQ_TM_PROCESSES = 0`. If you are using Oracle Streams, then setting this parameter to zero (which Oracle Database respects no matter what) can cause serious problems.

---

**Job Queue Processes**

Propagation and PL/SQL notifications are handled by job queue (Jnnn) processes. The parameter `JOB_QUEUE_PROCESSES` no longer needs to be specified. The database scheduler automatically starts the job queue processes that are needed for the propagation and notification jobs.
This chapter describes the different language options and elements you must work with and issues to consider in preparing your Oracle Streams Advanced Queuing (AQ) application environment.

**Note:** Java package oracle.AQ was deprecated in 10g Release 1 (10.1). Oracle recommends that you migrate existing Java AQ applications to Oracle JMS (or other Java APIs) and use Oracle JMS (or other Java APIs) to design your future Java AQ applications.

This chapter contains these topics:
- Programmatic Interfaces for Accessing Oracle® Database
- Using PL/SQL to Access Oracle® Database
- Using OCI to Access Oracle® Database
- Using OCCI to Access Oracle® Database
- Using Visual Basic (OO4O) to Access Oracle® Database
- Using Oracle Java Message Service (OJMS) to Access Oracle® Database
- Using Oracle® Database XML Servlet to Access Oracle® Database
- Comparing Oracle® Database Programmatic Interfaces

### Programmatic Interfaces for Accessing Oracle® Database

Table 3–1 lists Oracle® Database programmatic interfaces, functions supported in each interface, and syntax references.

<table>
<thead>
<tr>
<th>Language</th>
<th>Precompiler or Interface Program</th>
<th>Functions Supported</th>
<th>Syntax References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/SQL</td>
<td>DBMS_AQADM and DBMS_AQ Packages</td>
<td>Administrative and operational</td>
<td>Oracle Database PL/SQL Packages and Types Reference</td>
</tr>
<tr>
<td>C</td>
<td>Oracle Call Interface (OCI)</td>
<td>Operational only</td>
<td>Oracle Call Interface Programmer’s Guide</td>
</tr>
</tbody>
</table>
Using PL/SQL to Access Oracle® Database

The PL/SQL packages DBMS_AQADM and DBMS_AQ support access to Oracle® Database administrative and operational functions using the native Oracle® Database interface. These functions include:

- Create queue, queue table, nonpersistent queue, multiconsumer queue/topic, RAW message, or message with structured data
- Get queue table, queue, or multiconsumer queue/topic
- Alter queue table or queue/topic
- Drop queue/topic
- Start or stop queue/topic
- Grant and revoke privileges
- Add, remove, or alter subscriber
- Add, remove, or alter an Oracle® Database Internet agent
- Grant or revoke privileges of database users to Oracle® Database Internet agents
- Enable, disable, or alter propagation schedule
- Enqueue messages to single consumer queue (point-to-point model)
- Publish messages to multiconsumer queue/topic (publish/subscribe model)
- Subscribe for messages in multiconsumer queue
- Browse messages in a queue
- Receive messages from queue/topic
- Register to receive messages asynchronously
- Listen for messages on multiple queues/topics
- Post messages to anonymous subscriptions
- Bind or unbind agents in a Lightweight Directory Access Protocol (LDAP) server
- Add or remove aliases to Oracle® Database objects in a LDAP server

See Also:  Oracle Database PL/SQL Packages and Types Reference for detailed documentation of DBMS_AQADM and DBMS_AQ, including syntax, parameters, parameter types, return values, and examples

Available PL/SQL DBMS_AQADM and DBMS_AQ functions are listed in detail in Table 3–2 through Table 3–9.
Using OCI to Access Oracle® Database

OCI provides an interface to Oracle® Database functions using the native Oracle® Database interface.

An OCI client can perform the following actions:

■ Enqueue messages
■ Dequeue messages
■ Listen for messages on sets of queues
■ Register to receive message notifications

In addition, OCI clients can receive asynchronous notifications for new messages in a queue using OCISubscriptionRegister.

See Also: "OCI and Advanced Queuing" and "Publish-Subscribe Notification" in Oracle Call Interface Programmer’s Guide for syntax details

Oracle Type Translator

For queues with user-defined payload types, the Oracle type translator must be used to generate the OCI/OCCI mapping for the Oracle type. The OCI client is responsible for freeing the memory of the Oracle® Database descriptors and the message payload.

Using OCCI to Access Oracle® Database

C++ applications can use OCCI, which has a set of Oracle® Database interfaces that enable messaging clients to access Oracle® Database. OCCI AQ supports all the operational functions required to send/receive and publish/subscribe messages in a message-enabled database. Synchronous and asynchronous message consumption is available, based on a message selection rule.

See Also: "Oracle Streams Advanced Queuing” in Oracle C++ Call Interface Programmer’s Guide

Using Visual Basic (OO4O) to Access Oracle® Database

Visual Basic (OO4O) supports access to Oracle® Database operational functions using the native Oracle® Database interface.

These functions include the following:

■ Create a connection, RAW message, or message with structured data
■ Enqueue messages to a single-consumer queue (point-to-point model)
■ Publish messages to a multiconsumer queue/topic (publish/subscribe model)
■ Browse messages in a queue
■ Receive messages from a queue/topic
■ Register to receive messages asynchronously
Java Message Service (JMS) is a messaging standard defined by Sun Microsystems, Oracle, IBM, and other vendors. JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.

Oracle Java Message Service (OJMS) provides a Java API for Oracle® Database based on the JMS standard. OJMS supports the standard JMS interfaces and has extensions to support administrative operations and other features that are not a part of the standard.

Standard JMS features include:

- Point-to-point model of communication using queues
- Publish/subscribe model of communication using topics
- ObjectMessage, StreamMessage, TextMessage, BytesMessage, and MapMessage message types
- Asynchronous and synchronous delivery of messages
- Message selection based on message header fields or properties

Oracle JMS extensions include:

- Administrative API to create queue tables, queues and topics
- Point-to-multipoint communication using recipient lists for topics
- Message propagation between destinations, which allows the application to define remote subscribers
- Support for transactional sessions, enabling JMS and SQL operations in one transaction
- Message retention after messages have been dequeued
- Message delay, allowing messages to be made visible after a certain delay
- Exception handling, allowing messages to be moved to exception queues if they cannot be processed successfully
- Support for AdtMessage

These are stored in the database as Oracle objects, so the payload of the message can be queried after it is enqueued. Subscriptions can be defined on the contents of these messages as opposed to just the message properties.

Topic browsing

This allows durable subscribers to browse through the messages in a publish/subscribe (topic) destination. It optionally allows these subscribers to purge the browsed messages, so they are no longer retained by Oracle® Database for that subscriber.
Accessing Standard and Oracle JMS Applications

Standard JMS interfaces are in the javax.jms package. Oracle JMS interfaces are in the oracle.jms package. You must have EXECUTE privilege on the DBMS_AQIN and DBMS_AQJMS packages to use the Oracle JMS interfaces. You can also acquire these rights through the AQ_USER_ROLE or the AQ_ADMINSTRATOR_ROLE. You also need the appropriate system and queue or topic privileges to send or receive messages.

Because Oracle JMS uses Java Database Connectivity (JDBC) to connect to the database, its applications can run outside the database using the JDBC OCI driver or JDBC thin driver.

Using JDBC OCI Driver or JDBC Thin Driver

To use JMS with clients running outside the database, you must include the appropriate JDBC driver, Java Naming and Directory Interface (JNDI) jar files, and Oracle® Database jar files in your CLASSPATH.

For JDK 1.3.x and higher, include the following in the CLASSPATH:

$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jlib/orail8n.jar
$ORACLE_HOME/jlib/jta.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/rdbms/jlib/aqapi13.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar

For JDK 1.2 include the following in the CLASSPATH:

$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jlib/orail8n.jar
$ORACLE_HOME/jlib/jta.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/rdbms/jlib/aqapi12.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar

Using Oracle Server Driver in JServer

If your application is running inside the JServer, then you should be able to access the Oracle JMS classes that have been automatically loaded when the JServer was installed. If these classes are not available, then you must load jmscommon.jar followed by aqapi.jar using the $ORACLE_HOME/rdbms/admin/initjms SQL script.

Using Oracle® Database XML Servlet to Access Oracle® Database

You can use Oracle® Database XML servlet to access Oracle® Database over HTTP using Simple Object Access Protocol (SOAP) and an Oracle® Database XML message format called Internet Data Access Presentation (IDAP).

See Also:

- Oracle Streams Advanced Queuing Java API Reference
Comparing Oracle® Database Programmatic Interfaces

Using the Oracle® Database servlet, a client can perform the following actions:

- Send messages to single-consumer queues
- Publish messages to multiconsumer queues/topics
- Receive messages from queues
- Register to receive message notifications

See Also: "Deploying the Oracle® Database XML Servlet" on page 6-4 for more information on the Oracle® Database XML servlet

Comparing Oracle® Database Programmatic Interfaces

Available functions for the Oracle® Database programmatic interfaces are listed by use case in Table 3–2 through Table 3–9. Use cases are described in Chapter 8 through Chapter 10 and Chapter 12 through Chapter 15.

Oracle® Database Administrative Interfaces

Table 3–2 lists the equivalent Oracle® Database administrative functions for the PL/SQL and Java (JMS) programmatic interfaces.

### Table 3–2 Comparison of Oracle® Database Programmatic Interfaces: Administrative Interface

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>Java (JMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a connection factory</td>
<td>N/A</td>
<td>AQjmsFactory.getQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConnectionFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AQjmsFactory.getTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConnectionFactory</td>
</tr>
<tr>
<td>Register a ConnectionFactory in an LDAP server</td>
<td>N/A</td>
<td>AQjmsFactory.register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConnectionFactory</td>
</tr>
<tr>
<td>Create a queue table</td>
<td>DBMS_AQADM.CREATE_QUEUE_TABLE</td>
<td>AQjmsSession.createQueueTable</td>
</tr>
<tr>
<td>Get a queue table</td>
<td>Use schema.queue_table_name</td>
<td>AQjmsSession.getQueueTable</td>
</tr>
<tr>
<td>Alter a queue table</td>
<td>DBMS_AQADM.ALTER_QUEUE_TABLE</td>
<td>AQQueueTable.alter</td>
</tr>
<tr>
<td>Drop a queue table</td>
<td>DBMS_AQADM.DROP_QUEUE_TABLE</td>
<td>AQQueueTable.drop</td>
</tr>
<tr>
<td>Create a queue</td>
<td>DBMS_AQADM.CREATE_QUEUE</td>
<td>AQjmsSession.createQueue</td>
</tr>
<tr>
<td>Get a queue</td>
<td>Use schema.queue_name</td>
<td>AQjmsSession.getQueue</td>
</tr>
<tr>
<td>Create a multiconsumer queue/topic in a queue table with multiple consumers enabled</td>
<td>DBMS_AQADM.CREATE_QUEUE</td>
<td>AQjmsSession.createTopic</td>
</tr>
<tr>
<td>Get a multiconsumer queue/topic</td>
<td>Use schema.queue_name</td>
<td>AQjmsSession.getTopic</td>
</tr>
<tr>
<td>Alter a queue/topic</td>
<td>DBMS_AQADM.ALTER_QUEUE</td>
<td>AQjmsDestination.alter</td>
</tr>
<tr>
<td>Start a queue/topic</td>
<td>DBMS_AQADM.START_QUEUE</td>
<td>AQjmsDestination.start</td>
</tr>
<tr>
<td>Stop a queue/topic</td>
<td>DBMS_AQADM.STOP_QUEUE</td>
<td>AQjmsDestination.stop</td>
</tr>
<tr>
<td>Drop a queue/topic</td>
<td>DBMS_AQADM.DROP_QUEUE</td>
<td>AQjmsDestination.drop</td>
</tr>
<tr>
<td>Grant system privileges</td>
<td>DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE</td>
<td>AQjmsSession.grantSystem</td>
</tr>
<tr>
<td>Revoke system privileges</td>
<td>DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE</td>
<td>AQjmsSession.revokeSystem</td>
</tr>
<tr>
<td>Use Case</td>
<td>PL/SQL</td>
<td>Java (JMS)</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Grant a queue/topic privilege</td>
<td>DBMS_AQADM.GRANT_QUEUE_PRIVILEGE</td>
<td>AQjmsDestination.grantQueuePrivilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AQjmsDestination.grantTopicPrivilege</td>
</tr>
<tr>
<td>Revoke a queue/topic privilege</td>
<td>DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE</td>
<td>AQjmsDestination.revokeQueuePrivilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AQjmsDestination.revokeTopicPrivilege</td>
</tr>
<tr>
<td>Verify a queue type</td>
<td>DBMS_AQADM VERIFY_QUEUE_TYPES</td>
<td>Not supported</td>
</tr>
<tr>
<td>Add a subscriber</td>
<td>DBMS_AQADM ADD_SUBSCRIBER</td>
<td>See Table 3–6</td>
</tr>
<tr>
<td>Alter a subscriber</td>
<td>DBMS_AQADM ALTER_SUBSCRIBER</td>
<td>See Table 3–6</td>
</tr>
<tr>
<td>Remove a subscriber</td>
<td>DBMS_AQADM REMOVE_SUBSCRIBER</td>
<td>See Table 3–6</td>
</tr>
<tr>
<td>Schedule propagation</td>
<td>DBMS_AQADM SCHEDULE_PROPAGATION</td>
<td>AQjmsDestination.schedulePropagation</td>
</tr>
<tr>
<td>Enable a propagation schedule</td>
<td>DBMS_AQADM ENABLE_PROPAGATION_SCHEDULE</td>
<td>AQjmsDestination.enablePropagationSchedule</td>
</tr>
<tr>
<td>Alter a propagation schedule</td>
<td>DBMS_AQADM ALTER_PROPAGATION_SCHEDULE</td>
<td>AQjmsDestination.alterPropagationSchedule</td>
</tr>
<tr>
<td>Disable a propagation schedule</td>
<td>DBMS_AQADM DISABLE_PROPAGATION_SCHEDULE</td>
<td>AQjmsDestination.disablePropagationSchedule</td>
</tr>
<tr>
<td>Unschedule a propagation</td>
<td>DBMS_AQADM UNSCHEDULE_PROPAGATION</td>
<td>AQjmsDestination.unschedulePropagation</td>
</tr>
<tr>
<td>Create an Oracle® Database Internet Agent</td>
<td>DBMS_AQADM CREATE_AQ_AGENT</td>
<td>Not supported</td>
</tr>
<tr>
<td>Alter an Oracle® Database Internet Agent</td>
<td>DBMS_AQADM ALTER_AQ_AGENT</td>
<td>Not supported</td>
</tr>
<tr>
<td>Drop an Oracle® Database Internet Agent</td>
<td>DBMS_AQADM DROP_AQ_AGENT</td>
<td>Not supported</td>
</tr>
<tr>
<td>Grant database user privileges to an Oracle® Database Internet Agent</td>
<td>DBMS_AQADM ENABLE_AQ_AGENT</td>
<td>Not supported</td>
</tr>
<tr>
<td>Revoke database user privileges from an Oracle® Database Internet Agent</td>
<td>DBMS_AQADM DISABLE_AQ_AGENT</td>
<td>Not supported</td>
</tr>
<tr>
<td>Add alias for queue, agent, ConnectionFactory in a LDAP server</td>
<td>DBMS_AQADM ADD_ALIAS_TO_LDAP</td>
<td>Not supported</td>
</tr>
<tr>
<td>Delete alias for queue, agent, ConnectionFactory in a LDAP server</td>
<td>DBMS_AQADM DEL_ALIAS_FROM_LDAP</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Oracle® Database Operational Interfaces

Table 3–3 through Table 3–9 list equivalent Oracle® Database operational functions for the programmatic interfaces PL/SQL, OCI, Oracle® Database XML Servlet, and JMS, for various use cases.
## Table 3–3 Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Create Connection, Session, Message Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a connection</td>
<td>N/A</td>
<td>OCIServer Attach</td>
<td>Open an HTTP connection after authenticating with the Web server</td>
<td>AQjmsQueueConnectionFactory.createQueueConnection AQjmsTopicConnectionFactory.createTopicConnection</td>
</tr>
<tr>
<td>Create a session</td>
<td>N/A</td>
<td>OCISession Begin</td>
<td>An HTTP servlet session is automatically started with the first SOAP request</td>
<td>QueueConnection.createQueueSession TopicConnection.createTopicSession</td>
</tr>
<tr>
<td>Create a RAW message</td>
<td>Use SQL RAW type for message</td>
<td>Use OCIRaw for Message</td>
<td>Supply the hex representation of the message payload in the XML message. For example, <code>&lt;raw&gt;023f4523&lt;/raw&gt;</code></td>
<td>Not supported</td>
</tr>
<tr>
<td>Create a message with structured data</td>
<td>Use SQL Oracle object type for message</td>
<td>Use SQL Oracle object type for message</td>
<td>For Oracle object type queues that are not JMS queues (that is, they are not type AQJMS_*), the XML specified in <code>&lt;message payload&gt;</code> must map to the SQL type of the payload for the queue table. For JMS queues, the XML specified in the <code>&lt;message payload&gt;</code> must be one of the following: <code>&lt;jms_text_message&gt;,&lt;jms_map_message&gt;,&lt;jms_bytes_message&gt;,&lt;jms_object_message&gt;</code></td>
<td>Session.createTextMessage Session.createObjectMessage Session.createMapMessage Session.createBytesMessage Session.createStreamMessage AQjmsSession.createAdtMessage</td>
</tr>
<tr>
<td>Create a message producer</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>QueueSession.createSender TopicSession.createPublisher</td>
</tr>
</tbody>
</table>
## Table 3-4  Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enqueue a message to a single-consumer queue</td>
<td>DBMS_AQ.enqueue</td>
<td>OCIAQEnq</td>
<td>&lt;AQXmlSend&gt;</td>
<td>QueueSender.send</td>
</tr>
<tr>
<td>Specify visibility in ENQUEUE_OPTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enqueue a message to a queue and specify visibility options</td>
<td>DBMS_AQ.enqueue</td>
<td>OCIAQEnq</td>
<td>&lt;AQXmlSend&gt;</td>
<td>Not supported</td>
</tr>
<tr>
<td>Specify OCI_ATTR_VISIBILITY in OCIAQEnqOptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enqueue a message to a single-consumer queue and specify message properties priority and expiration</td>
<td>DBMS_AQ.enqueue</td>
<td>OCIAQEnq</td>
<td>&lt;AQXmlSend&gt;</td>
<td>Specify priority and TimeToLive during</td>
</tr>
<tr>
<td>Specify OCI_ATTR_PRIORITY, OCI_ATTR_EXPIRATION in OCIAQMsgProperties</td>
<td></td>
<td></td>
<td></td>
<td>QueueSender.send or</td>
</tr>
<tr>
<td>Enqueue a message to a single-consumer queue and specify message properties correlationID, delay, and exception queue</td>
<td>DBMS_AQ.enqueue</td>
<td>OCIAQEnq</td>
<td>&lt;AQXmlSend&gt;</td>
<td>Message.setJMS CorrelationI Delay and exception queue specified as provider specific message properties</td>
</tr>
<tr>
<td>Specify correlation, delay, exception_queue in MESSAGE_PROPERTIES</td>
<td></td>
<td></td>
<td></td>
<td>JMS_OracleDelay JMS_OracleExcpQ followed by QueueSender.send</td>
</tr>
<tr>
<td>Enqueue a message to a single-consumer queue and specify user-defined message properties</td>
<td>Not supported</td>
<td>Not supported</td>
<td>&lt;AQXmlSend&gt;</td>
<td>Message.setInt Property Message.setString Property Message.setBoolean Property and so forth, followed by QueueSender.send</td>
</tr>
<tr>
<td>Enqueue a message to a single-consumer queue and specify message transformation</td>
<td>DBMS_AQ.enqueue</td>
<td>OCIAQEnq</td>
<td>&lt;AQXmlSend&gt;</td>
<td>AQ.jmsQueueSender. setTransformation followed by QueueSender.send</td>
</tr>
<tr>
<td>Specify transformation in ENQUEUE_OPTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3–5 Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a message to a multiconsumer queue/topic using default subscription list</td>
<td>DBMS_AQ.enqueue Set recipient_list to NULL in MESSAGE_PROPERTIES</td>
<td>OCIAQEnq Set OCI_ATTR_RECIPIENT_LIST to NULL in OCIAQMsgProperties</td>
<td>&lt;AQXmlPublish&gt;</td>
<td>TopicPublisher.publish</td>
</tr>
<tr>
<td>Publish a message to a multiconsumer queue/topic using specific recipient list</td>
<td>DBMS_AQ.enqueue Specify recipient list in MESSAGE_PROPERTIES</td>
<td>OCIAQEnq Specify OCI_ATTR_RECIPIENT_LIST in OCIAQMsgProperties</td>
<td>&lt;AQXmlPublish&gt; Specify &lt;recipient_list&gt; in &lt;message_header&gt;</td>
<td>AQ.jmsTopic Publisher.publish Specify recipients as an array of AQ.jmsAgent</td>
</tr>
<tr>
<td>Publish a message to a multiconsumer queue/topic and specify message properties priority and expiration</td>
<td>DBMS_AQ.enqueue Specify priority, expiration in MESSAGE_PROPERTIES</td>
<td>OCIAQEnq Specify OCI_ATTR_PRIORITY, OCI_ATTR_EXPIRATION in OCIAQMsgProperties</td>
<td>&lt;AQXmlPublish&gt; Specify &lt;priority&gt;, &lt;expiration&gt; in &lt;message_header&gt;</td>
<td>TopicPublisher.publish Specify priority and TimeToLive during TopicPublisher.publish or MessageProducer.setTimeToLive and MessageProducer.setPriority followed by TopicPublisher.publish</td>
</tr>
</tbody>
</table>
Comparing Oracle® Database Programmatic Interfaces

Table 3–6  Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Subscribing for Messages in a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a subscriber</td>
<td>See administrative interfaces</td>
<td>Not supported</td>
<td>Not supported</td>
<td>TopicSession. createDurable Subscriber AQjmsSession. createDurable Subscriber</td>
</tr>
<tr>
<td>Alter a subscriber</td>
<td>See administrative interfaces</td>
<td>Not supported</td>
<td>Not supported</td>
<td>TopicSession. createDurable Subscriber AQjmsSession. createDurable Subscriber using the new selector</td>
</tr>
<tr>
<td>Remove a subscriber</td>
<td>See administrative interfaces</td>
<td>Not supported</td>
<td>Not supported</td>
<td>AQjmsSession. unsubscribe</td>
</tr>
</tbody>
</table>
## Table 3–7 Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Browse Messages in a Queue Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse messages in a queue/topic</td>
<td>DBMS_AQ.dequeue</td>
<td>OCIAQDeq</td>
<td>AQXmlReceive</td>
<td>QueueSession.createBrowser</td>
</tr>
<tr>
<td></td>
<td>Set dequeue_</td>
<td>Set OCI_ATTR_</td>
<td>Specify &lt;dequeue_mode&gt;</td>
<td>QueueBrowser.getEnumeration</td>
</tr>
<tr>
<td></td>
<td>mode to BROWSE in DEQUEUE_OPTIONS</td>
<td>DEQ_MODE to BROWSE in</td>
<td>&lt;consumer_options&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OCIAQDeqOptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;AQXmlReceive&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specify &lt;dequeue_mode&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LOCKED in &lt;consumer_options&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AQjmsSession.createBrowser</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set locked to TRUE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>QueueBrowser.getEnumeration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not supported on topics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oracle.jms.AQjmsSession. createBrowser</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oracle.jms.TopicBrowser. getEnumeration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not supported on topics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oracle.jms.AQjmsSession. createBrowser</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oracle.jms.TopicBrowser. getEnumeration</td>
<td></td>
</tr>
</tbody>
</table>

## Table 3–8 Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Receive Messages from a Queue/Topic Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start a connection for receiving messages</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Connection.start</td>
</tr>
<tr>
<td>Create a message consumer</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>QueueSession.createQueueReceiver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TopicSession.create</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DurableSubscriber</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AQjmsSession.createTopicReceiver</td>
</tr>
<tr>
<td>Dequeue a message from a queue/topic and specify visibility</td>
<td>DBMS_AQ.dequeue</td>
<td>OCIAQDeq</td>
<td>AQXmlReceive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify visibility in DEQUEUE_OPTIONS</td>
<td>Specify OCI_ATTR_VISIBILITY in</td>
<td>Specify &lt;visibility&gt; in &lt;consumer_options&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OCIAQDeqOptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dequeue a message from a queue/topic and specify transformation</td>
<td>DBMS_AQ.dequeue</td>
<td>OCIAQDeq</td>
<td>AQXmlReceive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify transformation in DEQUEUE_OPTIONS</td>
<td>Specify OCI_ATTR_TRANSFORMATION in</td>
<td>Specify &lt;transformation&gt; in &lt;consumer_options&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OCIAQDeqOptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dequeue a message from a queue/topic and specify navigation mode</td>
<td>DBMS_AQ.dequeue</td>
<td>OCIAQDeq</td>
<td>AQXmlReceive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specify navigation in DEQUEUE_OPTIONS</td>
<td>Specify OCI_ATTR_NAVIGATION in</td>
<td>Specify &lt;navigation&gt; in &lt;consumer_options&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OCIAQDeqOptions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dequeue a message from a single-consumer queue

- **PL/SQL**: `DBMS_AQ.dequeue`
- **OCI**: `OCIAQDeq`
- **AQ XML Servlet**: `<AQXmlReceive>`
- **JMS**: `QueueReceiver.receive` or `QueueReceiver.receiveNoWait` or `AQjmsQueueReceiver.receiveNoData`

Dequeue a message from a multiconsumer queue/topic using subscription name

- **PL/SQL**: `DBMS_AQ.dequeue`
- **OCI**: `OCIAQDeq`
- **AQ XML Servlet**: `<AQXmlReceive>`
- **JMS**: Create a durable `TopicSubscriber` on the topic using the subscription name, then `TopicSubscriber.receive` or `TopicSubscriber.receiveNoWait` or `AQjmsTopicSubscriber.receiveNoData`

Dequeue a message from a multiconsumer queue/topic using recipient name

- **PL/SQL**: `DBMS_AQ.dequeue`
- **OCI**: `OCIAQDeq`
- **AQ XML Servlet**: `<AQXmlReceive>`
- **JMS**: Create a `TopicReceiver` on the topic using the recipient name, then `AQjmsSession.createTopicReceiver` or `AQjmsTopicReceiver.receive` or `AQjmsTopicReceiver.receiveNoWait` or `AQjmsTopicReceiver.receiveNoData`
## Table 3–9  Comparison of Oracle® Database Programmatic Interfaces: Operational Interface—Register to Receive Messages Asynchronously from a Queue/Topic Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>PL/SQL</th>
<th>OCI</th>
<th>AQ XML Servlet</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive messages asynchronously from a single-consumer queue</td>
<td>Define a PL/SQL callback procedure Register it using <code>DBMS_AQ_REGISTER</code></td>
<td><code>OCISubscription Register</code> Specify queue_name as subscription name <code>OCISubscription Enable</code></td>
<td><code>&lt;AQXmlRegister&gt;</code> Specify queue name in <code>&lt;destination&gt;</code> and notification mechanism in <code>&lt;notify_url&gt;</code></td>
<td>Create a <code>QueueReceiver</code> on the queue, then <code>QueueReceiver.set</code> <code>MessageListener</code></td>
</tr>
<tr>
<td>Receive messages asynchronously from a multiconsumer queue/topic</td>
<td>Define a PL/SQL callback procedure Register it using <code>DBMS_AQ_REGISTER</code></td>
<td><code>OCISubscription Register</code> Specify queue:OCI_ATTR_CONSUMER_NAME as subscription name <code>OCISubscription Enable</code></td>
<td><code>&lt;AQXmlRegister&gt;</code> Specify queue name in <code>&lt;destination&gt;</code>, consumer in <code>&lt;consumer_name&gt;</code> and notification mechanism in <code>&lt;notify_url&gt;</code></td>
<td>Create a <code>TopicSubscriber</code> or <code>TopicReceiver</code> on the topic, then <code>TopicSubscriber.set</code> <code>MessageListener</code></td>
</tr>
<tr>
<td>Listen for messages on multiple queues/topics</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Listen for messages on one (many) single-consumer queues</td>
<td><code>DBMS_AQ_LISTEN</code> Use agent_name as NULL for all agents in agent_list</td>
<td><code>OCIAQListen</code> Use agent_name as NULL for all agents in agent_list</td>
<td>Not supported</td>
<td>Create multiple <code>QueueReceivers</code> on a <code>QueueSession</code>, then <code>QueueSession.set</code> <code>MessageListener</code></td>
</tr>
<tr>
<td>Listen for messages on one (many) multiconsumer queues/Topics</td>
<td><code>DBMS_AQ_LISTEN</code> Specify agent_name for all agents in agent_list</td>
<td><code>OCIAQListen</code> Specify agent_name for all agents in agent_list</td>
<td>Not supported</td>
<td>Create multiple <code>TopicSubscribers</code> or <code>TopicReceivers</code> on a <code>TopicSession</code>, then <code>TopicSession.set</code> <code>MessageListener</code></td>
</tr>
</tbody>
</table>
This chapter discusses topics related to managing Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- Oracle® Database Compatibility Parameters
- Queue Security and Access Control
- Queue Table Export-Import
- Oracle Enterprise Manager Support
- Using Oracle® Database with XA
- Restrictions on Queue Management
- Managing Propagation

**Oracle® Database Compatibility Parameters**

The queues in which buffered messages are stored must be created with compatibility set to 8.1 or higher.

The `compatible` parameter of `init.ora` and the `compatible` parameter of the queue table should be set to 8.1 or higher to use the following features:

- Queue-level access control
- Support for Real Application Clusters environments
- Rule-based subscribers for publish/subscribe
- Asynchronous notification
- Sender identification
- Separate storage of history management information
- Secure queues

**See Also:** Oracle Streams Concepts and Administration for more information on secure queues

Mixed case (upper and lower case together) queue names, queue table names, and subscriber names are supported if database compatibility is 10.0, but the names must be enclosed in double quote marks. So `abc.efg` means the schema is `ABC` and the name is `EFG`, but "abc"."efg" means the schema is `abc` and the name is `efg`. 
Queue Security and Access Control

This section contains these topics:

- Oracle® Database Security
- Queue Security
- Queue Privileges and Access Control
- OCI Applications and Queue Access
- Security Required for Propagation

Oracle® Database Security

Configuration information can be managed through procedures in the DBMS_AQADM package. Initially, only SYS and SYSTEM have execution privilege for the procedures in DBMS_AQADM and DBMS_AQ. Users who have been granted EXECUTE rights to these two packages are able to create, manage, and use queues in their own schemas. The MANAGE_ANY AQ system privilege is used to create and manage queues in other schemas.

See Also: "Granting Oracle® Database System Privileges" on page 8-18 for more information on AQ system privileges

Users of the Java Message Service (JMS) API need EXECUTE privileges on DBMS_AQJMS and DBMS_AQIN.

This section contains these topics:

- Administrator Role
- User Role
- Access to Oracle® Database Object Types

Administrator Role

The AQ_ADMINISTRATOR_ROLE has all the required privileges to administer queues. The privileges granted to the role let the grantee:

- Perform any queue administrative operation, including create queues and queue tables on any schema in the database
- Perform enqueue and dequeue operations on any queues in the database
- Access statistics views used for monitoring the queue workload
- Create transformations using DBMS_TRANSFORM
- Run all procedures in DBMS_AQELM
- Run all procedures in DBMS_AQJMS

User Role

You should avoid granting AQ_USER_ROLE, because this role does not provide sufficient privileges for enqueuing or dequeuing.

Your database administrator has the option of granting the system privileges ENQUEUE_ANY and DEQUEUE_ANY, exercising DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE and DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE directly to a database user, if you want the user to have this level of control.
You as the application developer give rights to a queue by granting and revoking privileges at the object level by exercising `DBMS_AQADM.GRANT_QUEUE_PRIVILEGE` and `DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE`.

As a database user, you do not need any explicit object-level or system-level privileges to enqueue or dequeue to queues in your own schema other than the `EXECUTE` right on `DBMS_AQ`.

**Access to Oracle® Database Object Types**

All internal Oracle® Database objects are now accessible to `PUBLIC`.

**Queue Security**

Oracle® Database administrators of Oracle Database can create queues. When you create queues, the default value of the `compatible` parameter in `DBMS_AQADM.CREATE_QUEUE_TABLE` is that of the `compatible` parameter.

To enqueue or dequeue, users need `EXECUTE` rights on `DBMS_AQ` and either enqueue or dequeue privileges on target queues, or `ENQUEUE_ANY/DEQUEUE_ANY` system privileges.

**Queue Privileges and Access Control**

You can grant or revoke privileges at the object level on queues. You can also grant or revoke various system-level privileges. Table 4–1 lists all common Oracle® Database operations and the privileges needed to perform these operations.

<table>
<thead>
<tr>
<th>Operation(s)</th>
<th>Privileges Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE/DROP/MONITOR own queues</td>
<td>Must be granted <code>EXECUTE</code> rights on <code>DBMS_AQADM</code>. No other privileges needed.</td>
</tr>
<tr>
<td>CREATE/DROP/MONITOR any queues</td>
<td>Must be granted <code>EXECUTE</code> rights on <code>DBMS_AQADM</code> and be granted <code>AQ_ADMINISTRATOR_ROLE</code> by another user who has been granted this role (SYS and SYSTEM are the first granters of <code>AQ_ADMINISTRATOR_ROLE</code>)</td>
</tr>
<tr>
<td>ENQUEUE/ DEQUEUE to own queues</td>
<td>Must be granted <code>EXECUTE</code> rights on <code>DBMS_AQ</code>. No other privileges needed.</td>
</tr>
<tr>
<td>ENQUEUE/ DEQUEUE to another's queues</td>
<td>Must be granted <code>EXECUTE</code> rights on <code>DBMS_AQ</code> and be granted privileges by the owner using <code>DBMS_AQADM.GRANT_QUEUE_PRIVILEGE</code>.</td>
</tr>
<tr>
<td>ENQUEUE/ DEQUEUE to any queues</td>
<td>Must be granted <code>EXECUTE</code> rights on <code>DBMS_AQ</code> and be granted <code>ENQUEUE_ANY QUEUE</code> or <code>DEQUEUE_ANY QUEUE</code> system privileges by an Oracle® Database administrator using <code>DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE</code>.</td>
</tr>
</tbody>
</table>

**OCI Applications and Queue Access**

For an Oracle Call Interface (OCI) application to access a queue, the session user must be granted either the object privilege of the queue he intends to access or the `ENQUEUE ANY QUEUE` or `DEQUEUE ANY QUEUE` system privileges. The `EXECUTE` right of `DBMS_AQ` is not checked against the session user’s rights.
Security Required for Propagation

Oracle® Database propagates messages through database links. The propagation driver dequeues from the source queue as owner of the source queue; hence, no explicit access rights need be granted on the source queue. At the destination, the login user in the database link should either be granted ENQUEUE ANY QUEUE privilege or be granted the right to enqueue to the destination queue. However, if the login user in the database link also owns the queue tables at the destination, then no explicit Oracle® Database privileges must be granted.

See Also: "Propagation from Object Queues" on page 4-9

Queue Table Export-Import

When a queue table is exported, the queue table data and anonymous blocks of PL/SQL code are written to the export dump file. When a queue table is imported, the import utility executes these PL/SQL anonymous blocks to write the metadata to the data dictionary.

Oracle AQ does not export registrations with a user export. All applications that make use of client registrations should take this into account as the client may not be present in the imported database.

Note: You cannot export or import buffered messages.

If there exists a queue table with the same name in the same schema in the database as in the export dump, then ensure that the database queue table is empty before importing a queue table with queues. Failing to do so has a possibility of ruining the metadata for the imported queue.

This section contains these topics:

- Exporting Queue Table Data
- Importing Queue Table Data
- Data Pump Export and Import

Exporting Queue Table Data

The export of queues entails the export of the underlying queue tables and related dictionary tables. Export of queues can also be accomplished at queue-table granularity.

Exporting Queue Tables with Multiple Recipients

A queue table that supports multiple recipients is associated with the following tables:

- Dequeue index-organized table (IOT)
- Time-management index-organized table
- Subscriber table
- A history IOT

These tables are exported automatically during full database mode, user mode and table mode exports. See "Export Modes" on page 4-5.
Because the metadata tables contain ROWIDs of some rows in the queue table, the import process generates a note about the ROWIDs being made obsolete when importing the metadata tables. This message can be ignored, because the queuing system automatically corrects the obsolete ROWIDs as part of the import operation. However, if another problem is encountered while doing the import (such as running out of rollback segment space), then you should correct the problem and repeat the import.

**Export Modes**
Exporting operates in full database mode, user mode, and table mode. Incremental exports on queue tables are not supported.

In full database mode, queue tables, all related tables, system-level grants, and primary and secondary object grants are exported automatically.

In user mode, queue tables, all related tables, and primary object grants are exported automatically. However, doing a user-level export from one schema to another using the `FROMUSER TOUSER` clause is not supported.

In table mode, queue tables, all related tables, and primary object grants are exported automatically. For example, when exporting a multiconsumer queue table, the following tables are automatically exported:

- `AQ$_queue_table_I` (the dequeue IOT)
- `AQ$_queue_table_T` (the time-management IOT)
- `AQ$_queue_table_S` (the subscriber table)
- `AQ$_queue_table_H` (the history IOT)

**Importing Queue Table Data**
Similar to exporting queues, importing queues entails importing the underlying queue tables and related dictionary data. After the queue table data is imported, the import utility executes the PL/SQL anonymous blocks in the dump file to write the metadata to the data dictionary.

**Importing Queue Tables with Multiple Recipients**
A queue table that supports multiple recipients is associated with the following tables:

- A dequeue IOT
- A time-management IOT
- A subscriber table
- A history IOT

These tables must be imported as well as the queue table itself.

**Import IGNORE Parameter**
You must not import queue data into a queue table that already contains data. The `IGNORE` parameter of the import utility must always be set to `NO` when importing queue tables. If the `IGNORE` parameter is set to `YES`, and the queue table that already exists is compatible with the table definition in the dump file, then the rows are loaded from the dump file into the existing table. At the same time, the old queue table definition is lost and re-created. Queue table definition prior to the import is lost and duplicate rows appear in the queue table.
Data Pump Export and Import

The Data Pump replace and skip modes are supported for queue tables. In the replace mode an existing queue table is dropped and replaced by the new queue table from the export dump file. In the skip mode, a queue table that already exists is not imported.

The truncate and append modes are not supported for queue tables. The behavior in this case is the same as the replace mode.

See Also: Oracle Database Utilities for more information on Data Pump Export and Data Pump Import

Oracle Enterprise Manager Support

Oracle Enterprise Manager supports most of the administrative functions of Oracle® Database. Oracle® Database functions are found under the Distributed node in the navigation tree of the Enterprise Manager console. Functions available through Oracle Enterprise Manager include:

- Using queues as part of the schema manager to view properties
- Creating, starting, stopping, and dropping queues
- Scheduling and unscheduling propagation
- Adding and removing subscribers
- Viewing propagation schedules for all queues in the database
- Viewing errors for all queues in the database
- Viewing the message queue
- Granting and revoking privileges
- Creating, modifying, or removing transformations

Using Oracle® Database with XA

You must specify "Objects=T" in the xa_open string if you want to use the Oracle® Database OCI interface. This forces XA to initialize the client-side cache in Objects mode. You are not required to do this if you plan to use Oracle® Database through PL/SQL wrappers from OCI or Pro*C.

The large object (LOB) memory management concepts from the Pro* documentation are not relevant for Oracle® Database raw messages because Oracle® Database provides a simple RAW buffer abstraction (although they are stored as LOBs).

When using the Oracle® Database navigation option, you must reset the dequeue position by using the FIRST_MESSAGE option if you want to continue dequeuing between services (such as xa_start and xa_end boundaries). This is because XA cancels the cursor fetch state after an xa_end. If you do not reset, then you get an error message stating that the navigation is used out of sequence (ORA-25237).

See Also:
- "Working with Transaction Monitors with Oracle XA" in Oracle Database Advanced Application Developer’s Guide for more information on XA
- "Large Objects (LOBs)" in Pro*C/C++ Programmer’s Guide
Restrictions on Queue Management

This section discusses restrictions on queue management.
This section contains these topics:

- Subscribers
- DML Not Supported on Queue Tables or Associated IOTs
- Propagation from Object Queues with REF Payload Attributes
- Collection Types in Message Payloads
- Synonyms on Queue Tables and Queues
- Synonyms on Object Types
- Tablespace Point-in-Time Recovery
- Virtual Private Database

Note: Mixed case (upper and lower case together) queue names, queue table names, and subscriber names are supported if database compatibility is 10.0, but the names must be enclosed in double quote marks. So abc.efg means the schema is ABC and the name is EFG, but "abc"."efg" means the schema is abc and the name is efg.

Subscribers

You cannot have more than 1,000 local subscribers for each queue. Also, only 32 remote subscribers are allowed for each remote destination database.

DML Not Supported on Queue Tables or Associated IOTs

Oracle® Database does not support data manipulation language (DML) operations on queue tables or associated index-organized tables (IOTs), if any. The only supported means of modifying queue tables is through the supplied APIs. Queue tables and IOTs can become inconsistent and therefore effectively ruined, if DML operations are performed on them.

Propagation from Object Queues with REF Payload Attributes

Oracle® Database does not support propagation from object queues that have REF attributes in the payload.

Collection Types in Message Payloads

You cannot construct a message payload using a VARRAY that is not itself contained within an object. You also cannot currently use a NESTED Table even as an embedded object within a message payload. However, you can create an object type that contains one or more VARRAYs, and create a queue table that is founded on this object type, as shown in Example 4–1.

Example 4–1 Creating Objects Containing VARRAYs

```
CREATE TYPE number_varray AS VARRAY(32) OF NUMBER;
CREATE TYPE embedded_varray AS OBJECT (col1 number_varray);
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'QT',
);```
queue_payload_type => 'embedded_varray');

Synonyms on Queue Tables and Queues
No Oracle® Database PL/SQL calls resolve synonyms on queues and queue tables. Although you can create synonyms, you should not apply them to the Oracle® Database interface.

Synonyms on Object Types
If you have created synonyms on object types, you cannot use them in `DBMS_AQADM.CREATE_QUEUE_TABLE`. Error ORA-24015 results.

Tablespace Point-in-Time Recovery
Oracle® Database currently does not support tablespace point-in-time recovery. Creating a queue table in a tablespace disables that particular tablespace for point-in-time recovery. Oracle® Database does support regular point-in-time recovery.

Virtual Private Database
You can use Oracle® Database with Virtual Private Database by specifying a security policy with Oracle® Database queue tables. While dequeuing, use the dequeue condition (`deq_cond`) or the correlation identifier for the policy to be applied. You can use "1=1" as the dequeue condition. If you do not use a dequeue condition or correlation ID, then the dequeue results in an error.

Note: When a dequeue condition or correlation identifier is used, the order of the messages dequeued is indeterminate, and the sort order of the queue is not honored.

Managing Propagation
This section contains these topics:

- EXECUTE Privileges Required for Propagation
- Propagation from Object Queues
- Optimizing Propagation
- Handling Failures in Propagation

Caution: For propagation to work correctly, the queue `aq$_prop_notify_X` should never be stopped or dropped and the table `aq$_prop_table_X` should never be dropped.

EXECUTE Privileges Required for Propagation
Propagation jobs are owned by SYS, but the propagation occurs in the security context of the queue table owner. Previously propagation jobs were owned by the user scheduling propagation, and propagation occurred in the security context of the user setting up the propagation schedule. The queue table owner must be granted EXECUTE privileges on the `DBMS_AQADM` package. Otherwise, the Oracle Database snapshot processes do not propagate and generate trace files with the error identifier `SYS.DBMS_AQADM` not defined. Private database links owned by the queue table
owner can be used for propagation. The username specified in the connection string must have EXECUTE access on the DBMS_AQ and DBMS_AQADM packages on the remote database.

**Propagation from Object Queues**

Propagation from object queues with BFILE objects is supported. To be able to propagate object queues with BFILE objects, the source queue owner must have read privileges on the directory object corresponding to the directory in which the BFILE is stored. The database link user must have write privileges on the directory object corresponding to the directory of the BFILE at the destination database.

AQ propagation does not support non-final types. Propagation of BFILE objects from object queues without specifying a database link is not supported.

See Also: "CREATE DIRECTORY" in Oracle Database SQL Language Reference for more information on directory objects

**Optimizing Propagation**

AQ propagation jobs are run by the Oracle Scheduler. Propagation may be scheduled in the following ways:

- A dedicated schedule in which the propagation runs forever or for a specified duration. This mode provides the lowest propagation latencies.
- A periodic schedule in which the propagation runs periodically for a specified interval. This may be used when propagation can be run in a batched mode.
- An event based system in which propagation is started when there are messages to be propagated. This mode makes more efficient use of available resources, while still providing a fast response time.

The administrator may choose a schedule that best meets the application performance requirements.

Oracle Scheduler will start the required number of job queue processes for the propagation schedules. Since the scheduler optimizes for throughput, if the system is heavily loaded, it may not run some propagation jobs. The resource manager may be used to have better control over the scheduling decisions. In particular, associating propagation jobs with different resource groups can allow for fairness in scheduling which may be important in heavy load situations.

In setting the number of JOB_QUEUE_PROCESSES, DBAs should be aware that this number is determined by the number of queues from which the messages must be propagated and the number of destinations (rather than queues) to which messages must be propagated.

A scheduling algorithm handles propagation. The algorithm optimizes available job queue processes and minimizes the time it takes for a message to show up at a destination after it has been enqueued into the source queue, thereby providing near-OLTP action. The algorithm can handle an unlimited number of schedules and various types of failures. While propagation tries to make the optimal use of the available job queue processes, the number of job queue processes to be started also depends on the existence of jobs unrelated to propagation, such as replication jobs. Hence, it is important to use the following guidelines to get the best results from the scheduling algorithm.

The scheduling algorithm uses the job queue processes as follows (for this discussion, an active schedule is one that has a valid current window):
If the number of active schedules is fewer than half the number of job queue processes, then the number of job queue processes acquired corresponds to the number of active schedules.

If the number of active schedules is more than half the number of job queue processes, after acquiring half the number of job queue processes, then multiple active schedules are assigned to an acquired job queue process.

If the system is overloaded (all schedules are busy propagating), depending on availability, then additional job queue processes are acquired up to one fewer than the total number of job queue processes.

If none of the active schedules handled by a process has messages to be propagated, then that job queue process is released.

The algorithm performs automatic load balancing by transferring schedules from a heavily loaded process to a lightly load process such that no process is excessively loaded.

Handling Failures in Propagation

The scheduling algorithm has robust support for handling failures. Common failures that prevent message propagation include the following:

- Database link failed
- Remote database is not available
- Remote queue does not exist
- Remote queue was not started
- Security violation while trying to enqueue messages into remote queue

Under all these circumstances the appropriate error messages are reported in the `DBA_QUEUE_SCHEDULES` view.

When an error occurs in a schedule, propagation of messages in that schedule is attempted again after a retry period of 30*(number of failures) seconds, with an upper bound of ten minutes. After sixteen consecutive retries, the schedule is disabled.

If the problem causing the error is fixed and the schedule is enabled, then the error fields that indicate the last error date, time, and message continue to show the error information. These fields are reset only when messages are successfully propagated in that schedule.

See Also: Chapter 7, "Troubleshooting Oracle® Database"
This chapter discusses performance and scalability issues relating to Oracle Streams Advanced Queuing (AQ).

This chapter contains the following topics:

- Persistent Messaging Performance Overview
- Persistent Messaging Basic Tuning Tips
- Propagation Tuning Tips
- Buffered Messaging Tuning

**Persistent Messaging Performance Overview**

When persistent messages are enqueued, they are stored in database tables. The performance characteristics of `queue` operations on persistent messages are similar to underlying database operations. The code path of an `enqueue` operation is comparable to `SELECT` and `INSERT` into a multicolumn `queue table` with three index-organized tables. The code path of a `dequeue` operation is comparable to a `SELECT` operation on the multi-column table and a `DELETE` operation on the dequeue index-organized table.

In many scenarios, for example when Oracle Real Application Clusters (RAC) is not used and there is adequate streams pool memory, the dequeue operation is optimized and is comparable to a `SELECT` operation on a multi-column table.

**Note:** Performance is not affected by the number of queues in a table.

**Oracle® Database and Oracle Real Application Clusters**

Real Application Clusters (RAC) can be used to ensure highly available access to queue data. The entry and exit points of a queue, commonly called its tail and head respectively, can be extreme hot spots. Because RAC may not scale well in the presence of hot spots, limit usual access to a queue from one instance only. If an instance failure occurs, then messages managed by the failed instance can be processed immediately by one of the surviving instances.

You can associate RAC instance affinities with 8.1-compatible queue tables. If you are using q1 and q2 in different instances, then you can use `ALTER_QUEUE_TABLE` or `CREATE_QUEUE_TABLE` on the queue table and set `primary_instance` to the appropriate `instance_id`.
See Also:
- "Creating a Queue Table" on page 8-1
- "Altering a Queue Table" on page 8-8

Oracle® Database in a Shared Server Environment

Queue operation scalability is similar to the underlying database operation scalability. If a dequeue operation with wait option is applied, then it does not return until it is successful or the wait period has expired. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes. For this reason, Oracle recommends that dequeue requests with wait option be applied using dedicated server processes. This restriction is not enforced.

See Also:  "DEQUEUE_OPTIONS_T Type" in Oracle Database PL/SQL Packages and Types Reference for more information on the wait option

Persistent Messaging Basic Tuning Tips

Oracle® Database table layout is similar to a layout with ordinary database tables and indexes.

See Also:  Oracle Database Performance Tuning Guide for tuning recommendations

Memory Requirements

Streams pool size should be at least 20 MB for optimal multi-consumer dequeue performance in a non-RAC database. Persistent queuing dequeue operations use the streams pool to optimize performance, especially under concurrency situations. This is, however, not a requirement and the code automatically switches to a less optimal code path.

Using Storage Parameters

Storage parameters can be specified when creating a queue table using the storage_clause parameter. Storage parameters are inherited by other IOTs and tables created with the queue table. The tablespace of the queue table should have sufficient space to accommodate data from all the objects associated with the queue table. With retention specified, the history table as well as the queue table can grow to be quite big.

Oracle recommends you use automatic segment-space management (ASSM). Otherwise initrans, freelists and freelist groups must be tuned for AQ performance under high concurrency.

Increasing pctfree will reduce the number of messages in a queue table/IOT block. This will reduce block level contention when there is concurrency.

Storage parameters specified at queue table creation are shared by the queue table, IOTs and indexes. These may be individually altered by an online redefinition using DBMS_REDEFINTION.
I/O Configuration

Because Oracle® Database is very I/O intensive, you will usually need to tune I/O to remove any bottlenecks.

See Also: "I/O Configuration and Design" in Oracle Database Performance Tuning Guide

Running Enqueue and Dequeue Processes Concurrently in a Single Queue Table

Some environments must process messages in a constant flow, requiring that enqueue and dequeue processes run concurrently. If the message delivery system has only one queue table and one queue, then all processes must work on the same segment area at the same time. This precludes reasonable performance levels when delivering a high number of messages.

The best number for concurrent processes depends on available system resources. For example, on a four-CPU system, it is reasonable to start with two concurrent enqueue and two concurrent dequeue processes. If the system cannot deliver the wanted number of messages, then use several subscribers for load balancing rather than increasing the number of processes.

Tune the enqueue and dequeue rates on the queue so that in the common case the queue size remains small and bounded. A queue that grows and shrinks considerably will have indexes and IOTs that are out of balance, which will affect performance.

With multi-consumer queues, using several subscribers for load balancing rather than increasing the number of processes will reduce contention. Multiple queue tables may be used garnering horizontal scalability.

Running Enqueue and Dequeue Processes Serially in a Single Queue Table

When enqueue and dequeue processes are running serially, contention on the same data segment is lower than in the case of concurrent processes. The total time taken to deliver messages by the system, however, is longer than when they run concurrently. Increasing the number of processes helps both enqueuing and dequeuing. The message throughput rate may be higher for enqueuers than for dequeuers when the number of processes is increased, especially with single consumer queues. Dequeue processes on multi-consumer queues scale much better.

Creating Indexes on a Queue Table

Creating an index on a queue table is useful if you:

- Dequeue using correlation ID
  
  An index created on the column `corr_id` of the underlying queue table `AQ$QueueTableName` expedites dequeues.

- Dequeue using a condition

  This is like adding the condition to the where-clause for the `SELECT` on the underlying queue table. An index on `QueueTableName` expedites performance on this `SELECT` statement.

Other Tips

- Ensure that statistics are being gathered so that the optimal query plans for retrieving messages are being chosen. By default, queue tables are locked out from
automatic gathering of statistics. The recommended use is to gather statistics with a representative queue message load and lock them.

- The queue table indexes and IOTs are automatically coalesced by AQ background processes. However, they must continue to be monitored and coalesced if needed. In 10.2, with automatic space segment management (ASSM), an online shrink operation may be used for the same purpose. A well balanced index reduces queue monitor CPU consumption, and ensures optimal enqueue-dequeue performance.

- Ensure that there are enough queue monitor processes running to perform the background tasks. The queue monitor must also be running for other crucial background activity. Multiple qmn processes share the load; make sure that there are enough of them. These are auto-tuned, but can be forced to a minimum number, if needed.

- It is recommended that dequeue with a wait time is only used with dedicated server processes. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes.

- Long running dequeue transactions worsen dequeue contention on the queue, and must be avoided.

- Batching multiple dequeue operations on multi-consumer queues into a single transaction gives best throughput.

- Use NEXT as navigation mode, if not using message priorities. This offers the same semantics but improved performance.

- Use the REMOVE_NODATA dequeue mode if dequeuing in BROWSE mode followed by a REMOVE.

### Propagation Tuning Tips

Propagation can be considered a special kind of dequeue operation with an additional INSERT at the remote (or local) queue table. Propagation from a single schedule is not parallelized across multiple job queue processes. Rather, they are load balanced. For better scalability, configure the number of propagation schedules according to the available system resources (CPUs).

Propagation rates from transactional and nontransactional (default) queue tables vary to some extent because Oracle® Database determines the batching size for nontransactional queues, whereas for transactional queues, batch size is mainly determined by the user application.

Optimized propagation happens in batches. If the remote queue is in a different database, then Oracle® Database uses a sequencing algorithm to avoid the need for a two-phase commit. When a message must be sent to multiple queues in the same destination, it is sent multiple times. If the message must be sent to multiple consumers in the same queue at the destination, then it is sent only once.

### Buffered Messaging Tuning

Buffered messaging operations in a Real Application Clusters environment will be fastest on the OWNER_INSTANCE of the queue.
Performance Views

Oracle provides views to monitor system performance and troubleshooting:

- (G)V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance
- (G)V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance
- (G)V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance
- (G)V$BUFFERED_QUEUES: All Buffered Queues in the Instance.
- (G)V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance
- (G)V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance
- (G)V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues

These views are integrated with the Automatic Workload Repository (AWR). Users can generate a report based on two AWR snapshots to compute enqueue rate, dequeue rate, and other statistics per queue/subscriber.
You can access Oracle Streams Advanced Queuing (AQ) over the Internet by using **Simple Object Access Protocol** (SOAP). **Internet Data Access Presentation** (IDAP) is the SOAP specification for Oracle® Database operations. IDAP defines XML **message** structure for the body of the SOAP request. An IDAP-structured message is transmitted over the Internet using HTTP.

Users can register for notifications using the IDAP interface.

This chapter contains these topics:

- Overview of Oracle® Database Operations over the Internet
- Deploying the Oracle® Database XML Servlet
- Internet Data Access Presentation (IDAP)
- Request and Response IDAP Documents
- Notification of Messages by e-mail

**See Also:**

- Appendix B, "SOAP and Oracle Streams AQ XML Schemas", which appears only in the HTML version of this guide
- Table 1–3, "Oracle® Database XML Demonstrations" on page 1-39 for the locations of AQ XML demonstrations

### Overview of Oracle® Database Operations over the Internet

This section contains these topics:

- Oracle® Database Internet Operations Architecture
- Internet Message Payloads
- Configuring the Web Server to Authenticate Users Sending POST Requests
- Client Requests Using HTTP
- Oracle® Database Servlet Responses Using HTTP
- Oracle® Database Propagation Using HTTP and HTTPS

### Oracle® Database Internet Operations Architecture

Figure 6–1 shows the architecture for performing Oracle® Database operations over HTTP. The major components are:

- Oracle® Database client program
■ Web server/servlet runner hosting the Oracle® Database servlet
■ Oracle Database server

A Web browser or any other HTTP client can serve as an Oracle® Database client program, sending XML messages conforming to IDAP to the Oracle® Database servlet, which interprets the incoming XML messages. The Oracle® Database servlet connects to the Oracle Database server and performs operations on user queues.

Figure 6–1 Architecture for Performing Oracle® Database Operations Using HTTP

Internet Message Payloads

Oracle® Database supports messages of three types: RAW, Oracle object, and Java Message Service (JMS). All these message types can be accessed using SOAP and Web services. If the queue holds messages in RAW, Oracle object, or JMS format, then XML payloads are transformed to the appropriate internal format during enqueue and stored in the queue. During dequeue, when messages are obtained from queues containing messages in any of the preceding formats, they are converted to XML before being sent to the client.

The message payload type depends on the queue type on which the operation is being performed:

RAW Queues
The contents of RAW queues are raw bytes. You must supply the hex representation of the message payload in the XML message. For example, <raw>023f4523</raw>.

Oracle Object Type Queues
For Oracle object type queues that are not JMS queues (that is, they are not type AQ$_JMS_*”), the type of the payload depends on the type specified while creating the queue table that holds the queue. The content of the XML elements must map to the attributes of the object type of the queue table.

JMS Type Queues/Topics
For queues with JMS types (that is, those with payloads of type AQ$_JMS_*”), there are four XML elements, depending on the JMS type. IDAP supports queues or topics with the following JMS types:
■ TextMessage
■ MapMessage
■ BytesMessage
■ ObjectMessage

JMS queues with payload type StreamMessage are not supported through IDAP.
Configuring the Web Server to Authenticate Users Sending POST Requests

After the servlet is installed, the Web server must be configured to authenticate all users that send POST requests to the Oracle® Database servlet. The Oracle® Database servlet allows only authenticated users to access the servlet. If the user is not authenticated, then an error is returned by the servlet.

The Web server can be configured in multiple ways to restrict access. Some of the common techniques are basic authentication (username/password) over SSL and client certificates. Consult your Web server documentation to see how you can restrict access to servlets.

In the context of the Oracle® Database servlet, the username that is used to connect to the Web server is known as the Oracle® Database HTTP agent or Oracle® Database Internet user.

Client Requests Using HTTP

An Oracle® Database client begins a request to the Oracle® Database servlet using HTTP by opening a connection to the server. The client logs in to the server using HTTP basic authentication (with or without SSL) or SSL certificate-based client authentication. The client constructs an XML message representing the send, publish, receive or register request.

The client sends an HTTP POST to the servlet at the remote server.

User Sessions and Transactions

After a client is authenticated and connects to the Oracle® Database servlet, an HTTP session is created on behalf of the user. The first request in the session also implicitly starts a new database transaction. This transaction remains open until it is explicitly committed or terminated. The responses from the servlet includes the session ID in the HTTP headers as cookies.

If the client wishes to continue work in the same transaction, then it must include this HTTP header containing the session ID cookie in subsequent requests. This is automatically accomplished by most Web browsers. However, if the client is using a Java or C client to post requests, then this must be accomplished programmatically.

Oracle® Database Servlet Responses Using HTTP

The server accepts the client HTTP(S) connection and authenticates the user (Oracle® Database agent) specified by the client. The server receives the POST request and invokes the Oracle® Database servlet.
If this is the first request from this client, then a new HTTP session is created. The XML message is parsed and its contents are validated. If a session ID is passed by the client in the HTTP headers, then this operation is performed in the context of that session.

**See Also:** "User Sessions and Transactions" on page 6-3

The servlet determines which object (queue/topic) the agent is trying to perform operations on. The servlet looks through the list of database users that map to this Oracle® Database agent. If any one of these users has privileges to access the queue/topic specified in the request, then the Oracle® Database servlet superuser creates a session on behalf of this user.

If no transaction is active in the HTTP session, then a new database transaction is started. Subsequent requests in the session are part of the same transaction until an explicit COMMIT or ROLLBACK request is made. The effects of the transaction are visible only after it is committed. If the transaction remains inactive for 120 seconds, then it is automatically terminated.

The requested operation is performed. The response is formatted as an XML message and sent back the client. The response also includes the session ID in the HTTP headers as a cookie.

**Oracle® Database Propagation Using HTTP and HTTPS**

You can propagate over HTTP and HTTPS (HTTP over SSL) instead of Oracle Net Services. HTTP, unlike Oracle Net Services, is easy to configure for firewalls. The background process doing propagation pushes messages to an Oracle® Database servlet that enqueues them into the destination database, as shown in **Figure 6–2**.

**Figure 6–2 HTTP Oracle® Database Propagation**

You can set up any application to use Oracle® Database HTTP propagation without any change to the existing code. An application using Oracle® Database HTTP propagation can easily switch back to Net Services propagation just by re-creating the database link with a Net Services connection string, without any other changes.

**Deploying the Oracle® Database XML Servlet**

Follow these steps to deploy the AQ XML servlet using OC4J:

1. For JDK1.2.x or JDK1.3.x, include the following in your CLASSPATH:

   ```
   ORACLE_HOME/jdbc/lib/classes12.zip
   ORACLE_HOME/jdbc/lib/nls_charset12.zip
   ORACLE_HOME/jlib/javax-ssl-1_1.jar
   ORACLE_HOME/jlib/jndi.jar
   ORACLE_HOME/jlib/jssl-1_1.jar
   ORACLE_HOME/jlib/jta.jar
   ORACLE_HOME/jlib/orai18n.jar
   ```
Deploying the Oracle® Database XML Servlet

ORACLE_HOME/jlib/orai18n-collation.jar
ORACLE_HOME/jlib/orai18n-mapping.jar
ORACLE_HOME/jlib/orai18n-utility.jar
ORACLE_HOME/jlib/http_client.jar
ORACLE_HOME/lib/lclasses12.zip
ORACLE_HOME/lib/servlet.jar
ORACLE_HOME/lib/xmlparserv2.jar
ORACLE_HOME/lib/xschema.jar
ORACLE_HOME/lib/xsu12.jar
ORACLE_HOME/rdbms/jlib/aqapi.jar
ORACLE_HOME/rdbms/jlib/aqxml.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
ORACLE_HOME/rdbms/jlib/xdb.jar

2. For JDK1.4.x, include the following in your CLASSPATH:

ORACLE_HOME/jdbc/lib/ojdbc14.jar
ORACLE_HOME/jlib/javax-ssl-1_1.jar
ORACLE_HOME/jlib/jndi.jar
ORACLE_HOME/jlib/jssl-1_1.jar
ORACLE_HOME/jlib/jta.jar
ORACLE_HOME/jlib/orai18n.jar
ORACLE_HOME/jlib/orai18n-collation.jar
ORACLE_HOME/jlib/orai18n-mapping.jar
ORACLE_HOME/jlib/orai18n-utility.jar
ORACLE_HOME/jlib/http_client.jar
ORACLE_HOME/lib/lclasses12.zip
ORACLE_HOME/lib/servlet.jar
ORACLE_HOME/lib/xmlparserv2.jar
ORACLE_HOME/lib/xschema.jar
ORACLE_HOME/lib/xsu12.jar
ORACLE_HOME/rdbms/jlib/aqapi.jar
ORACLE_HOME/rdbms/jlib/aqxml.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
ORACLE_HOME/rdbms/jlib/xdb.jar

Note: http_client.jar, jssl-1_1.jar, and javax-ssl-1_1.jar are required by HTTPClient used in AQHttp.java and AQHttpRq.java.

3. Compile AQHttpRq.java:

```
cd ORACLE_HOME/rdbms/demo
javac AQHttpRq.java AQHttpGet.java
```

4. Set the following database initialization parameters to the indicated values:

```
job_queue_processes=2
compatible=10.2.0
```

5. Restart the database and listener.

6. Set up queues and authenticate users for restricted access.

See Also: aqxxmlREADME.txt and aqxxmlmdo.sql in ORACLE_HOME/rdbms/demo for additional information.

7. Deploy the servlet and start the OC4J instance:
cd ORACLE_HOME/bin
sh aqxmlctl deploy
sh aqxmlctl start

---

**Note:** Use `sh aqxmlctl stop` to stop the OC4J instance. The deploy servlet and start OC4J instance steps might have been done during your Oracle Database installation. You can verify this in the following steps.

---

8. Check the status of the servlet and information on the protocol and port number used for deploying the servlet in the following files:

   ORACLE_HOME/rdbms/demo/aqxml.ini

   ORACLE_HOME/oc4j/j2ee/OC4J_AQ/config/rmi.xml

   ORACLE_HOME/oc4j/j2ee/OC4J_AQ/config/http-web-site.xml

9. Point a web browser to the following URL:

   https://hostname:portnumber/aqserv/servlet/AQDemoServlet

   where **hostname** is the server name, and **portnumber** is the value discovered in the previous step. After you respond to a username/password prompt, the servlet displays:

   Sample AQ Servlet

   AQxmlServlet is working!

10. Create an SSL Certificate and generate a keystore. The following files provide examples:

    ORACLE_HOME/rdbms/demo/aqxml4j.cert

    ORACLE_HOME/rdbms/demo/keystore

    The following tags in **ORACLE_HOME/oc4j/j2ee/OC4J_AQ/config/http-web-site.xml** indicate that the Web site is secure and keystore is used for SSL authentication:

    <web-site port="443" secure="true">
        ...
        <ssl-config
            keystore="ORACLE_HOME/oc4j/j2ee/home/keystore"
            keystore-password="welcome" />
    </web-site>

    To make the site access only HTTP requests, remove `secure="true"` and `<ssl-config>` from **http-web-site.xml**.

11. Stop and restart the AQ XML servlet:

    sh aqxmlctl stop

    sh aqxmlctl start

---

See Also: Keytool documentation at

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html

The following tags in **ORACLE_HOME/oc4j/j2ee/OC4J_AQ/config/http-web-site.xml** indicate that the Web site is secure and keystore is used for SSL authentication:

```xml
<web-site port="443" secure="true">
    ...
    <ssl-config
        keystore="ORACLE_HOME/oc4j/j2ee/home/keystore"
        keystore-password="welcome" />
</web-site>
```

To make the site access only HTTP requests, remove `secure="true"` and `<ssl-config>` from **http-web-site.xml**.

---

6-6  Oracle Streams Advanced Queuing User's Guide
Internet Data Access Presentation (IDAP)

Internet Data Access Presentation (IDAP) uses the Content-Type of text/xml to specify the body of the SOAP request. XML provides the presentation for IDAP request and response messages as follows:

- All request and response tags are scoped in the SOAP namespace.
- Oracle® Database operations are scoped in the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes in the SOAP body.
- The receiver processes SOAP messages that have correct namespaces and returns an invalid request error for requests with incorrect namespaces.
- The SOAP namespace has the value http://schemas.xmlsoap.org/soap/envelope/
- The IDAP namespace has the value http://ns.oracle.com/AQ/schemas/access

SOAP Message Structure

SOAP structures a message request or response as follows:

- SOAP Envelope
- SOAP Header
- SOAP Body

SOAP Envelope

This is the root or top element in an XML tree. Its tag is SOAP:Envelope. SOAP defines a global attribute SOAP:encodingStyle that indicates serialization rules used instead of those described by the SOAP specification. This attribute can appear on any element and is scoped to that element and all child elements not themselves containing such an attribute. Omitting this attribute means that type specification has been followed unless overridden by a parent element.

The SOAP envelope also contains namespace declarations and additional attributes, provided they are namespace-qualified. Additional namespace-qualified subelements can follow the body.

SOAP Header

This is the first element under the root. Its tag is SOAP:Header. A SOAP header passes necessary information, such as the transaction identifier. The header is encoded as a child of the SOAP:Envelope XML element. Headers are identified by the name element and are namespace-qualified. A header entry is encoded as an embedded element.

SOAP Body

This is the Oracle® Database XML document. Its tag is SOAP:Body, and it contains a first subelement whose name is the method name. This method request element contains elements for each input and output parameter. The element names are the parameter names. The body also contains SOAP:Fault, indicating information about an error. The Oracle® Database XML document has the namespace http://ns.oracle.com/AQ/schemas/access
SOAP Method Invocation

A method invocation is performed by creating the request header and body and processing the returned response header and body. The request and response headers can consist of standard transport protocol-specific and extended headers.

HTTP Headers

The `POST` method within the HTTP request header performs the SOAP method invocation. The request should include the header `SOAPMethodName`, whose value indicates the method to be invoked on the target. The value is of the form `URI#method name`. For example:

```
SOAPMethodName: http://ns.oracle.com/AQ/schemas/access#AQXmlSend
```

The URI used for the interface must match the implied or specified namespace qualification of the method name element in the `SOAP:Body` part of the payload. The method name must not include the "#" character.

Method Invocation Body

SOAP method invocation consists of a method request and optionally a method response. The SOAP method request and method response are an HTTP request and response, respectively, whose contents are XML documents consisting of the root and mandatory body elements. These XML documents are referred to as SOAP payloads in the rest of this chapter.

A SOAP payload is defined as follows:

- The SOAP root element is the top element in the XML tree.
- The SOAP payload headers contain additional information that must travel with the request.
- The method request is represented as an XML element with additional elements for parameters. It is the first child of the `SOAP:Body` element. This request can be one of the Oracle® Database XML client requests described in the next section.
- The response is the return value or an error or exception that is passed back to the client.

At the receiving site, a request can have one of the following outcomes:

- The HTTP infrastructure on the receiving site is able to receive and process the request. In this case, the HTTP infrastructure passes the headers and body to the SOAP infrastructure.
- The HTTP infrastructure on the receiving site cannot receive and process the request. In this case, the result is an HTTP response containing an HTTP error in the status field and no XML body.
- The SOAP infrastructure on the receiving site is able to decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the method indicated in the method request. In this case, the result of the method request consists of a response or error.
- The SOAP infrastructure on the receiving site cannot decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the interface or method indicated in the method request. In this case, the result of the method is an error.
that prevented the dispatching infrastructure on the receiving side from successful completion.

In the last two cases, additional message headers can be present in the results of the request for extensibility.

Results from a Method Request

The results of the request are to be provided in the form of a request response. The HTTP response must be of Content-Type text/xml. A SOAP result indicates success and an error indicates failure. The method response never contains both a result and an error.

Request and Response IDAP Documents

The body of a SOAP message is an IDAP message. This XML document has the namespace http://ns.oracle.com/AQ/schemas/access. The body represents:

- Client requests for enqueue, dequeue, and registration
- Server responses to client requests for enqueue, dequeue, and registration
- Notifications from the server to the client

Note: Oracle® Database Internet access is supported only for 8.1 or higher style queues.

This section contains these topics:

- IDAP Client Requests for Enqueue
- IDAP Client Requests for Dequeue
- IDAP Client Requests for Registration
- IDAP Client Requests to Commit a Transaction
- IDAP Client Requests to Roll Back a Transaction
- IDAP Server Response to an Enqueue Request
- IDAP Server Response to a Dequeue Request
- IDAP Server Response to a Register Request
- IDAP Commit Response
- IDAP Rollback Response
- IDAP Notification
- IDAP Response in Case of Error

IDAP Client Requests for Enqueue

Client send and publish requests use AQXmlSend to enqueue to a single-consumer queue and AQXmlPublish to enqueue to multiconsumer queues/topics. AQXmlSend and AQXmlPublish contain the following elements:

- producer_options
- message_set
message_header
message_payload
AQQxmlCommit

**producer_options**
This is a required element. It contains the following child elements:

- **destination**
  This element is required. It specifies the queue/topic to which messages are to be sent. It has an optional lookup_type attribute, which determines how the destination value is interpreted. If lookup_type is DATABASE, which is the default, then the destination is interpreted as schema.queue_name. If lookup_type is LDAP, then the LDAP server is used to resolve the destination.

- **visibility**
  This element is optional. It determines when an enqueue becomes visible. The default is ON_COMMIT, which makes the enqueue visible when the current transaction commits. If IMMEDIATE is specified, then the effects of the enqueue are visible immediately after the request is completed. The enqueue is not part of the current transaction. The operation constitutes a transaction on its own.

- **transformation**
  This element is optional. It specifies the PL/SQL transformation to be invoked before the message is enqueued.

**message_set**
This is a required element and contains one or more messages. Each message consists of a message_header and a message_payload.

**message_header**
This element is optional. It contains the following child elements:

- **sender_id**
  If a message_header element is included, then it must contain a sender_id element, which specifies an application-specific identifier. The sender_id element can contain agent_name, address, protocol, and agent_alias elements. The agent_alias element resolves to a name, address, and protocol using LDAP.

- **message_id**
  This element is optional. It is a unique identifier of the message, supplied during dequeue.

- **correlation**
  This element is optional. It is the correlation identifier of the message.

- **delay**
  This element is optional. It specifies the duration in seconds after which a message is available for processing.

- **expiration**
  This element is optional. It specifies the duration in seconds that a message is available for dequeuing. This parameter is an offset from the delay. By default
messages never expire. If a message is not dequeued before it expires, then it is moved to an **exception queue** in the **EXPIRED** state.

- **priority**
  This element is optional. It specifies the priority of the message. The priority can be any number, including negative numbers. A smaller number indicates higher priority.

- **recipient_list**
  This element is optional. It is a list of recipients which overrides the default subscriber list. Each recipient is represented in **recipient_list** by a **recipient element**, which can contain **agent_name**, **address**, **protocol**, and **agent_alias** elements. The **agent_alias** element resolves to a name, address, and protocol using LDAP.

- **message_state**
  This element is optional. It specifies the state of the message. It is filled in automatically during dequeue. If **message_state** is 0, then the message is ready to be processed. If it is 1, then the message delay has not yet been reached. If it is 2, then the message has been processed and is retained. If it is 3, then the message has been moved to an exception queue.

- **exception_queue**
  This element is optional. It specifies the name of the queue to which the message is moved if the number of unsuccessful dequeue attempts has exceeded **max_retries** or the message has expired. All messages in the exception queue are in the **EXPIRED** state.

  If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is used, then the parameter returns a **NULL** value at dequeue time.

**message_payload**
This is a required element. It can contain different elements based on the payload type of the destination queue/topic. The different payload types are described in "IDAP Client Requests for Dequeue" on page 6-11.

**AQXmlCommit**
This is an optional empty element. If it is included, then the transaction is committed at the end of the request.

*See Also:* "Internet Message Payloads" on page 6-2 for an explanation of IDAP message payloads

## IDAP Client Requests for Dequeue
Client requests for dequeue use **AQXmlReceive**, which contains the following elements:

- **consumer_options**
- **AQXmlCommit**

**consumer_options**
This is a required element. It contains the following child elements:
- **destination**
  This element is required. It specifies the queue/topic from which messages are to be received. The `destination` element has an optional `lookup_type` attribute, which determines how the destination value is interpreted. If `lookup_type` is `DATABASE`, which is the default, then the destination is interpreted as `schema.queue_name`. If `lookup_type` is `LDAP`, then the LDAP server is used to resolve the destination.

- **consumer_name**
  This element is optional. It specifies the name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should not be specified.

- **wait_time**
  This element is optional. It specifies the number of seconds to wait if there is no message currently available which matches the search criteria.

- **selector**
  This element is optional. It specifies criteria used to select the message. It can contain child elements `correlation`, `message_id`, or `condition`.

  A dequeue condition element is a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties, user object payload data properties, and PL/SQL or SQL functions. Message properties include `priority`, `corrid` and other columns in the queue table.

  To specify dequeue conditions on a message payload, use attributes of the `object type` in clauses. You must prefix each attribute with `tabuser_data` as a qualifier to indicate the specific column of the queue table that stores the payload.

  A dequeue condition element cannot exceed 4000 characters.

  **Note:** When a dequeue condition or correlation identifier is used, the order of the messages dequeued is indeterminate, and the sort order of the queue is not honored.

- **visibility**
  This element is optional. It determines when a dequeue becomes visible. The default is `ON_COMMIT`, which makes the dequeue visible when the current transaction commits. If `IMMEDIATE` is specified, then the effects of the dequeue are visible immediately after the request is completed. The dequeue is not part of the current transaction. The operation constitutes a transaction on its own.

- **dequeue_mode**
  This element is optional. It specifies the locking action associated with the dequeue. The possible values are `REMOVE`, `BROWSE`, and `LOCKED`.

  `REMOVE` is the default and causes the message to be read and deleted. The message can be retained in the queue table based on the retention properties. `BROWSE` reads the message without acquiring any lock on it. This is equivalent to a select statement. `LOCKED` reads the message and obtains a write lock on it. The lock lasts for the duration of the transaction. This is equivalent to a select for update statement.
- **navigation_mode**

  This element is optional. It specifies the position of the message that is retrieved. First, the position is determined. Second, the search criteria are applied. Finally, the message is retrieved. Possible values are **FIRST_MESSAGE**, **NEXT_MESSAGE**, and **NEXT_TRANSACTION**.

  **FIRST_MESSAGE** retrieves the first message which is available and which matches the search criteria. This resets the position to the beginning of the queue. **NEXT_MESSAGE** is the default and retrieves the next message which is available and which matches the search criteria. If the previous message belongs to a message group, then Oracle Database retrieves the next available message which matches the search criteria and which belongs to the message group. **NEXT_TRANSACTION** skips the remainder of the current transaction group and retrieves the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue.

- **transformation**

  This element is optional. It specifies the PL/SQL **transformation** to be invoked after the message is dequeued.

**AQXmlCommit**

This is an optional empty element. If it is included, then the transaction is committed at the end of the request.

**IDAP Client Requests for Registration**

Client requests for registration use AQXmlRegister, which must contain a **register_options** element. The **register_options** element contains the following child elements:

- **destination**

  This element is required. It specifies the queue/topic on which notifications are registered. The **destination** element has an optional **lookup_type** attribute, which determines how the destination value is interpreted. If **lookup_type** is **DATABASE**, which is the default, then the destination is interpreted as **schema.queue_name**. If **lookup_type** is **LDAP**, then the LDAP server is used to resolve the destination.

- **consumer_name**

  This element is optional. It specifies the consumer name for multiconsumer queues or topics. This parameter must not be specified for single-consumer queues.

- **notify_url**

  This element is required. It specifies where notification is sent when a message is enqueued. The form can be **http://url**, **mailto://email address** or **plsql://pl/sql procedure**.

**IDAP Client Requests to Commit a Transaction**

A request to commit all actions performed by the user in a session uses AQXmlCommit. A commit request has the following format:

```xml
<?xml version='1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <AQXmlCommit xmlns="http://ns.oracle.com/AQ/schemas/access"/>
  </Body>
</Envelope>
```
IDAP Client Requests to Roll Back a Transaction

A request to roll back all actions performed by the user in a session uses AQXmlRollback. Actions performed with IMMEDIATE visibility are not rolled back. An IDAP client rollback request has the following format:

```xml
<?xml version="1.0"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <AQXmlRollback xmlns="http://ns.oracle.com/AQ/schemas/access"/>
  </Body>
</Envelope>
```

IDAP Server Response to an Enqueue Request

The response to an enqueue request to a single-consumer queue uses AQXmlSendResponse. It contains the following elements:

- **status_response**
  - This element contains child elements status_code, error_code, and error_message. The status_code element takes value 0 for success or -1 for failure. The error_code element contains an Oracle error code. The error_message element contains a description of the error.

- **send_result**
  - This element contains child elements destination and message_id. The destination element specifies where the message was sent. The message_id element uniquely identifies every message sent.

The response to an enqueue request to a multiconsumer queue or topic uses AQXmlPublishResponse. It contains the following elements:

- **status_response**
  - This element contains child elements status_code, error_code, and error_message. The status_code element takes value 0 for success or -1 for failure. The error_code element contains an Oracle error code. The error_message element contains a description of the error.

- **publish_result**
  - This element contains child elements destination and message_id. The destination element specifies where the message was sent. The message_id element uniquely identifies every message sent.

IDAP Server Response to a Dequeue Request

The response to a dequeue request uses AQXmlReceiveResponse. It contains the following elements:

- **status_response**
  - This element contains child elements status_code, error_code, and error_message. The status_code element takes value 0 for success or -1 for failure. The error_code element contains an Oracle error code. The error_message element contains a description of the error.
receive_result

This element contains child elements destination and message_set. The destination element specifies where the message was sent. The message_set element specifies the set of messages dequeued.

**IDAP Server Response to a Register Request**

The response to a register request uses AQXmlRegisterResponse. It contains the status_response element described in "IDAP Server Response to a Dequeue Request" on page 6-14.

**IDAP Commit Response**

The response to a commit request uses AQXmlCommitResponse. It contains the status_response element described in "IDAP Server Response to a Dequeue Request" on page 6-14. The response to a commit request has the following format:

```xml
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <AQXmlCommitResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
      <status_response>
        <status_code>0</status_code>
      </status_response>
    </AQXmlCommitResponse>
  </Body>
</Envelope>
```

**IDAP Rollback Response**

The response to a rollback request uses AQXmlRollbackResponse. It contains the status_response element described in "IDAP Server Response to a Dequeue Request" on page 6-14.

**IDAP Notification**

When an event for which a client has registered occurs, a notification is sent to the client at the URL specified in the REGISTER request using AQXmlNotification. It contains the following elements:

- notification_options
  - This element has child elements destination and consumer_name. The destination element specifies the destination queue/topic on which the event occurred. The consumer_name element specifies the consumer name for which the event occurred. It applies only to multiconsumer queues/topics.

- message_set
  - This element specifies the set of message properties.

**IDAP Response in Case of Error**

In case of an error in any of the preceding requests, a FAULT is generated. The FAULT element contains the following elements:

- faultcode
  - This element specifies the error code for the fault.
Notification of Messages by e-mail

- **faultstring**
  This element indicates a client error or a server error. A client error means that the request is not valid. A server error indicates that the Oracle® Database servlet has not been set up correctly.

- **detail**
  This element contains the `status_response` element, which is described in "IDAP Server Response to a Dequeue Request" on page 6-14.

### Notification of Messages by e-mail

Here are the steps for setting up your database for e-mail notifications:

1. Set the SMTP mail host by invoking `DBMS_AQELM.SET_MAILHOST` as an Oracle® Database administrator.
2. Set the SMTP mail port by invoking `DBMS_AQELM.SET_MAILPORT` as an Oracle® Database administrator. If not explicit, set defaults to 25.
3. Set the SendFrom address by invoking `DBMS_AQELM.SET_SENDFROM`.
4. After setup, you can register for e-mail notifications using the Oracle Call Interface (OCI) or PL/SQL API.
This chapter describes how to troubleshoot Oracle Streams Advanced Queuing (AQ). The chapter contains these topics:

- Debugging Oracle® Database Propagation Problems
- Oracle® Database Error Messages

Debugging Oracle® Database Propagation Problems

The following tips should help with debugging propagation problems. This discussion assumes that you have created queue tables and queues in source and target databases and defined a database link for the destination database. The notation assumes that you supply the actual name of the entity (without the brackets).

See Also: “Optimizing Propagation” on page 4-9

To begin debugging, do the following:

1. Check that the propagation schedule has been created and that a job queue process has been assigned.

   Look for the entry in the DBA_QUEUE_SCHEDULES view and make sure that the status of the schedule is enabled. SCHEDULE_DISABLED must be set to ‘N’. Check that it has a nonzero entry for JOBNO in table AQ$$_SCHEDULES, and that there is an entry in table JOB$ with that JOBNO.

   To check if propagation is occurring, monitor the DBA_QUEUE_SCHEDULES view for the number of messages propagated (TOTAL_NUMBER).

   If propagation is not occurring, check the view for any errors. Also check the NEXT_RUN_DATE and NEXT_RUN_TIME in DBA_QUEUE_SCHEDULES to see if propagation is scheduled for a later time, perhaps due to errors or the way it is set up.

2. Check if the database link to the destination database has been set up properly. Make sure that the queue owner can use the database link. You can do this with:

   select count(*) from table_name@dblink_name;

3. Make sure that at least two job queue processes are running.

4. Check for messages in the source queue with:

   select count(*) from AQ$<source_queue_table>
   where q_name = 'source_queue_name';
5. Check for messages in the destination queue with:

   ```sql
   select count (*) from AQ$<destination_queue_table>
   where q_name = 'destination_queue_name';
   ```

6. Check to see who is using job queue processes.

   Check which jobs are being run by querying `dba_jobs_running`. It is possible that other jobs are starving the propagation jobs.

7. Check to see that the queue table `sys.aq$_prop_table_instno` exists in `DBA_QUEUES_TABLES`. The queue `sys.aq$_prop_notify_queue_instno` must also exist in `DBA_QUEUES` and must be enabled for enqueue and dequeue.

   In case of Real Application Clusters (RAC), this queue table and queue pair must exist for each RAC node in the system. They are used for communication between job queue processes and are automatically created.

8. Check that the consumer attempting to dequeue a message from the destination queue is a recipient of the propagated messages.

   For 8.1-style queues, you can do the following:

   ```sql
   select consumer_name, deq_txn_id, deq_time, deq_user_id,
   propagated_msgid from aq$<destination_queue_table>
   where queue = 'queue_name';
   ```

   For 8.0-style queues, you can obtain the same information from the history column of the queue table:

   ```sql
   select h.consumer, h.transaction_id, h.deq_time, h.deq_user,
   h.propagated_msgid from aq$<destination_queue_table> t, table(t.history) h
   where t.q_name = 'queue_name';
   ```

   **Note:** Queues created in a queue table with `compatible` set to 8.0 (referred to in this guide as 8.0-style queues) are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

9. Turn on propagation tracing at the highest level using event 24040, level 10.

   Debugging information is logged to job queue trace files as propagation takes place. You can check the trace file for errors and for statements indicating that messages have been sent.

**Oracle® Database Error Messages**

**ORA-1555**

You might get this error when using the `NEXT_MESSAGE` navigation option for dequeue. `NEXT_MESSAGE` uses the snapshot created during the first dequeue call. After that, undo information may not be retained.

The workaround is to use the `FIRST_MESSAGE` option to dequeue the message. This reexecutes the cursor and gets a new snapshot. `FIRST_MESSAGE` does not perform as well as `NEXT_MESSAGE`, so Oracle recommends that you dequeue messages in batches: `FIRST_MESSAGE` for one, `NEXT_MESSAGE` for the next 1000 messages, then `FIRST_MESSAGE` again, and so on.
ORA-24033
This error is raised if a message is enqueued to a multiconsumer queue with no recipient and the queue has no subscribers (or rule-based subscribers that match this message). This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

ORA-25237
When using the Oracle® Database navigation option, you must reset the dequeue position by using the FIRST_MESSAGE option if you want to continue dequeuing between services (such as xa_start and xa_end boundaries). This is because XA cancels the cursor fetch state after an xa_end. If you do not reset, then you get an error message stating that the navigation is used out of sequence.

ORA-25307
Flow control has been enabled for the message sender. This means that the fastest subscriber of the sender’s message is not able to keep pace with the rate at which messages are enqueued. The buffered messaging application must handle this error and attempt again to enqueue messages after waiting for some time.
This chapter describes the Oracle Streams Advanced Queuing (AQ) administrative interface.

This chapter contains these topics:

- Managing Queue Tables
- Managing Queues
- Managing Transformations
- Granting and Revoking Privileges
- Managing Subscribers
- Managing Propagations
- Managing Oracle® Database Agents
- Adding an Alias to the LDAP Server
- Deleting an Alias from the LDAP Server

See Also:

- Chapter 3, "Oracle® Database: Programmatic Interfaces" for a list of available functions in each programmatic interface
- Oracle Database PL/SQL Packages and Types Reference for information on the DBMS_AQADM Package

Managing Queue Tables

This section contains these topics:

- Creating a Queue Table
- Altering a Queue Table
- Dropping a Queue Table
- Purging a Queue Table
- Migrating a Queue Table

Creating a Queue Table

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table IN VARCHAR2,
    queue_payload_type IN VARCHAR2,
```
This procedure creates a queue table for messages of a predefined type. It has the following required and optional parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>This required parameter specifies the queue table name. Mixed case (upper and lower case together) queue table names are supported if database compatibility is 10.0, but the names must be enclosed in double quote marks. So abc.efg means the schema is ABC and the name is EFG, but &quot;abc&quot;,&quot;efg&quot; means the schema is abc and the name is efg. Queue table names must not be longer than 24 characters. If you attempt to create a queue table with a longer name, error ORA-24019 results.</td>
</tr>
<tr>
<td>queue_payload_type</td>
<td>This required parameter specifies the payload type as RAW or an object type. See “Payload Type” on page 8-3 for more information.</td>
</tr>
<tr>
<td>storage_clause</td>
<td>This optional parameter specifies a tablespace for the queue table. See “Storage Clause” on page 8-3 for more information.</td>
</tr>
<tr>
<td>sort_list</td>
<td>This optional parameter specifies one or two columns to be used as sort keys in ascending order. It has the format sort_column1,sort_column2. See “Sort Key” on page 8-4 for more information.</td>
</tr>
<tr>
<td>multiple_consumers</td>
<td>This optional parameter specifies the queue table as single-consumer or multiconsumer. The default FALSE means queues created in the table can have only one consumer for each message. TRUE means queues created in the table can have multiple consumers for each message.</td>
</tr>
<tr>
<td>message_grouping</td>
<td>This optional parameter specifies whether messages are grouped or not. The default NONE means each message is treated individually. TRANSACTIONAL means all messages enqueued in one transaction are considered part of the same group and can be dequeued as a group of related messages.</td>
</tr>
<tr>
<td>comment</td>
<td>This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.</td>
</tr>
<tr>
<td>primary_instance</td>
<td>This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value 0 means queue monitor scheduling and propagation is done in any available instance. You can specify and modify this parameter only if compatible is 8.1 or higher.</td>
</tr>
<tr>
<td>secondary_instance</td>
<td>This optional parameter specifies the owner of the queue table if the primary instance is not available. The default value 0 means that the queue table will fail over to any available instance. You can specify and modify this parameter only if primary_instance is also specified and compatible is 8.1 or higher.</td>
</tr>
</tbody>
</table>
Managing Queue Tables

Payload Type
To specify the payload type as an object type, you must define the object type.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compatible</td>
<td>This optional parameter specifies the lowest database version with which the queue table is compatible. The possible values are 8.0, 8.1, and 10.0. If the database is in 10.1-compatible mode, then the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, then the default value is 8.1. If the database is in 8.0-compatible mode, then the default value is 8.0. The 8.0 value is deprecated in Oracle Database 10g Release 2 (10.2). For more information on compatibility, see &quot;Oracle Database Compatibility Parameters&quot; on page 4-1.</td>
</tr>
<tr>
<td>secure</td>
<td>This optional parameter must be set to TRUE if you want to use the queue table for secure queues. Secure queues are queues for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot unless they are configured as secure queue users.</td>
</tr>
</tbody>
</table>

Payload Type
To specify the payload type as an object type, you must define the object type.

Note: If you have created synonyms on object types, then you cannot use them in DBMS_AQADM.CREATE_QUEUE_TABLE. Error ORA-24015 results.

CLOB, BLOB, and BFILE objects are valid in an Oracle® Database message. You can propagate these object types using Oracle® Database propagation with Oracle software since Oracle8i release 8.1.x. To enqueue an object type that has a LOB, you must first set the LOB_attribute to EMPTY_BLOB() and perform the enqueue. You can then select the LOB locator that was generated from the queue table's view and use the standard LOB operations.

Note: Payloads containing LOBs require users to grant explicit Select, Insert and Update privileges on the queue table for doing enqueues and dequeues.

Storage Clause
The storage_clause argument can take any text that can be used in a standard CREATE TABLE storage_clause argument.

Once you pick the tablespace, any index-organized table (IOT) or index created for that queue table goes to the specified tablespace. You do not currently have a choice to split them between different tablespaces.
Sort Key
The sort_list parameter determines the order in which messages are dequeued. You cannot change the message sort order after you have created the queue table. Your choices are:

- ENQ_TIME
- ENQ_TIME, PRIORITY
- PRIORITY
- PRIORITY, ENQ_TIME
- PRIORITY, COMMIT_TIME
- COMMIT_TIME

The COMMIT_TIME choice is a new feature in Oracle® Database 10g Release 2 (10.2). If it is specified, then any queue that uses the queue table is a commit-time queue, and Oracle® Database computes an approximate CSCN for each enqueued message when its transaction commits.

If you specify COMMIT_TIME as the sort key, then you must also specify the following:

- multiple_consumers = TRUE
- message_grouping = TRANSACTIONAL
- compatible = 8.1 or higher

Commit-time ordering is useful when transactions are interdependent or when browsing the messages in a queue must yield consistent results.

See Also:
- "Commit-Time Queues" in Oracle Streams Concepts and Administration
- "Dequeue Modes" on page 1-22

Other Tables and Views
The following objects are created at table creation time:

- AQ$_queue_table_name, a read-only view which is used by Oracle® Database applications for querying queue data
- AQ$_queue_table_name_E, the default exception queue associated with the queue table
Managing Queue Tables

Oracle® Database Administrative Interface

Managing Queue Tables

- AQ$_queue_table_name_I, an index or an index-organized table (IOT) in the case of multiple consumer queues for dequeue operations
- AQ$_queue_table_name_T, an index for the queue monitor operations

The following objects are created only for 8.1-compatible multiconsumer queue tables:
- AQ$_queue_table_name_S, a table for storing information about subscribers
- AQ$_queue_table_name_H, an index organized table (IOT) for storing dequeue history data
- AQ$_queue_table_name_L, dequeue log table, used for storing message identifiers of committed dequeue operations on the queue

---

**Note:** Oracle® Database does not support the use of triggers on these internal AQ queue tables.

---

**See Also:** Oracle Database SecureFiles and Large Objects Developer’s Guide

If you do not specify a schema, then you default to the user's schema.

If GLOBAL_TOPIC_ENABLED = TRUE when a queue table is created, then a corresponding Lightweight Directory Access Protocol (LDAP) entry is also created.

If the queue type is ANYDATA, then a buffered queue and two additional objects are created. The buffered queue stores logical change records created by a capture process. The logical change records are staged in a memory buffer associated with the queue; they are not ordinarily written to disk.

If they have been staged in the buffer for a period of time without being dequeued, or if there is not enough space in memory to hold all of the captured events, then they are spilled to:
- AQ$_queue_table_name_P, a table for storing the captured events that spill from memory
- AQ$_queue_table_name_D, a table for storing information about the propagations and apply processes that are eligible for processing each event

**See Also:** Chapter 3, "Streams Staging and Propagation" in Oracle Streams Concepts and Administration

Examples

The following examples assume you are in a SQL*Plus testing environment. In Example 8–1, you create users in preparation for the other examples in this chapter. For this example, you must connect as a user with administrative privileges. For most of the other examples in this chapter, you can connect as user test_adm. A few examples must be run as test with EXECUTE privileges on DBMS_AQADM.

**Example 8–1 Setting Up AQ Administrative Users**

```sql
CREATE USER test_adm IDENTIFIED BY test_adm DEFAULT TABLESPACE example;
GRANT DBA, CREATE ANY TYPE TO test_adm;
GRANT EXECUTE ON DBMS_AQADM TO test_adm;
GRANT aq_administrator_role TO test_adm;
BEGIN
  DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
```

Oracle® Database Administrative Interface 8-5
privilege => 'MANAGE ANY',
grantee => 'test_adm',
admin_option => FALSE);
END;
/
CREATE USER test IDENTIFIED BY test;
GRANT DBA TO test;
GRANT EXECUTE ON dbms_aq TO test;

Example 8–2 Setting Up AQ Administrative Example Types

CREATE TYPE test.message_typ AS object(
  sender_id NUMBER,
  subject VARCHAR2(30),
  text VARCHAR2(1000));
/
CREATE TYPE test.msg_table AS TABLE OF test.message_typ;
/
CREATE TYPE test.order_typ AS object(
  custno NUMBER,
  item VARCHAR2(30),
  description VARCHAR2(1000));
/
CREATE TYPE test.lob_typ AS object(
  id NUMBER,
  subject VARCHAR2(100),
  data BLOB,
  trailer NUMBER);
/

Example 8–3 Creating a Queue Table for Messages of Object Type

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'test.obj_qtab',
    queue_payload_type => 'test.message_typ');
END;
/

Example 8–4 Creating a Queue Table for Messages of RAW Type

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'test.raw_qtab',
    queue_payload_type => 'RAW');
END;
/

Example 8–5 Creating a Queue Table for Messages of LOB Type

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'test.lob_qtab',
    queue_payload_type => 'test.lob_typ');
END;
/

Example 8–6 Creating a Queue Table for Messages of XMLType

BEGIN
Managing Queue Tables

```sql
DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'test.xml_qtab',
    queue_payload_type  => 'SYS.XMLType',
    multiple_consumers  => TRUE,
    compatible          => '8.1',
    comment             => 'Oversea Shipping multiconsumer orders queue table');
END;
/

Example 8–7  Creating a Queue Table for Grouped Messages
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table          => 'test.group_qtab',
        queue_payload_type   => 'test.message_typ',
        message_grouping     => DBMS_AQADM.TRANSACTIONAL);
END;
/

Example 8–8  Creating Queue Tables for Prioritized Messages and Multiple Consumers
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'test.priority_qtab',
        queue_payload_type => 'test.order_typ',
        sort_list => 'PRIORITY,ENQ_TIME',
        multiple_consumers => TRUE);
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'test.multiconsumer_qtab',
        queue_payload_type => 'test.message_typ',
        sort_list => 'PRIORITY,ENQ_TIME',
        multiple_consumers => TRUE);
END;
/

Example 8–9  Creating a Queue Table with Commit-Time Ordering
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'test.commit_time_qtab',
        queue_payload_type => 'test.message_typ',
        sort_list => 'COMMIT_TIME',
        multiple_consumers => TRUE,
        message_grouping => DBMS_AQADM.TRANSACTIONAL,
        compatible => '10.0');
END;
/

Example 8–10  Creating an 8.1-Compatible Queue Table for Multiple Consumers
BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'test.multiconsumer_81_qtab',
        queue_payload_type => 'test.message_typ',
        multiple_consumers => TRUE,
        compatible => '8.1');
END;
/
Managing Queue Tables

Example 8–11 Creating a Queue Table in a Specified Tablespace

BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table        => 'test.example_qtab',
        queue_payload_type => 'test.message_typ',
        storage_clause     => 'tablespace example');
END;
/

Example 8–12 Creating a Queue Table with Freelists or Freelist Groups

BEGIN
    DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table            => 'test.freelist_qtab',
        queue_payload_type     => 'RAW',
        storage_clause         => 'STORAGE (FREELISTS 4 FREELIST GROUPS 2)',
        compatible             => '8.1');
END;
/

Altering a Queue Table

DBMS_AQADM.ALTER_QUEUE_TABLE ( 
    queue_table          IN   VARCHAR2,
    comment              IN   VARCHAR2       DEFAULT NULL,
    primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
    secondary_instance   IN   BINARY_INTEGER DEFAULT NULL);

This procedure alters the existing properties of a queue table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>This required parameter specifies the queue table name.</td>
</tr>
<tr>
<td>comment</td>
<td>This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.</td>
</tr>
<tr>
<td>primary_instance</td>
<td>This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. You can specify and modify this parameter only if compatible is 8.1 or higher.</td>
</tr>
<tr>
<td>secondary_instance</td>
<td>This optional parameter specifies the owner of the queue table if the primary instance is not available. You can specify and modify this parameter only if primary_instance is also specified and compatible is 8.1 or higher.</td>
</tr>
</tbody>
</table>

Note: In general, DDL statements are not supported on queue tables and may even render them inoperable. For example, issuing an ALTER TABLE ... SHRINK statement against a queue table results in an internal error, and all subsequent attempts to use the queue table will also result in errors. Oracle recommends that you not use DDL statements on queue tables.

If GLOBAL_TOPIC_ENABLED = TRUE when a queue table is modified, then a corresponding LDAP entry is also altered.
Example 8–13  Altering a Queue Table by Changing the Primary and Secondary Instances

BEGIN
    DBMS_AQADM.ALTER_QUEUE_TABLE(
        queue_table => 'test.obj_qtab',
        primary_instance => 3,
        secondary_instance => 2);
END;
/

Example 8–14  Altering a Queue Table by Changing the Comment

BEGIN
    DBMS_AQADM.ALTER_QUEUE_TABLE(
        queue_table => 'test.obj_qtab',
        comment => 'revised usage for queue table');
END;
/

Dropping a Queue Table

DBMS_AQADM.DROP_QUEUE_TABLE(
    queue_table => 'test.obj_qtab',
    force => TRUE);

This procedure drops an existing queue table. You must stop and drop all the queues in a queue table before the queue table can be dropped. You must do this explicitly if force is set to FALSE. If force is set to TRUE, then all queues in the queue table and their associated propagation schedules are dropped automatically.

If GLOBAL_TOPIC_ENABLED = TRUE when a queue table is dropped, then a corresponding LDAP entry is also dropped.

Example 8–15  Dropping a Queue Table

BEGIN
    DBMS_AQADM.DROP_QUEUE_TABLE(
        queue_table => 'test.obj_qtab');
END;
/

Example 8–16  Dropping a Queue Table with force Option

BEGIN
    DBMS_AQADM.DROP_QUEUE_TABLE(
        queue_table => 'test.raw_qtab',
        force => TRUE);
END;
/

Purging a Queue Table

DBMS_AQADM.PURGE_QUEUE_TABLE(
    queue_table => 'test.obj_qtab',
    purge_condition => 'true',
    purge_options => 'ao$purge_options_t');

This procedure purges messages from a queue table. It has the following parameters:
A trace file is generated in the udump destination when you run this procedure. It details what the procedure is doing. The procedure commits after it has processed all the messages.

**See Also:** "DBMS_AQADM" in Oracle Database PL/SQL Packages and Types Reference for more information on DBMS_AQADM.PURGE_QUEUE_TABLE

### Example 8–17 Purging All Messages in a Queue Table

```plsql
DECLARE
    po dbms_aqadm.aq$_purge_options_t;
BEGIN
    po.block := FALSE;
    DBMS_AQADM.PURGE_QUEUE_TABLE(
        queue_table     => 'test.obj_qtab',
        purge_condition => NULL,
        purge_options   => po);
END;
/
```

### Example 8–18 Purging All Messages in a Named Queue

```plsql
DECLARE
    po dbms_aqadm.aq$_purge_options_t;
BEGIN
    po.block := TRUE;
    DBMS_AQADM.PURGE_QUEUE_TABLE(
        queue_table     => 'test.obj_qtab',
        purge_condition => 'qtview.queue = ''TEST.OBJ_QUEUE''',
        purge_options   => po);
END;
/
```
Example 8–19  Purging All PROCESSED Messages in a Named Queue

DECLARE
  po dbms_aqadm.aq$_purge_options_t;
BEGIN
  po.block := TRUE;
  DBMS_AQADM.PURGE_QUEUE_TABLE(
    queue_table => 'test.obj_qtab',
    purge_condition => 'qtview.queue = ''TEST.OBJ_QUEUE''
                        and qtview.msg_state = ''PROCESSED''',
    purge_options => po);
END;
/

Example 8–20  Purging All Messages in a Named Queue and for a Named Consumer

DECLARE
  po dbms_aqadm.aq$_purge_options_t;
BEGIN
  po.block := TRUE;
  DBMS_AQADM.PURGE_QUEUE_TABLE(
    queue_table => 'test.multiconsumer_81_qtab',
    purge_condition => 'qtview.queue = ''TEST.MULTICONSUMER_81_QUEUE''
                        and qtview.consumer_name = ''PAYROLL_APP''',
    purge_options => po);
END;
/

Note: Some purge conditions, such as consumer_name in Example 8–20 and sender_name in Example 8–21, are supported only in 8.1-compatible queue tables. For more information, see Table 9–1, “AQ$Queue_Table_Name View” on page 9-4.

Example 8–21  Purging All Messages from a Named Sender

DECLARE
  po dbms_aqadm.aq$_purge_options_t;
BEGIN
  po.block := TRUE;
  DBMS_AQADM.PURGE_QUEUE_TABLE(
    queue_table => 'test.multiconsumer_81_qtab',
    purge_condition => 'qtview.sender_name = ''TEST.OBJ_QUEUE''',
    purge_options => po);
END;
/

Migrating a Queue Table

DBMS_AQADM.MIGRATE_QUEUE_TABLE(
  queue_table IN VARCHAR2,
  compatible IN VARCHAR2);

This procedure migrates a queue table from 8.0, 8.1, or 10.0 to 8.0, 8.1, or 10.0. Only the owner of the queue table can migrate it.
If a schema was created by an import of an export dump from a lower release or has Oracle® Database queues upgraded from a lower release, then attempts to drop it with DROP USER CASCADE will fail with ORA-24005. To drop such schemas:

1. Event 10851 should be set to level 1.
2. Drop all tables of the form AQ$$_queue_table_name_NR from the schema.
3. Turn off event 10851.
4. Drop the schema.

Example 8–22  Upgrading a Queue Table from 8.1-Compatible to 10.0-Compatible

BEGIN
  DBMS_AQADM.MIGRATE_QUEUE_TABLE (
    queue_table => 'test.xml_qtab',
    compatible  => '10.0');
END;
/

Managing Queues

This section contains these topics:

- Creating a Queue
- Altering a Queue
- Starting a Queue
- Stopping a Queue
- Dropping a Queue

Creating a Queue

DBMS_AQADM.CREATE_QUEUE(
  queue_name IN VARCHAR2,
  queue_table IN VARCHAR2,
  queue_type IN BINARY_INTEGER DEFAULT NORMAL_QUEUE,
  max_retries IN NUMBER DEFAULT NULL,
  retry_delay IN NUMBER DEFAULT 0,
  retention_time IN NUMBER DEFAULT 0,
  dependency_tracking IN BOOLEAN DEFAULT FALSE,
This procedure creates a queue. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>This required parameter specifies the name of the new queue.</td>
</tr>
<tr>
<td>queue_table</td>
<td>This required parameter specifies the queue table in which the queue is</td>
</tr>
<tr>
<td>queue_type</td>
<td>This parameter specifies what type of queue to create. The default NORMAL_</td>
</tr>
<tr>
<td>max_retries</td>
<td>This parameter limits the number of times a dequeue with the REMOVE mode</td>
</tr>
<tr>
<td>retry_delay</td>
<td>This parameter specifies the number of seconds after which this message</td>
</tr>
<tr>
<td>retention_time</td>
<td>This parameter specifies the number of seconds a message is retained in the</td>
</tr>
<tr>
<td>dependency_tracking</td>
<td>This parameter is reserved for future use. FALSE is the default.</td>
</tr>
<tr>
<td>comment</td>
<td>This optional parameter is a user-specified description of the queue.</td>
</tr>
</tbody>
</table>

All queue names must be unique within a schema. Once a queue is created with CREATE_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled. To view retained messages, you can either dequeue by message ID or use SQL. If GLOBAL_TOPIC_ENABLED = TRUE when a queue is created, then a corresponding LDAP entry is also created.

The following examples (Example 8–23 through Example 8–30) use data structures created in Example 8–1 through Example 8–12.

**Example 8–23  Creating a Queue for Messages of Object Type**

```sql
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
  ```
Example 8–24  Creating a Queue for Messages of RAW Type
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.raw_queue',
    queue_table => 'test.raw_qtab');
END;
/

Example 8–25  Creating a Queue for Messages of LOB Type
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.lob_queue',
    queue_table => 'test.lob_qtab');
END;
/

Example 8–26  Creating a Queue for Grouped Messages
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.group_queue',
    queue_table => 'test.group_qtab');
END;
/

Example 8–27  Creating a Queue for Prioritized Messages
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.priority_queue',
    queue_table => 'test.priority_qtab');
END;
/

Example 8–28  Creating a Queue for Prioritized Messages and Multiple Consumers
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.multiconsumer_queue',
    queue_table => 'test.multiconsumer_qtab');
END;
/

Example 8–29  Creating a Queue to Demonstrate Propagation
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'test.another_queue',
    queue_table => 'test.multiconsumer_qtab');
END;
/
Example 8–30  Creating an 8.1-Style Queue for Multiple Consumers

BEGIN
    DBMS_AQADM.CREATE_QUEUE(
        queue_name    => 'test.multiconsumer_81_queue',
        queue_table   => 'test.multiconsumer_81_qtab');
END;
/

Altering a Queue

DBMS_AQADM.ALTER_QUEUE(
    queue_name    IN VARCHAR2,
    max_retries   IN NUMBER   DEFAULT NULL,
    retry_delay   IN NUMBER   DEFAULT NULL,
    retention_time IN NUMBER   DEFAULT NULL,
    comment       IN VARCHAR2 DEFAULT NULL);

This procedure alters existing properties of a queue. Only max_retries, comment, retry_delay, and retention_time can be altered. To view retained messages, you can either dequeue by message ID or use SQL. If GLOBAL_TOPIC_ENABLED = TRUE when a queue is modified, then a corresponding LDAP entry is also altered.

Example 8–31 changes retention time, saving messages for 1 day after dequeuing.

Example 8–31  Altering a Queue by Changing Retention Time

BEGIN
    DBMS_AQADM.ALTER_QUEUE(
        queue_name    => 'test.another_queue',
        retention_time => 86400);
END;
/

Starting a Queue

DBMS_AQADM.START_QUEUE(
    queue_name IN VARCHAR2,
    enqueue    IN BOOLEAN DEFAULT TRUE,
    dequeue    IN BOOLEAN DEFAULT TRUE);

This procedure enables the specified queue for enqueuing or dequeuing. After creating a queue, the administrator must use START_QUEUE to enable the queue. The default is to enable it for both enqueue and dequeue. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Example 8–32  Starting a Queue with Both Enqueue and Dequeue Enabled

BEGIN
    DBMS_AQADM.START_QUEUE(
        queue_name => 'test.obj_queue');
END;
/
Example 8–33  Starting a Queue for Dequeue Only

BEGIN
  DBMS_AQADM.START_QUEUE(
    queue_name => 'test.raw_queue',
    dequeue    => TRUE,
    enqueue    => FALSE);
END;
/

Stopping a Queue

DBMS_AQADM.STOP_QUEUE(
  queue_name IN VARCHAR2,
  enqueue IN BOOLEAN DEFAULT TRUE,
  dequeue IN BOOLEAN DEFAULT TRUE,
  wait IN BOOLEAN DEFAULT TRUE);

This procedure disables enqueuing, dequeuing, or both on the specified queue.

By default, this call disables both enqueue and dequeue. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Example 8–34  Stopping a Queue

BEGIN
  DBMS_AQADM.STOP_QUEUE(
    queue_name => 'test.obj_queue');
END;
/

Dropping a Queue

DBMS_AQADM.DROP_QUEUE(
  queue_name IN VARCHAR2,

This procedure drops an existing queue. DROP_QUEUE is not allowed unless STOP_QUEUE has been called to disable the queue for both enqueuing and dequeuing. All the queue data is deleted as part of the drop operation.

If GLOBAL_TOPIC_ENABLED = TRUE when a queue is dropped, then a corresponding LDAP entry is also dropped.

Example 8–35  Dropping a Standard Queue

BEGIN
  DBMS_AQADM.DROP_QUEUE(
    queue_name => 'test.obj_queue');
END;
/

Managing Transformations

Transformations change the format of a message, so that a message created by one application can be understood by another application. You can use transformations on both persistent and buffered messages.

This section contains these topics:

- Creating a Transformation
Managing Transformations

- Modifying a Transformation
- Dropping a Transformation

Creating a Transformation

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema               VARCHAR2(30),
    name                 VARCHAR2(30),
    from_schema          VARCHAR2(30),
    from_type            VARCHAR2(30),
    to_schema            VARCHAR2(30),
    to_type              VARCHAR2(30),
    transformation       VARCHAR2(4000));
```

This procedure creates a message format transformation. The transformation must be a SQL function with input type from_type, returning an object of type to_type. It can also be a SQL expression of type to_type, referring to from_type. All references to from_type must be of the form source.user_data.

You must be granted EXECUTE privilege on dbms_transform to use this feature. This privilege is included in the AQ_ADMINISTRATOR_ROLE.

See Also: "Oracle® Database Security" on page 4-2 for more information on administrator and user roles

You must also have EXECUTE privilege on the user-defined types that are the source and destination types of the transformation, and have EXECUTE privileges on any PL/SQL function being used in the transformation function. The transformation cannot write the database state (that is, perform DML operations) or commit or rollback the current transaction.

Example 8–36  Creating a Transformation

```
BEGIN
    DBMS_TRANSFORM.CREATE_TRANSFORMATION(
        schema => 'test',
        name   => 'message_order_transform',
        from_schema => 'test',
        from_type => 'message_typ',
        to_schema => 'test',
        to_type => 'order_typ',
        transformation => 'test.order_typ(''
            source.user_data.sender_id,
            source.user_data.subject,
            source.user_data.text')');
END;
/
```

Modifying a Transformation

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
    schema            VARCHAR2(30),
    name              VARCHAR2(30),
    attribute_number  INTEGER,
    transformation    VARCHAR2(4000));
```

This procedure changes the transformation function and specifies transformations for each attribute of the target type. If the attribute number 0 is specified, then the
transformation expression singularly defines the transformation from the source to
target types.

All references to from_type must be of the form source.user_data. All references
to the attributes of the source type must be prefixed by source.user_data.

You must be granted EXECUTE privileges on dbms_transform to use this feature.
You must also have EXECUTE privileges on the user-defined types that are the source
and destination types of the transformation, and have EXECUTE privileges on any
PL/SQL function being used in the transformation function.

---

Dropping a Transformation

DBMS_TRANSFORM.DROP_TRANSFORMATION (
    schema      VARCHAR2(30),
    name        VARCHAR2(30));

This procedure drops a transformation.

You must be granted EXECUTE privileges on dbms_transform to use this feature.
You must also have EXECUTE privileges on the user-defined types that are the source
and destination types of the transformation, and have EXECUTE privileges on any
PL/SQL function being used in the transformation function.

---

Granting and Revoking Privileges

This section contains these topics:

- Granting Oracle® Database System Privileges
- Revoking Oracle® Database System Privileges
- Granting Queue Privileges
- Revoking Queue Privileges

Granting Oracle® Database System Privileges

DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
    privilege         IN    VARCHAR2,
    grantee           IN    VARCHAR2,
    admin_option      IN    BOOLEAN := FALSE);

This procedure grants Oracle® Database system privileges to users and roles. The
privileges are ENQUEUE_ANY, DEQUEUE_ANY, MANAGE_ANY. Initially, only SYS and
SYSTEM can use this procedure successfully.

Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any
queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to
dequeue messages from any queues in the database. Users granted the MANAGE_ANY
privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

---

Example 8–37  Granting AQ System Privileges

BEGIN
    DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
        privilege => 'ENQUEUE_ANY',
        grantee   => 'test',
        admin_option => FALSE);
    DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
        privilege => 'DEQUEUE_ANY',
        grantee   => 'test',
        admin_option => FALSE);
END;
Revoking Oracle® Database System Privileges

DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(
    privilege IN VARCHAR2,
    grantee   IN VARCHAR2);

This procedure revokes Oracle® Database system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.

Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to dequeue messages from any queues in the database. Users granted the MANAGE_ANY privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

Example 8–38 Revoking AQ System Privileges

BEGIN
    DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(
        privilege => 'DEQUEUE_ANY',
        grantee  => 'test');
END;
/

Granting Queue Privileges

DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
    privilege    IN VARCHAR2,
    queue_name   IN VARCHAR2,
    grantee      IN VARCHAR2,
    grant_option IN BOOLEAN := FALSE);

This procedure grants privileges on a queue to users and roles. The privileges are ENQUEUE, DEQUEUE, or ALL. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Caution: This procedure requires that EXECUTE privileges on DBMS_AQADM be granted to the queue table owner, who is probably an ordinary queue user. If you do not want ordinary queue users to be able to create and drop queues and queue tables, add and delete subscribers, and so forth, then you must revoke the EXECUTE privilege as soon as the initial GRANT_QUEUE_PRIVILEGE is done.

Example 8–39 Granting Queue Privilege

BEGIN
    DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
        privilege => 'ALL',
        queue_name => 'test.multiconsumer_81_queue',
        grantee    => 'test_adm',
        grant_option => TRUE);
END;
/
Revoking Queue Privileges

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (  
    privilege IN VARCHAR2,  
    queue_name IN VARCHAR2,  
    grantee IN VARCHAR2);  
```

This procedure revokes privileges on a queue from users and roles. The privileges are `ENQUEUE` or `DEQUEUE`.

To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the `GRANT` option are revoked if the grantor’s privileges are revoked.

You can revoke the dequeue right of a grantee on a specific queue, leaving the grantee with only the enqueue right as in Example 8–40.

**Example 8–40  Revoking Dequeue Privilege**

```
BEGIN  
    DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(  
        privilege => 'DEQUEUE',  
        queue_name => 'test.multiconsumer_81_queue',  
        grantee => 'test_adm');  
END;  
```

Managing Subscribers

This section contains these topics:

- Adding a Subscriber
- Altering a Subscriber
- Removing a Subscriber

Adding a Subscriber

```
DBMS_AQADM.ADD_SUBSCRIBER (  
    queue_name IN VARCHAR2,  
    subscriber IN sys.aq$_agent,  
    rule IN VARCHAR2 DEFAULT NULL,  
    transformation IN VARCHAR2 DEFAULT NULL,  
    queue_to_queue IN BOOLEAN DEFAULT FALSE,  
    delivery_mode IN PLS_INTEGER DEFAULT PERSISTENT);  
```

This procedure adds a default subscriber to a queue.

An application can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation succeeds only on queues that allow multiple consumers, and the total number of subscribers must be 1024 or less. This operation takes effect immediately and the containing transaction is committed. Enqueue requests that are executed after the completion of this call reflect the new action. Any string within the `rule` must be quoted (with single quotation marks) as follows:

```
rule  => 'PRIORITY <= 3 AND CORRID = ''FROM JAPAN'''  
```

User data properties or attributes apply only to object payloads and must be prefixed with `tab.userdata` in all cases.
If `GLOBAL_TOPIC_ENABLED` is set to true when a subscriber is created, then a corresponding LDAP entry is also created.

Specify the name of the transformation to be applied during dequeue or propagation. The transformation must be created using the `DBMS_TRANSFORM` package.

**See Also:** Oracle Database PL/SQL Packages and Types Reference for more information on the `DBMS_TRANSFORM` package

For queues that contain payloads with XMLType attributes, you can specify rules that contain operators such as `XMLType.existsNode()` and `XMLType.extract()`.

If parameter `queue_to_queue` is set to `TRUE`, then the added subscriber is a queue-to-queue subscriber. When queue-to-queue propagation is set up between a source queue and a destination queue, queue-to-queue subscribers receive messages through that propagation schedule.

**See Also:** "Scheduling a Queue Propagation" on page 8-24

If the `delivery_mode` parameter is the default `PERSISTENT`, then the subscriber receives only persistent messages. If it is set to `BUFFERED`, then the subscriber receives only buffered messages. If it is set to `PERSISTENT_OR_BUFFERED`, then the subscriber receives both types. You cannot alter this parameter with `ALTER_SUBSCRIBER`.

The agent name should be `NULL` if the destination queue is a single consumer queue.

**Note:** `ADD_SUBSCRIBER` is an administrative operation on a queue. Although Oracle Streams AQ does not prevent applications from issuing administrative and operational calls concurrently, they are executed serially. `ADD_SUBSCRIBER` blocks until pending calls that are enqueuing or dequeuing messages complete. It will not wait for the pending transactions to complete.

---

**Example 8–41   Adding a Subscriber at a Designated Queue at a Database Link**

```plsql
DECLARE
  subscriber  sys.aq$_agent;
BEGIN
  subscriber := sys.aq$_agent('subscriber1', 'test2.msg_queue2@london', null);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name => 'test.multiconsumer_81_queue',
    subscriber  =>  subscriber);
END;
/
```

**Example 8–42   Adding a Single Consumer Queue at a Database Link as a Subscriber**

```plsql
DECLARE
  subscriber  sys.aq$_agent;
BEGIN
  subscriber := sys.aq$_agent('subscriber1', 'test2.msg_queue2@london', null);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name => 'test.multiconsumer_81_queue',
    subscriber  =>  subscriber);
END;
/
```
Managing Subscribers

Example 8–43 Adding a Subscriber with a Rule

```sql
DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('subscriber2', 'test2.msg_queue2@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'test.multiconsumer_81_queue',
        subscriber => subscriber,
        rule => 'priority < 2');
END;
/
```

Example 8–44 Adding a Subscriber and Specifying a Transformation

```sql
DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('subscriber3', 'test2.msg_queue2@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'test.multiconsumer_81_queue',
        subscriber => subscriber,
        transformation => 'test.message_order_transform');
END;
/
```

Example 8–45 Propagating from a Multiple-Consumer Queue to a Single Consumer Queue

```sql
DECLARE
    subscriber SYS.AQ$_AGENT;
BEGIN
    subscriber := SYS.AQ$_AGENT(NULL, 'test2.single_consumer__queue@london', null);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'test.multiconsumer_81_queue',
        subscriber => subscriber);
END;
```

Altering a Subscriber

```sql
DBMS_AQADM.ALTER_SUBSCRIBER (
    queue_name IN VARCHAR2,
    subscriber IN sys.aq$_agent,
    rule IN VARCHAR2,
    transformation IN VARCHAR2);
```

This procedure alters existing properties of a subscriber to a specified queue.

The rule, the transformation, or both can be altered. If you alter only one of these attributes, then specify the existing value of the other attribute to the alter call. If GLOBAL_TOPIC_ENABLED = TRUE when a subscriber is modified, then a corresponding LDAP entry is created.

Example 8–46 Altering a Subscriber Rule

```sql
DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('subscriber2', 'test2.msg_queue2@london', null);
    DBMS_AQADM.ALTER_SUBSCRIBER(
```
queue_name => 'test.multiconsumer_81_queue',
subscriber => subscriber,
rule => 'priority = 1');

END;
/

Removing a Subscriber

DBMS_AQADM.REMOVE_SUBSCRIBER {
  queue_name IN VARCHAR2,
  subscriber IN sys.aq$_agent);

This procedure removes a default subscriber from a queue.

This operation takes effect immediately and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation. If GLOBAL_TOPIC_ENABLED = TRUE when a subscriber is dropped, then a corresponding LDAP entry is also dropped.

It is not an error to run the REMOVE_SUBSCRIBER procedure even when there are pending messages that are available for dequeue by the consumer. These messages are automatically made unavailable for dequeue when the REMOVE_SUBSCRIBER procedure finishes.

**Note:** REMOVE_SUBSCRIBER is an administrative operation on a queue. Although Oracle Streams AQ does not prevent applications from issuing administrative and operational calls concurrently, they are executed serially. REMOVE_SUBSCRIBER blocks until pending calls that are enqueuing or dequeuing messages complete. It will not wait for the pending transactions to complete.

**Example 8-47 Removing a Subscriber**

DECLARE
  subscriber sys.aq$_agent;
BEGIN
  subscriber := sys.aq$_agent ('subscriber2', 'test2.msg_queue2@london', null);
  DBMS_AQADM.REMOVE_SUBSCRIBER(
    queue_name => 'test.multiconsumer_81_queue',
    subscriber => subscriber);
END;
/

Managing Propagations

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue. You can also temporarily disable a schedule instead of dropping it. All administrative calls can be made irrespective of whether the schedule is active or not. If a schedule is active, then it takes a few seconds for the calls to be processed.

This section contains these topics:

- Scheduling a Queue Propagation
- Verifying Propagation Queue Type
- Altering a Propagation Schedule
Managing Propagations

- Enabling a Propagation Schedule
- Disabling a Propagation Schedule
- Unscheduling a Queue Propagation

Scheduling a Queue Propagation

\[
\text{DBMS_AQADM.SCHEDULE_PROPAGATION (}
\text{queue_name IN VARCHAR2,}
\text{destination IN VARCHAR2 DEFAULT NULL,}
\text{start_time IN DATE DEFAULT SYSDATE,}
\text{duration IN NUMBER DEFAULT NULL,}
\text{next_time IN VARCHAR2 DEFAULT NULL,}
\text{latency IN NUMBER DEFAULT 60,}
\text{destination_queue IN VARCHAR2 DEFAULT NULL);}
\]

This procedure schedules propagation of messages.

The destination can be identified by a database link in the `destination` parameter, a queue name in the `destination_queue` parameter, or both. Specifying only a database link results in queue-to-dblink propagation. If you propagate messages to several queues in another database, then all propagations have the same frequency.

If a private database link in the schema of the queue table owner has the same name as a public database link, AQ always uses the private database link.

Specifying the destination queue name results in queue-to-queue propagation was introduced in Oracle® Database 10g Release 2 (10.2). If you propagate messages to several queues in another database, queue-to-queue propagation enables you to configure each schedule independently of the others. You can enable or disable individual propagations.

**Note:** If you want queue-to-queue propagation to a queue in another database, then you must specify parameters `destination` and `destination_queue`.

Queue-to-queue propagation mode supports transparent failover when propagating to a destination Real Application Clusters (RAC) system. With queue-to-queue propagation, it is not required to repoint a database link if the owner instance of the queue fails on RAC.

Messages can also be propagated to other queues in the same database by specifying a `NULL` destination. If a message has multiple recipients at the same destination in either the same or different queues, then the message is propagated to all of them at the same time.

The source queue must be in a queue table meant for multiple consumers. If you specify a single-consumer queue, than error ORA-24039 results. Oracle® Database does not support the use of synonyms to refer to queues or database links.

If you specify a propagation `next_time` and `duration`, propagation will run periodically for the specified duration.

If you specify a latency of zero with no `next_time` or `duration`, the resulting propagation will run forever, propagating messages as they appear in the queue, and idling otherwise.

If a non-zero latency is specified, with no `next_time` or `duration` (default), the propagation schedule will be event-based. It will be scheduled to run when there are
messages in the queue to be propagated. When there are no more messages for a system-defined period of time, the job will stop running until there are new messages to be propagated.

The time at which the job runs depends on other factors, such as the number of ready jobs and the number of job queue processes.

See Also:

- "Managing Job Queues" in *Oracle Database Administrator's Guide* for more information on job queues and Jnnn background processes
- Chapter 6, "Internet Access to Oracle® Database"

Propagation uses a linear backoff scheme for retrying propagation from a schedule that encountered a failure. If a schedule continuously encounters failures, then the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, then the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts are made after which the schedule is automatically disabled.

Note: Once a retry attempt slips to the next propagation window, it will always do so; the exponential backoff scheme no longer governs retry scheduling. If the date function specified in the next_time parameter of `DBMS_AQADM.SCHEDULE_PROPAGATION` results in a short interval between windows, then the number of unsuccessful retry attempts can quickly reach 16, disabling the schedule.

If you specify a value for `destination` that does not exist, then this procedure still runs without throwing an error. You can query runtime propagation errors in the `LAST_ERROR_MSG` column of the `USER_QUEUE_SCHEDULES` view.

See Also: "USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema" on page 9-3

**Example 8–48 Scheduling a Propagation to Queues in the Same Database**

```sql
BEGIN
    DBMS_AQADM.SCHEDULE_PROPAGATION(
        queue_name    =>    'test.multiconsumer_queue');
END;
/
```

**Example 8–49 Scheduling a Propagation to Queues in Another Database**

```sql
BEGIN
    DBMS_AQADM.SCHEDULE_PROPAGATION(
        queue_name    =>    'test.multiconsumer_queue',
        destination   =>    'another_db.world');
END;
/
```

**Example 8–50 Scheduling Queue-to-Queue Propagation**

```sql
BEGIN
    DBMS_AQADM.SCHEDULE_PROPAGATION(
```
Verifying Propagation Queue Type

```sql
DBMS_AQADM.VERIFY_QUEUE_TYPES(
    src_queue_name IN VARCHAR2,
    dest_queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL,
    rc OUT BINARY_INTEGER);
```

This procedure verifies that the source and destination queues have identical types. The result of the verification is stored in the dictionary table `SYS.AQ$_MESSAGE_TYPES`, overwriting all previous output of this command.

If the source and destination queues do not have identical types and a transformation was specified, then the transformation must map the source queue type to the destination queue type.

---

**Note:** `SYS.AQ$_MESSAGE_TYPES` can have multiple entries for the same source queue, destination queue, and database link, but with different transformations.

---

**Example 8–51 Verifying a Queue Type**

```sql
SET SERVEROUTPUT ON
DECLARE
    rc BINARY_INTEGER;
BEGIN
    DBMS_AQADM.VERIFY_QUEUE_TYPES(
        src_queue_name => 'test.multiconsumer_queue',
        dest_queue_name => 'test.another_queue',
        rc => rc);
    DBMS_OUTPUT.PUT_LINE('Compatible: '||rc);
END;
/
```

**Example 8–51** involves two queues of the same type. It returns:

VQT: new style queue
Compatible: 1

If the same example is run with `test.raw_queue` (a queue of type RAW) in place of `test.another_queue`, then it returns:

VQT: new style queue
Compatible: 0

---

Altering a Propagation Schedule

```sql
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
    queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL,
    duration IN NUMBER DEFAULT NULL,
    next_time IN VARCHAR2 DEFAULT NULL,
```
latency IN NUMBER DEFAULT 60,
destination_queue IN VARCHAR2 DEFAULT NULL);

This procedure alters parameters for a propagation schedule. The destination_queue parameter for queue-to-queue propagation cannot be altered.

**Example 8–52  Altering a Propagation Schedule to Queues in the Same Database**

BEGIN
    DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue',
        duration   => '2000',
        next_time  => 'SYSDATE + 3600/86400',
        latency    => '32');
END;
/

**Example 8–53  Altering a Propagation Schedule to Queues in Another Database**

BEGIN
    DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue',
        destination => 'another_db.world',
        duration   => '2000',
        next_time  => 'SYSDATE + 3600/86400',
        latency    => '32');
END;
/

**Enabling a Propagation Schedule**

DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL,
    destination_queue IN VARCHAR2 DEFAULT NULL);

This procedure enables a previously disabled propagation schedule.

**Example 8–54  Enabling a Propagation to Queues in the Same Database**

BEGIN
    DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue');
END;
/

**Example 8–55  Enabling a Propagation to Queues in Another Database**

BEGIN
    DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue',
        destination => 'another_db.world');
END;
/

**Disabling a Propagation Schedule**

DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL,
This procedure disables a previously enabled propagation schedule.

**Example 8–56  Disabling a Propagation to Queues in the Same Database**

```
BEGIN
    DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue');
END;
/
```

**Example 8–57  Disabling a Propagation to Queues in Another Database**

```
BEGIN
    DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
        queue_name => 'test.multiconsumer_queue',
        destination => 'another_db.world');
END;
/
```

Unscheduling a Queue Propagation

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION(
    queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL,
    destination_queue IN VARCHAR2 DEFAULT NULL);
```

This procedure unschedules a previously scheduled propagation of messages from a queue to a destination. The destination is identified by a specific database link in the **destination** parameter or by name in the **destination_queue** parameter.

**Example 8–58  Unscheduling a Propagation to Queues in the Same Database**

```
BEGIN
    DBMS_AQADM.UNSCHEDULE_PROPAGATION(
        queue_name => 'test.multiconsumer_queue');
END;
/
```

**Example 8–59  Unscheduling a Propagation to Queues in Another Database**

```
BEGIN
    DBMS_AQADM.UNSCHEDULE_PROPAGATION(
        queue_name => 'test.multiconsumer_queue',
        destination => 'another_db.world');
END;
/
```

Managing Oracle® Database Agents

This section contains these topics:

- Creating an Oracle® Database Agent
- Altering an Oracle® Database Agent
- Dropping an Oracle® Database Agent
- Enabling Database Access
Disabling Database Access

Creating an Oracle® Database Agent

```
DBMS_AQADM.CREATE_AQ_AGENT (    
    agent_name                IN VARCHAR2,    
    certificate_location      IN VARCHAR2 DEFAULT NULL,    
    enable_http               IN BOOLEAN DEFAULT FALSE,    
    enable_anyp               IN BOOLEAN DEFAULT FALSE);
```

This procedure registers an agent for Oracle® Database Internet access using HTTP protocols.

The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle® Database Internet agents. When an agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- `GLOBAL_TOPIC_ENABLED = TRUE`
- `certificate_location` is specified

Altering an Oracle® Database Agent

```
DBMS_AQADM.ALTER_AQ_AGENT (    
    agent_name                IN VARCHAR2,    
    certificate_location      IN VARCHAR2 DEFAULT NULL,    
    enable_http               IN BOOLEAN DEFAULT FALSE,    
    enable_anyp               IN BOOLEAN DEFAULT FALSE);
```

This procedure alters an agent registered for Oracle® Database Internet access.

When an Oracle® Database agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- `GLOBAL_TOPIC_ENABLED = TRUE`
- `certificate_location` is specified

Dropping an Oracle® Database Agent

```
DBMS_AQADM.DROP_AQ_AGENT (    
    agent_name     IN VARCHAR2);
```

This procedure drops an agent that was previously registered for Oracle® Database Internet access.

When an Oracle® Database agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- `GLOBAL_TOPIC_ENABLED = TRUE`
- `certificate_location` is specified

Enabling Database Access

```
DBMS_AQADM.ENABLE_DB_ACCESS (    
    agent_name                IN VARCHAR2,    
    db_username               IN VARCHAR2)
```

Adding an Alias to the LDAP Server

This procedure grants an Oracle® Database Internet agent the privileges of a specific database user. The agent should have been previously created using the `CREATE_AQ_AGENT` procedure.

The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle® Database Internet agents and the names of the database users whose privileges are granted to them.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Disabling Database Access

```sql
DBMS_AQADM.DISABLE_DB_ACCESS (
    agent_name                IN VARCHAR2,
    db_username               IN VARCHAR2)
```

This procedure revokes the privileges of a specific database user from an Oracle® Database Internet agent. The agent should have been previously granted those privileges using the `ENABLE_DB_ACCESS` procedure.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Adding an Alias to the LDAP Server

```sql
DBMS_AQADM.ADD_ALIASES_TO_LDAP(
    alias          IN VARCHAR2,
    obj_location   IN VARCHAR2);
```

This procedure adds an alias to the LDAP server.

This call takes the name of an alias and the distinguished name of an Oracle® Database object in LDAP, and creates the alias that points to the Oracle® Database object. The alias is placed immediately under the distinguished name of the database server. The object to which the alias points can be a queue, an agent, or a ConnectionFactory.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Deleting an Alias from the LDAP Server

```sql
DBMS_AQADM.DEL_ALIASES_FROM_LDAP(
    alias IN VARCHAR2);
```

This procedure removes an alias from the LDAP server.

This call takes the name of an alias as the argument, and removes the alias entry in the LDAP server. It is assumed that the alias is placed immediately under the database server in the LDAP directory.
This chapter describes the Oracle Streams Advanced Queuing (AQ) administrative interface views and Oracle Messaging Gateway (MGW) views.

**Note:** All views not detailed in this chapter are described in the *Oracle Database Reference*.

This chapter contains these topics:

**Oracle AQ Views**
- `DBA_QUEUE_TABLES`: All Queue Tables in Database
- `USER_QUEUE_TABLES`: Queue Tables in User Schema
- `ALL_QUEUE_TABLES`: Queue Tables Queue Accessible to the Current User
- `DBA_QUEUES`: All Queues in Database
- `USER_QUEUES`: Queues In User Schema
- `ALL_QUEUES`: Queues for Which User Has Any Privilege
- `DBA_QUEUE_SCHEDULES`: All Propagation Schedules
- `USER_QUEUE_SCHEDULES`: Propagation Schedules in User Schema
- `QUEUE_PRIVILEGES`: Queues for Which User Has Queue Privilege
- `AQ$Queue_Table_Name`: Messages in Queue Table
- `AQ$Queue_Table_Name_S`: Queue Subscribers
- `AQ$Queue_Table_Name_R`: Queue Subscribers and Their Rules
- `DBA_QUEUE_SUBSCRIBERS`: All Queue Subscribers in Database
- `USER_QUEUE_SUBSCRIBERS`: Queue Subscribers in User Schema
- `ALL_QUEUE_SUBSCRIBERS`: Subscribers for Queues Where User Has Queue Privileges
- `DBA_TRANSFORMATIONS`: All Transformations
- `DBA_ATTRIBUTE_TRANSFORMATIONS`: All Transformation Functions
- `USER_TRANSFORMATIONS`: User Transformations
- `USER_ATTRIBUTE_TRANSFORMATIONS`: User Transformation Functions
DBA_QUEUE_TABLES: All Queue Tables in Database

The DBA_QUEUE_TABLES view contains information about the owner instance for a queue table. A queue table can contain multiple queues. In this case, each queue in a
queue table has the same owner instance as the queue table. Its columns are the same as those in ALL_QUEUE_TABLES.

**USER_QUEUE_TABLES: Queue Tables in User Schema**

The USER_QUEUE_TABLES view is the same as DBA_QUEUE_TABLES with the exception that it only shows queue tables in the user's schema. It does not contain a column for OWNER.

**ALL_QUEUE_TABLES: Queue Tables Queue Accessible to the Current User**

The ALL_QUEUE_TABLES view describes queue tables accessible to the current user.

**DBA_QUEUES: All Queues in Database**

The DBA_QUEUES view specifies operational characteristics for every queue in a database. Its columns are the same as those ALL_QUEUES.

**USER_QUEUES: Queues In User Schema**

The USER_QUEUES view is the same as DBA_QUEUES with the exception that it only shows queues in the user's schema.

**ALL_QUEUES: Queues for Which User Has Any Privilege**

The ALL_QUEUES view describes all queues on which the current user has enqueue or dequeue privileges. If the user has any Advanced Queuing system privileges, like MANAGE ANY QUEUE, ENQUEUE ANY QUEUE or DEQUEUE ANY QUEUE, this view describes all queues in the database.

**DBA_QUEUE_SCHEDULES: All Propagation Schedules**

The DBA_QUEUE_SCHEDULES view describes all the current schedules in the database for propagating messages.

**USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema**

The USER_QUEUE_SCHEDULES view is the same as DBA_QUEUE_SCHEDULES with the exception that it only shows queue schedules in the user's schema.

**QUEUE_PRIVILEGES: Queues for Which User Has Queue Privilege**

The QUEUE_PRIVILEGES view describes queues for which the user is the grantor, grantee, or owner. It also shows queues for which an enabled role on the queue is granted to PUBLIC.
The AQ$Queue_Table_Name view describes the queue table in which message data is stored. This view is automatically created with each queue table and should be used for querying the queue data. The dequeue history data (time, user identification and transaction identification) is only valid for single-consumer queues.

In a queue table that is created with the compatible parameter set to '8.1' or higher, messages that were not dequeued by the consumer are shown as "UNDELIVERABLE". You can dequeue these messages by msgid. If the Oracle® Database queue process monitor is running, then the messages are eventually moved to an exception queue. You can dequeue these messages from the exception queue with an ordinary dequeue.

A multiconsumer queue table created without the compatible parameter, or with the compatible parameter set to '8.0', does not display the state of a message on a consumer basis, but only displays the global state of the message.

**Note:** Queues created in a queue table with compatible set to 8.0 (referred to in this guide as 8.0-style queues) are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

When a message is dequeued using the REMOVE mode, DEQ_TIME, DEQ_USER_ID, and DEQ_TXN_ID are updated for the consumer that dequeued the message.

You can use MSGID and ORIGINAL_MSGID to chain propagated messages. When a message with message identifier m1 is propagated to a remote queue, m1 is stored in the ORIGINAL_MSGID column of the remote queue.

Beginning with Oracle Database 10g, AQ$Queue_Table_Name includes buffered messages. For buffered messages, the value of MSG_STATE is one of the following:

- **IN MEMORY**
  - Buffered messages enqueued by a user
- **DEFERRED**
  - Buffered messages enqueued by a capture process
- **SPILLED**
  - User-enqueued buffered messages that have been spilled to disk
- **DEFERRED SPILLED**
  - Capture-enqueued buffered messages that have been spilled to disk
- **BUFFERED EXPIRED**
  - Expired buffered messages

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td>VARCHAR2(30)</td>
<td>-</td>
<td>Queue name</td>
</tr>
<tr>
<td>MSG_ID</td>
<td>RAW(16)</td>
<td>NOT NULL</td>
<td>Unique identifier of the message</td>
</tr>
<tr>
<td>CORR_ID</td>
<td>VARCHAR2(128)</td>
<td>-</td>
<td>User-provided correlation identifier</td>
</tr>
</tbody>
</table>
### Table 9–1 (Cont.) AQ$Queue_Table_Name View

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_PRIORITY</td>
<td>NUMBER</td>
<td>-</td>
<td>Message priority</td>
</tr>
<tr>
<td>MSG_STATE</td>
<td>VARCHAR2 (16)</td>
<td>-</td>
<td>Message state</td>
</tr>
<tr>
<td>DELAY</td>
<td>DATE</td>
<td>-</td>
<td>Time in date format at which the message in waiting state would become ready. Equals ENQUEUE_TIME + user specified DELAY</td>
</tr>
<tr>
<td>DELAY_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>-</td>
<td>Time as a timestamp format at which the message in waiting state would become ready. Equals ENQUEUE_TIMESTAMP + user specified DELAY</td>
</tr>
<tr>
<td>EXPIRATION</td>
<td>NUMBER</td>
<td>-</td>
<td>Number of seconds in which the message expires after being READY</td>
</tr>
<tr>
<td>ENQ_TIME</td>
<td>DATE</td>
<td>-</td>
<td>Enqueue time</td>
</tr>
<tr>
<td>ENQ_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>-</td>
<td>Enqueue time</td>
</tr>
<tr>
<td>ENQ_USER_ID</td>
<td>NUMBER</td>
<td>-</td>
<td>Enqueue user ID</td>
</tr>
<tr>
<td>ENQ_USER_ID (10.1 queue tables)</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Enqueue user name</td>
</tr>
<tr>
<td>ENQ_TXN_ID</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Enqueue transaction ID</td>
</tr>
<tr>
<td>DEQ_TIME</td>
<td>DATE</td>
<td>-</td>
<td>Dequeue time</td>
</tr>
<tr>
<td>DEQ_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>-</td>
<td>Dequeue time</td>
</tr>
<tr>
<td>DEQ_USER_ID</td>
<td>NUMBER</td>
<td>-</td>
<td>Dequeue user ID</td>
</tr>
<tr>
<td>DEQ_USER_ID (10.1 queue tables)</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Dequeue user name</td>
</tr>
<tr>
<td>DEQ_TXN_ID</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Dequeue transaction ID</td>
</tr>
<tr>
<td>RETRY_COUNT</td>
<td>NUMBER</td>
<td>-</td>
<td>Number of retries</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE_OWNER</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Exception queue schema</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Exception queue name</td>
</tr>
<tr>
<td>USER_DATA</td>
<td>-</td>
<td>-</td>
<td>User data</td>
</tr>
<tr>
<td>SENDER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Name of the agent enqueuing the message (valid only for 8.1-compatible queue tables)</td>
</tr>
<tr>
<td>SENDER_ADDRESS</td>
<td>VARCHAR2 (1024)</td>
<td>-</td>
<td>Queue name and database name of the source (last propagating) queue (valid only for 8.1-compatible queue tables). The database name is not specified if the source queue is in the local database.</td>
</tr>
<tr>
<td>SENDER_PROTOCOL</td>
<td>NUMBER</td>
<td>-</td>
<td>Protocol for sender address (reserved for future use and valid only for 8.1-compatible queue tables)</td>
</tr>
<tr>
<td>ORIGINAL_MSGID</td>
<td>RAW (16)</td>
<td>-</td>
<td>Message ID of the message in the source queue (valid only for 8.1-compatible queue tables)</td>
</tr>
<tr>
<td>CONSUMER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Name of the agent receiving the message (valid only for 8.1-compatible multiconsumer queue tables)</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>VARCHAR2 (1024)</td>
<td>-</td>
<td>Queue name and database link name of the agent receiving the message. The database link name is not specified if the address is in the local database. The address is NULL if the receiving agent is local to the queue (valid only for 8.1-compatible multiconsumer queue tables)</td>
</tr>
</tbody>
</table>
The `AQ$Queue_Table_Name_S` view provides information about subscribers for all the queues in any given queue table. It shows subscribers created by users with `DBMS_AQADM.ADD_SUBSCRIBER` and subscribers created for the apply process to apply user-created events. It also displays the transformation for the subscriber, if it was created with one. It is generated when the queue table is created.

This view provides functionality that is equivalent to the `DBMS_AQADM.QUEUE_SUBSCRIBERS()` procedure. For these queues, Oracle recommends that the view be used instead of this procedure to view queue subscribers. This view is created only for 8.1-compatible queue tables.

### Table 9-2  `AQ$Queue_Table_Name_S` View

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td>VARCHAR2(30)</td>
<td>NOT NULL</td>
<td>Name of queue for which subscriber is defined</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2(30)</td>
<td>-</td>
<td>Name of agent</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>VARCHAR2(1024)</td>
<td>-</td>
<td>Address of agent</td>
</tr>
<tr>
<td>PROTOCOL</td>
<td>NUMBER</td>
<td>-</td>
<td>Protocol of agent</td>
</tr>
<tr>
<td>TRANSFORMATION</td>
<td>VARCHAR2(61)</td>
<td>-</td>
<td>Name of the transformation (can be null)</td>
</tr>
</tbody>
</table>
AQ$Queue_Table_Name_R: Queue Subscribers and Their Rules

The AQ$Queue_Table_Name_R view displays only the subscribers based on rules for all queues in a given queue table, including the text of the rule defined by each subscriber. It also displays the transformation for the subscriber, if one was specified. It is generated when the queue table is created.

This view is created only for 8.1-compatible queue tables.

<table>
<thead>
<tr>
<th>Table 9–3 AQ$Queue_Table_Name_R View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
</tr>
<tr>
<td>QUEUE</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>ADDRESS</td>
</tr>
<tr>
<td>PROTOCOL</td>
</tr>
<tr>
<td>RULE</td>
</tr>
<tr>
<td>RULE_SET</td>
</tr>
<tr>
<td>TRANSFORMATION</td>
</tr>
</tbody>
</table>

DBA_QUEUE_SUBSCRIBERS: All Queue Subscribers in Database

The DBA_QUEUE_SUBSCRIBERS view returns a list of all subscribers on all queues in the database. Its columns are the same as those in ALL_QUEUE_SUBSCRIBERS.

USER_QUEUE_SUBSCRIBERS: Queue Subscribers in User Schema

The USER_QUEUE_SUBSCRIBERS view returns a list of subscribers on queues in the schema of the current user. Its columns are the same as those in ALL_QUEUE_SUBSCRIBERS except that it does not contain the OWNER column.

ALL_QUEUE_SUBSCRIBERS: Subscribers for Queues Where User Has Queue Privileges

The ALL_QUEUE_SUBSCRIBERS view returns a list of subscribers to queues that the current user has privileges to dequeue from.

DBA_TRANSFORMATIONS: All Transformations

The DBA_TRANSFORMATIONS view displays all the transformations in the database. These transformations can be specified with Advanced Queue operations like enqueue, dequeue and subscribe to automatically integrate transformations in messaging. This view is accessible only to users having DBA privileges.

DBA_ATTRIBUTE_TRANSFORMATIONS: All Transformation Functions

The DBA_ATTRIBUTE_TRANSFORMATIONS view displays the transformation functions for all the transformations in the database.
**USER_TRANSFORMATIONS: User Transformations**

The `USER_TRANSFORMATIONS` view displays all the transformations owned by the user. To view the transformation definition, query `USER_ATTRIBUTE_TRANSFORMATIONS`.

**USER_ATTRIBUTE_TRANSFORMATIONS: User Transformation Functions**

The `USER_ATTRIBUTE_TRANSFORMATIONS` view displays the transformation functions for all the transformations of the user.

**DBA_SUBSCR_REGISTRATIONS: All Subscription Registrations**

The `DBA_SUBSCR_REGISTRATIONS` view lists all the subscription registrations in the database.

**USER_SUBSCR_REGISTRATIONS: User Subscription Registrations**

The `USER_SUBSCR_REGISTRATIONS` view lists the subscription registrations in the database for the current user. Its columns are the same as those in `DBA_SUBSCR_REGISTRATIONS`.

**AQ$INTERNET_USERS: Oracle® Database Agents Registered for Internet Access**

The `AQ$INTERNET_USERS` view provides information about the agents registered for Internet access to Oracle® Database. It also provides the list of database users that each Internet agent maps to.

<table>
<thead>
<tr>
<th>Table 9–4</th>
<th>AQ$INTERNET_USERS View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column</strong></td>
<td><strong>Datatype</strong></td>
</tr>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>DB_USERNAME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>HTTP_ENABLED</td>
<td>VARCHAR2 (4)</td>
</tr>
<tr>
<td>FTP_ENABLED</td>
<td>VARCHAR2 (4)</td>
</tr>
</tbody>
</table>

**(G)V$AQ: Number of Messages in Different States in Database**

The `(G)V$AQ` view provides information about the number of messages in different states for the whole database.

In a Real Application Clusters environment, each instance keeps its own Oracle® Database statistics information in its own System Global Area (SGA), and does not have knowledge of the statistics gathered by other instances. When a `GV$AQ` view is queried by an instance, all other instances funnel their Oracle® Database statistics information to the instance issuing the query.
(G)V$BUFFERED_QUEUES: All Buffered Queues in the Instance.

The V$BUFFERED_QUEUES view displays information about all buffered queues in the instance. There is one row per queue.

(G)V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance

The V$BUFFERED_SUBSCRIBERS view displays information about the subscribers for all buffered queues in the instance. There is one row per subscriber per queue.

(G)V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance

The V$BUFFERED_PUBLISHERS view displays information about all buffered publishers in the instance. There is one row per queue per sender. The values are reset to zero when the database (or instance in an Oracle RAC environment) restarts.

(G)V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance

The V$PERSISTENT_QUEUES view displays information about all active persistent queues in the database since the queues' first activity time. There is one row per queue. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.

(G)V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues

The V$PERSISTENT_QMN_CACHE view displays detailed statistics about all background activities relating to all queue tables in the database. There is one row per queue table. The values are reset when the database (or instance in an Oracle RAC environment) restarts.

(G)V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance

The V$PERSISTENT_SUBSCRIBERS view displays information about all active subscribers of the persistent queues in the database. There is one row per instance per queue per subscriber. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.

(G)V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance

The V$PERSISTENT_PUBLISHERS view displays information about all active publishers of the persistent queues in the database. There is one row per instance per queue per publisher. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.
(G)V$PROPAGATION_SENDER: Buffer Queue Propagation Schedules on the Sending (Source) Side

The V$PROPAGATION_SENDER view displays information about buffer queue propagation schedules on the sending (source) side. The values are reset to zero when the database (or instance in a Real Application Clusters (RAC) environment) restarts, when propagation migrates to another instance, or when an unscheduled propagation is attempted.

(G)V$PROPAGATION_RECEIVER: Buffer Queue Propagation Schedules on the Receiving (Destination) Side

The V$PROPAGATION_RECEIVER view displays information about buffer queue propagation schedules on the receiving (destination) side. The values are reset to zero when the database (or instance in a Real Application Clusters (RAC) environment) restarts, when propagation migrates to another instance, or when an unscheduled propagation is attempted.

(G)V$SUBSCR_REGISTRATION_STATS: Diagnosability of Notifications

The V$SUBSCR_REGISTRATION_STATS view provides information for diagnosability of notifications.

V$METRICGROUP: Information about the Metric Group

This V$METRICGROUP view displays information about the metric group for each of the four major Streams components: capture, propagation, apply, and queue.

(G)V$STREAMSMETRIC: Streams Metrics for the Most Recent Interval

This view displays the capture, propagation, and apply metrics for the most recent interval.

Table 9–5  GV$STREAMSMETRIC View

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST_ID</td>
<td>Instance</td>
<td>ID (GV$ only)</td>
</tr>
<tr>
<td>BEGIN_TIME</td>
<td>DATE</td>
<td>Begin time of interval</td>
</tr>
<tr>
<td>END_TIME</td>
<td>DATE</td>
<td>End time of interval</td>
</tr>
<tr>
<td>INTSIZE_CSEC</td>
<td>NUMBER</td>
<td>Interval size (centi-seconds)</td>
</tr>
<tr>
<td>COMPONENT_TYPE</td>
<td>VARCHAR2 (32)</td>
<td>Type of the component (either 'CAPTURE', 'PROPAGATION', or 'APPLY')</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td>VARCHAR2 (32)</td>
<td>Name of streams component</td>
</tr>
<tr>
<td>COMPONENT_START_TIME</td>
<td>DATE</td>
<td>Time that component started</td>
</tr>
<tr>
<td>RATE1_VALUE</td>
<td>NUMBER</td>
<td>Value of rate 1</td>
</tr>
<tr>
<td>RATE1_NAME</td>
<td>VARCHAR2 (64)</td>
<td>Name of rate 1</td>
</tr>
<tr>
<td>RATE1_UNIT</td>
<td>VARCHAR2 (64)</td>
<td>Unit of measurement of rate 1</td>
</tr>
<tr>
<td>RATE2_VALUE</td>
<td>NUMBER</td>
<td>Value of rate 2</td>
</tr>
</tbody>
</table>
(G)V$STREAMSMETRIC_HISTORY: Streams Metrics Over Past Hour
This view returns all metric values for streams messages over the past hour. It has the same form as (G)V$STREAMSMETRIC: Streams Metrics for the Most Recent Interval.

(G)V$QUEUEMETRIC: Queue Metrics for the Most Recent Interval
This view displays the queue metrics for the most recent interval.

(G)V$QUEUEMETRIC_HISTORY: Queue Metrics Over Past Hour
This view returns all queue metric values over the past hour. It has the same shape as (G)V$QUEUEMETRIC: Queue Metrics for the Most Recent Interval.

DBA_HIST_STREAMSMETRIC: Streams Metric History
This view displays view provides catalog access to streams metric history.
DBA_HIST_QUEUEMETRIC: Queue Metric History

This view displays view provides catalog access to queue metric history.

Table 9–7  (Cont.)  DBA_HIST_STREAMSMETRIC View

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_TIME</td>
<td>DATE</td>
<td>End time of interval</td>
</tr>
<tr>
<td>INTSIZE</td>
<td>NUMBER</td>
<td>Interval size (centi-seconds)</td>
</tr>
<tr>
<td>COMPONENT_TYPE</td>
<td>VARCHAR2(32)</td>
<td>Type of the component (either 'CAPTURE', 'PROPAGATION', or 'APPLY')</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td>VARCHAR2(32)</td>
<td>Name of streams component</td>
</tr>
<tr>
<td>COMPONENT_START_TIME</td>
<td>DATE</td>
<td>Time that component started</td>
</tr>
<tr>
<td>RATE1_VALUE</td>
<td>NUMBER</td>
<td>Value of rate 1</td>
</tr>
<tr>
<td>RATE2_VALUE</td>
<td>NUMBER</td>
<td>Value of rate 2</td>
</tr>
<tr>
<td>LATENCY</td>
<td>NUMBER</td>
<td>Latency from time last message processed by component was written to redo to time the message was processed by this component</td>
</tr>
</tbody>
</table>

DBA_HIST_QUEUEMETRIC: Queue Metric History

This view displays view provides catalog access to queue metric history.

Table 9–8  DBA_ATTRIBUTE_TRANSFORMATIONS View

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAP_ID</td>
<td>NUMBER</td>
<td>Required by AWR, snapshot ID</td>
</tr>
<tr>
<td>DBID</td>
<td>NUMBER</td>
<td>Required by AWR, database ID</td>
</tr>
<tr>
<td>INSTANCE_NUMBER</td>
<td>NUMBER</td>
<td>Required by AWR, instance number</td>
</tr>
<tr>
<td>QUEUE_NAME</td>
<td>VARCHAR2(32)</td>
<td>Name of queue process</td>
</tr>
<tr>
<td>QUEUE_START_TIME</td>
<td>DATE</td>
<td>Time when queue started</td>
</tr>
<tr>
<td>BEGIN_TIME</td>
<td>DATE</td>
<td>Begin time of interval</td>
</tr>
<tr>
<td>END_TIME</td>
<td>DATE</td>
<td>End time of interval</td>
</tr>
<tr>
<td>INTSIZE</td>
<td>NUMBER</td>
<td>Interval size (centi-seconds)</td>
</tr>
<tr>
<td>ENQUEUED_PER_SECOND</td>
<td>NUMBER</td>
<td>Messages enqueue per second</td>
</tr>
<tr>
<td>SPILLED_PER_SECOND</td>
<td>NUMBER</td>
<td>Messages spilled per second</td>
</tr>
<tr>
<td>NUMMESSAGES</td>
<td>NUMBER</td>
<td>Number of messages in the queue</td>
</tr>
</tbody>
</table>

MGW_GATEWAY: Configuration and Status Information

This view lists configuration and status information for Messaging Gateway.

Table 9–9  MGW_GATEWAY View Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_DATABASE</td>
<td>VARCHAR2</td>
<td>The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection is used.</td>
</tr>
<tr>
<td>AGENT_INSTANCE</td>
<td>NUMBER</td>
<td>The database instance on which the Messaging Gateway agent is currently running. This should be NULL if the agent is not running.</td>
</tr>
<tr>
<td>AGENT_JOB</td>
<td>NUMBER</td>
<td>[Deprecated] Job number of the queued job used to start the Messaging Gateway agent process. The job number is set when Messaging Gateway is started and cleared when it shuts down.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent</td>
</tr>
<tr>
<td>AGENT_PING</td>
<td>VARCHAR2</td>
<td>Gateway agent ping status. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ NULL means no ping attempt was made.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ REACHABLE means ping attempt was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ UNREACHABLE means ping attempt failed.</td>
</tr>
<tr>
<td>AGENT_PING</td>
<td></td>
<td>AGENT_PING attempts to contact the Messaging Gateway agent. There is a short delay (up to 5 seconds) if the ping attempt fails. No ping is attempted if the AGENT_STATUS is NOT_STARTED or START_SCHEDULED.</td>
</tr>
<tr>
<td>AGENT_START_TIME</td>
<td>TIMESTAMP</td>
<td>The time when the Messaging Gateway agent job currently running was started. This should be NULL if the agent is not running.</td>
</tr>
<tr>
<td>AGENT_STATUS</td>
<td>VARCHAR2</td>
<td>Status of the Messaging Gateway agent. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ NOT_STARTED means the Messaging Gateway agent has not been started</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ START_SCHEDULED means Messaging Gateway agent has been scheduled to start. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, but the queued job used to start the Messaging Gateway agent has not yet run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ STARTING means Messaging Gateway agent is starting. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, the queued job has run, and the Messaging Gateway agent is starting up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ INITIALIZING means the Messaging Gateway agent has started and is initializing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ RUNNING means the Messaging Gateway agent is running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ SHUTTING_DOWN means the Messaging Gateway agent is shutting down</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ BROKEN means an unexpected condition has been encountered that prevents the Messaging Gateway agent from starting. DBMS_MGWADM.CLEANUP_GATEWAY must be called before the agent can be started.</td>
</tr>
<tr>
<td>AGENT_USER</td>
<td>VARCHAR2</td>
<td>Database username used by the Messaging Gateway agent to connect the database</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>VARCHAR2</td>
<td>Comments for the agent</td>
</tr>
<tr>
<td>CONNTYPE</td>
<td>VARCHAR2</td>
<td>Connection type used by the agent:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ JDBC_OCI if the JDBC OCI driver is used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ JDBC_THIN if the JDBC Thin driver is used</td>
</tr>
<tr>
<td>INITFILE</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway initialization file used by the agent. NULL indicates that the default initialization file is used.</td>
</tr>
<tr>
<td>LAST_ERROR_DATE</td>
<td>DATE</td>
<td>Date of last Messaging Gateway agent error. The last error information is cleared when Messaging Gateway is started. It is set if the Messaging Gateway agent fails to start or terminates due to an abnormal condition.</td>
</tr>
<tr>
<td>LAST_ERROR_MSG</td>
<td>VARCHAR2</td>
<td>Message for last Messaging Gateway agent error</td>
</tr>
<tr>
<td>LAST_ERROR_TIME</td>
<td>VARCHAR2</td>
<td>Time of last Messaging Gateway agent error</td>
</tr>
<tr>
<td>MAX_CONNECTIONS</td>
<td>NUMBER</td>
<td>[Deprecated] Maximum number of messaging connections to Oracle Database</td>
</tr>
</tbody>
</table>
**MGW_AGENT_OPTIONS: Supplemental Options and Properties**

This view lists supplemental options and properties for a Messaging Gateway agent.

**Table 9–10  MGW_AGENT_OPTIONS View**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent</td>
</tr>
<tr>
<td>ENCRYPTED</td>
<td>VARCHAR2</td>
<td>Indicates whether the value is stored as encrypted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TRUE if the value is stored encrypted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FALSE if the value is stored as cleartext</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2</td>
<td>Name of the option</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR2</td>
<td>Option type or usage: JAVA_SYSTEM_PROP if the option is used to set a Java System property</td>
</tr>
<tr>
<td>VALUE</td>
<td>VARCHAR2</td>
<td>Value for the option. This will be &lt;&lt;ENCRYPTED&gt;&gt; if the value is stored in an encrypted form.</td>
</tr>
</tbody>
</table>

**MGW_LINKS: Names and Types of Messaging System Links**

This view lists the names and types of messaging system links currently defined.

**Table 9–11  MGW_LINKS View Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent that will process propagation jobs for this link</td>
</tr>
<tr>
<td>LINK_COMMENT</td>
<td>VARCHAR2</td>
<td>User comment for the link</td>
</tr>
<tr>
<td>LINK_NAME</td>
<td>VARCHAR2</td>
<td>Name of the messaging system link</td>
</tr>
<tr>
<td>LINK_TYPE</td>
<td>VARCHAR2</td>
<td>Type of messaging system link. Values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MQSERIES is for WebSphere MQ links.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TIBRV is for TIB/Rendezvous links.</td>
</tr>
</tbody>
</table>

**MGW_MQSERIES_LINKS: WebSphere MQ Messaging System Links**

This view lists information for the WebSphere MQ messaging system links. The view includes most of the messaging system properties specified when the link is created.
**Table 9–12  MGW_MQSERIES_LINKS View Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent that will process propagation jobs for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this link</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>VARCHAR2</td>
<td>Connection channel</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>VARCHAR2</td>
<td>Name of the WebSphere MQ host</td>
</tr>
<tr>
<td>INBOUND_LOG_QUEUE</td>
<td>VARCHAR2</td>
<td>Inbound propagation log queue</td>
</tr>
<tr>
<td>INTERFACE_TYPE</td>
<td>VARCHAR2</td>
<td>Messaging interface type. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ BASE_JAVA is for WebSphere MQ Base Java interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ JMS_CONNECTION is for WebSphere MQ JMS unified, domain-independent connections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ JMS_QUEUE_CONNECTION is for WebSphere MQ JMS queue connections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ JMS_TOPIC_CONNECTION is for WebSphere MQ JMS topic connections</td>
</tr>
<tr>
<td>LINK_COMMENT</td>
<td>VARCHAR2</td>
<td>User comment for the link</td>
</tr>
<tr>
<td>LINK_NAME</td>
<td>VARCHAR2</td>
<td>Name of the messaging system link</td>
</tr>
<tr>
<td>MAX_CONNECTIONS</td>
<td>NUMBER</td>
<td>Maximum number of messaging connections</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>SYS.MGW_</td>
<td>Link options</td>
</tr>
<tr>
<td></td>
<td>PROPERTIES</td>
<td></td>
</tr>
<tr>
<td>OUTBOUND_LOG_QUEUE</td>
<td>VARCHAR2</td>
<td>Outbound propagation log queue</td>
</tr>
<tr>
<td>PORT</td>
<td>NUMBER</td>
<td>Port number</td>
</tr>
<tr>
<td>QUEUE_MANAGER</td>
<td>VARCHAR2</td>
<td>Name of the WebSphere MQ queue manager</td>
</tr>
</tbody>
</table>

**MGW_TIBRV_LINKS: TIB/Rendezvous Messaging System Links**

This view lists information for TIB/Rendezvous messaging system links. The view includes most of the messaging system properties specified when the link was created.

**Table 9–13  MGW_TIBRV_LINKS View Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent that will process propagation jobs for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this link</td>
</tr>
<tr>
<td>CM_LEDGER</td>
<td>VARCHAR2</td>
<td>TIB/Rendezvous CM ledger file name</td>
</tr>
<tr>
<td>CM_NAME</td>
<td>VARCHAR2</td>
<td>TIB/Rendezvous CM correspondent name</td>
</tr>
<tr>
<td>DAEMON</td>
<td>VARCHAR2</td>
<td>TIB/Rendezvous daemon parameter for RVD transport</td>
</tr>
<tr>
<td>LINK_COMMENT</td>
<td>VARCHAR2</td>
<td>User comment for the link</td>
</tr>
<tr>
<td>LINK_NAME</td>
<td>VARCHAR2</td>
<td>Name of the messaging system link</td>
</tr>
<tr>
<td>NETWORK</td>
<td>VARCHAR2</td>
<td>TIB/Rendezvous network parameter for rvd transport</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>SYS.MGW_</td>
<td>Link options</td>
</tr>
<tr>
<td></td>
<td>PROPERTIES</td>
<td></td>
</tr>
<tr>
<td>SERVICE</td>
<td>VARCHAR2</td>
<td>TIB/Rendezvous service parameter for rvd transport</td>
</tr>
</tbody>
</table>
MGW_FOREIGN_QUEUES: Foreign Queues

This view lists information for foreign queues. The view includes most of the queue properties specified when the queue is registered.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN</td>
<td>VARCHAR2</td>
<td>Queue domain type. Values:</td>
</tr>
<tr>
<td>LINK_NAME</td>
<td>VARCHAR2</td>
<td>Name of the messaging system link</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2</td>
<td>Name of the registered queue</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>SYS.MGW_PROPERTIES</td>
<td>Optional queue properties</td>
</tr>
<tr>
<td>PROVIDER_QUEUE</td>
<td>VARCHAR2</td>
<td>Message provider (native) queue name</td>
</tr>
<tr>
<td>QUEUE_COMMENT</td>
<td>VARCHAR2</td>
<td>User comment for the foreign queue</td>
</tr>
</tbody>
</table>

MGW_JOBS: Messaging Gateway Propagation Jobs

This view lists information for Messaging Gateway propagation jobs. The view includes most of the job properties specified when the propagation job was created, as well as other status and statistical information.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway agent that processes this job</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>VARCHAR2</td>
<td>Comments for the propagation job</td>
</tr>
<tr>
<td>DESTINATION</td>
<td>VARCHAR2</td>
<td>Destination queue to which messages are propagated</td>
</tr>
<tr>
<td>ENABLED</td>
<td>VARCHAR2</td>
<td>Indicates whether the job is enabled or not:</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE</td>
<td>VARCHAR2</td>
<td>Exception queue used for propagation logging purposes</td>
</tr>
<tr>
<td>EXCEPTIONQ_MSGS</td>
<td>NUMBER</td>
<td>Option type or usage: JAVA_SYSTEM_PROP if the option is used to set a Java System property</td>
</tr>
<tr>
<td>FAILURES</td>
<td>NUMBER</td>
<td>Number of messages moved to exception queue since the last time the agent was started</td>
</tr>
<tr>
<td>JOB_NAME</td>
<td>VARCHAR2</td>
<td>Name of the propagation job</td>
</tr>
<tr>
<td>LAST_ERROR_MSG</td>
<td>VARCHAR2</td>
<td>Message for the last propagation error</td>
</tr>
<tr>
<td>LAST_ERROR_DATE</td>
<td>DATE</td>
<td>Date of the last propagation error</td>
</tr>
<tr>
<td>LAST_ERROR_TIME</td>
<td>VARCHAR2</td>
<td>Time of the last propagation error</td>
</tr>
<tr>
<td>LINK_NAME</td>
<td>VARCHAR2</td>
<td>Name of the Messaging Gateway link used by this job</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>SYS.MGW_PROPERTIES</td>
<td>Job options</td>
</tr>
</tbody>
</table>
This view lists configuration and status information for Messaging Gateway subscribers. The view includes most of the subscriber properties specified when the subscriber is added, as well as other status and statistical information.

### Table 9–16  MGW_SUBSCRIBERS View Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTINATION</td>
<td>VARCHAR2</td>
<td>Destination queue to which messages are propagated</td>
</tr>
<tr>
<td>EXCEPTIONQ_MSGS</td>
<td>NUMBER</td>
<td>Number of messages moved to the propagation exception queue since the last time the agent was started</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE</td>
<td>VARCHAR2</td>
<td>Exception queue used for logging purposes</td>
</tr>
<tr>
<td>FAILURES</td>
<td>NUMBER</td>
<td>Number of propagation failures</td>
</tr>
<tr>
<td>LAST_ERROR_DATE</td>
<td>DATE</td>
<td>Date of last propagation error</td>
</tr>
</tbody>
</table>

**MGW_SUBSCRIBERS: Information for Subscribers**
### MGW_SCHEDULES: Information about Schedules

This view lists configuration and status information for Messaging Gateway schedules. The view includes most of the schedule properties specified when the schedule is created, as well as other status information.

#### Table 9–17  MGW_SCHEDULES View Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTINATION</td>
<td>VARCHAR2</td>
<td>Propagation destination</td>
</tr>
<tr>
<td>LATENCY</td>
<td>NUMBER</td>
<td>Propagation window latency (in seconds)</td>
</tr>
<tr>
<td>NEXT_TIME</td>
<td>VARCHAR2</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>PROPAGATION_TYPE</td>
<td>VARCHAR2</td>
<td>Propagation type. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OUTBOUND is for Oracle Streams AQ to non-Oracle propagation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- INBOUND is for non-Oracle to Oracle Streams AQ propagation</td>
</tr>
<tr>
<td>PROPAGATION_WINDOW</td>
<td>NUMBER</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>SCHEDULE_DISABLED</td>
<td>VARCHAR2</td>
<td>Indicates whether the schedule is disabled. Y means the schedule is disabled.</td>
</tr>
<tr>
<td>SCHEDULE_ID</td>
<td>VARCHAR2</td>
<td>Propagation schedule identifier</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>SOURCE</td>
<td>VARCHAR2</td>
<td>Propagation source</td>
</tr>
<tr>
<td>START_DATE</td>
<td>DATE</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>START_TIME</td>
<td>VARCHAR2</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>
This chapter describes the Oracle Streams Advanced Queuing (AQ) PL/SQL operational interface.

This chapter contains these topics:

- Using Secure Queues
- Enqueuing Messages
- Enqueuing an Array of Messages
- Listening to One or More Queues
- Dequeuing Messages
- Dequeuing an Array of Messages
- Registering for Notification
- Posting for Subscriber Notification
- Adding an Agent to the LDAP Server
- Removing an Agent from the LDAP Server

See Also:

- Chapter 3, "Oracle® Database: Programmatic Interfaces" for a list of available functions in each programmatic interface
- "DBMS_AQ" in Oracle Database PL/SQL Packages and Types Reference for more information on the PL/SQL interface
- Oracle Objects for OLE Online Help > Contents tab > OO4O Automation Server > OBJECTS > OraAQ Object for more information on the Visual Basic (OO4O) interface
- Oracle Streams Advanced Queuing Java API Reference for more information on the Java interface
- "More OCI Relational Functions" and "OCI Programming Advanced Topics" in Oracle Call Interface Programmer’s Guide for more information on the Oracle Call Interface (OCI)

Using Secure Queues

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter. See "MESSAGE_PROPERTIES_T Type" in Oracle Database PL/SQL Packages and Types Reference for more information about `sender_id`. 
When you use secure queues, the following are required:

- You must have created a valid Oracle® Database agent using `DBMS_AQADM.CREATE_AQ_AGENT`.
- You must map `sender_id` to a database user with enqueue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this.

**See Also:**
- "Creating an Oracle® Database Agent" on page 8-29
- "Enabling Database Access" on page 8-29
- Oracle Streams Concepts and Administration for information about secure queues

### Enqueuing Messages

```sql
DBMS_AQ.ENQUEUE(
    queue_name          IN      VARCHAR2,
    enqueue_options     IN      enqueue_options_t,
    message_properties  IN      message_properties_t,
    payload             IN      'type_name',
    msgid               OUT     RAW);
```

This procedure adds a message to the specified queue.

It is not possible to update the message payload after a message has been enqueued. If you want to change the message payload, then you must dequeue the message and enqueue a new message.

To store a payload of type `RAW`, Oracle® Database creates a queue table with LOB column as the payload repository. The maximum size of the payload is determined by which programmatic interface you use to access Oracle® Database. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G.

If a message is enqueued to a multiconsumer queue with no recipient and the queue has no subscribers (or rule-based subscribers that match this message), then Oracle error ORA 24033 is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

If several messages are enqueued in the same second, then they all have the same enq_time. In this case the order in which messages are dequeued depends on step_no, a variable that is monotonically increasing for each message that has the same enq_time. There is no situation when both enq_time and step_no are the same for two messages enqueued in the same session.

### Enqueue Options

The `enqueue_options` parameter specifies the options available for the enqueue operation. It has the following attributes:

- `visibility`

  The `visibility` attribute specifies the transactional behavior of the enqueue request. `ON_COMMIT` (the default) makes the enqueue is part of the current transaction. `IMMEDIATE` makes the enqueue operation an autonomous transaction which commits at the end of the operation.
Do not use the \texttt{IMMEDIATE} option when you want to use \texttt{LOB} locators. \texttt{LOB} locators are valid only for the duration of the transaction. Your locator will not be valid, because the \texttt{immediate} option automatically commits the transaction.

You must set the \texttt{visibility} attribute to \texttt{IMMEDIATE} to use buffered messaging.

- **relative\_msgid**
  
  The \texttt{relative\_msgid} attribute specifies the message identifier of the message referenced in the sequence deviation operation. This parameter is ignored unless \texttt{sequence\_deviation} is specified with the \texttt{BEFORE} attribute.

- **sequence\_deviation**
  
  The \texttt{sequence\_deviation} attribute specifies when the message should be dequeued, relative to other messages already in the queue. \texttt{BEFORE} puts the message ahead of the message specified by \texttt{relative\_msgid}. \texttt{TOP} puts the message ahead of any other messages.

  Specifying \texttt{sequence\_deviation} for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

---

\textbf{Note:} The \texttt{sequence\_deviation} attribute has no effect in releases prior to Oracle\textsuperscript{®} Database 10g Release 1 (10.1) if \texttt{message\_grouping} is set to \texttt{TRANSACTIONAL}.

The sequence deviation feature is deprecated in Oracle\textsuperscript{®} Database 10g Release 2 (10.2).

---

- **transformation**
  
  The \texttt{transformation} attribute specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.

- **delivery\_mode**
  
  If the \texttt{delivery\_mode} attribute is the default \texttt{PERSISTENT}, then the message is enqueued as a persistent message. If it is set to \texttt{BUFFERED}, then the message is enqueued as an buffered message. Null values are not allowed.

---

\textbf{Message Properties}

The \texttt{message\_properties} parameter contains the information that Oracle\textsuperscript{®} Database uses to manage individual messages. It has the following attributes:

- **priority**
  
  The \texttt{priority} attribute specifies the priority of the message. It can be any number, including negative numbers. A smaller number indicates higher priority.

- **delay**
  
  The \texttt{delay} attribute specifies the number of seconds during which a message is in the \texttt{WAITING} state. After this number of seconds, the message is in the \texttt{READY} state and available for dequeuing. If you specify \texttt{NO\_DELAY}, then the message is available for immediate dequeuing. Dequeuing by \texttt{msgid} overrides the \texttt{delay} specification.
Enqueuing Messages

Note: Delay is not supported with buffered messaging.

- expiration
  The expiration attribute specifies the number of seconds during which the message is available for dequeuing, starting from when the message reaches the READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state. If you specify NEVER, then the message does not expire.

Note: Message delay and expiration are enforced by the queue monitor (QMN) background processes. You must start the QMN processes for the database if you intend to use the delay and expiration features of Oracle® Database.

- correlation
  The correlation attribute is an identifier supplied by the producer of the message at enqueue time.

- attempts
  The attempts attribute specifies the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.

- recipient_list
  The recipient_list parameter is valid only for queues that allow multiple consumers. The default recipients are the queue subscribers.

- exception_queue
  The exception_queue attribute specifies the name of the queue into which the message is moved if it cannot be processed successfully. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log.

- delivery_mode
  Any value for delivery_mode specified in message properties at enqueue time is ignored. The value specified in enqueue options is used to set the delivery mode of the message. If the delivery mode in enqueue options is left unspecified, then it defaults to persistent.

- enqueue_time
  The enqueue_time attribute specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.

Note: Because information about seasonal changes in the system clock (switching between standard time and daylight-saving time, for example) is stored with each queue table, seasonal changes are automatically reflected in enqueue_time. If the system clock is changed for some other reason, then you must restart the database for Oracle® Database to pick up the changed time.
Enqueuing Messages

- **state**
  
  The `state` attribute specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time.

- **sender_id**
  
  The `sender_id` attribute is an identifier of type `aq$_agent` specified at enqueue time by the message producer.

- **original_msgid**
  
  The `original_msgid` attribute is used by Oracle Streams AQ for propagating messages.

- **transaction_group**
  
  The `transaction_group` attribute specifies the transaction group for the message. This attribute is set only by `DBMS_AQ.DEQUEUE_ARRAY`. This attribute cannot be used to set the transaction group of a message through `DBMS_AQ.ENQUEUE` or `DBMS_AQ.ENQUEUE_ARRAY`.

- **user_property**
  
  The `user_property` attribute is optional. It is used to store additional information about the payload.

The examples in this chapter use the same users, message types, queue tables, and queues as do the examples in Chapter 8, "Oracle® Database Administrative Interface". If you have not already created these structures in your test environment, then you must run the following examples:

- Example 8–1, "Setting Up AQ Administrative Users" on page 8-5
- Example 8–2, "Setting Up AQ Administrative Example Types" on page 8-6
- Example 8–3, "Creating a Queue Table for Messages of Object Type" on page 8-6
- Example 8–5, "Creating a Queue Table for Messages of LOB Type" on page 8-6
- Example 8–7, "Creating a Queue Table for Grouped Messages" on page 8-7
- Example 8–8, "Creating Queue Tables for Prioritized Messages and Multiple Consumers" on page 8-7
- Example 8–23, "Creating a Queue for Messages of Object Type" on page 8-13
- Example 8–25, "Creating a Queue for Messages of LOB Type" on page 8-14
- Example 8–26, "Creating a Queue for Grouped Messages" on page 8-14
- Example 8–27, "Creating a Queue for Prioritized Messages" on page 8-14
- Example 8–28, "Creating a Queue for Prioritized Messages and Multiple Consumers" on page 8-14
- Example 8–36, "Creating a Transformation" on page 8-17

For Example 8–1, you must connect as a user with administrative privileges. For the other examples in the preceding list, you can connect as user `test_adm`. After you have created the queues, you must start them as shown in "Starting a Queue" on page 8-15. Except as noted otherwise, you can connect as ordinary queue user 'test' to run all examples appearing in this chapter.

**Example 10–1 Enqueuing a Message, Specifying Queue Name and Payload**

```sql
DECLARE
```

Oracle® Database Operations Using PL/SQL 10-5
Enqueuing Messages

Example 10–2  Enqueuing a Message, Specifying Priority

DECLARE
  enqueue_options       DBMS_AQ.enqueue_options_t;
  message_properties    DBMS_AQ.message_properties_t;
  message_handle        RAW(16);
  message               test.order_typ;
BEGIN
  message := test.order_typ(002, 'PRIORITY MESSAGE', 'priority 30');
  message_properties.priority := 30;
  DBMS_AQ.ENQUEUE(
    queue_name => 'test.priority_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
  COMMIT;
END;
/

Enqueuing a LOB Type Message

Example 10–3 creates procedure blobenqueue() using the test.lob_type message payload object type created in Example 8–1 on page 8-5. On enqueue, the LOB attribute is set to EMPTY_BLOB. After the enqueue completes, but before the transaction is committed, the LOB attribute is selected from the user_data column of the test.lob_qtab queue table. The LOB data is written to the queue using the LOB interfaces (which are available through both OCI and PL/SQL). The actual enqueue operation is shown in

On dequeue, the message payload will contain the LOB locator. You can use this LOB locator after the dequeue, but before the transaction is committed, to read the LOB data. This is shown in Example 10–14 on page 10-16.

Example 10–3  Creating an Enqueue Procedure for LOB Type Messages

CREATE OR REPLACE PROCEDURE blobenqueue(msgno IN NUMBER) AS
  enq_userdata          test.lob_typ;
  enq_msgid             RAW(16);
  enqueue_options       DBMS_AQ.enqueue_options_t;
  message_properties    DBMS_AQ.message_properties_t;
  lob_loc               BLOB;
  buffer                RAW(4096);
BEGIN
buffer       := HEXTORAW(RPAD('FF', 4096, 'FF'));
enq_userdata := test.lob_typ(msgno, 'Large Lob data', EMPTY_BLOB(), msgno);
DBMS_AQ.ENQUEUE(
    queue_name => 'test.lob_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => enq_userdata,
msgid => enqmsgid);
SELECT t.user_data.data INTO lob_loc
FROM lob_qtab t
WHERE tmsgid = enqmsgid;
DBMS_LOB.WRITE(lob_loc, 2000, 1, buffer);
COMMIT;
END;
/

**Example 10–4  Enqueuing a LOB Type Message**

```
BEGIN
    FOR i IN 1..5 LOOP
        blobenqueue(i);
    END LOOP;
END;
/
```

**Example 10–5  Enqueuing Multiple Messages to a Single-Consumer Queue**

*Example 10–5* enqueues six messages to *test.obj_queue*. These messages are dequeued in *Example 10–17* on page 10-18.

```
SET SERVEROUTPUT ON
DECLARE
    enqueue_options     DBMS_AQ.enqueue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             test.message_typ;
BEGIN
    message := test.message_typ(001, 'ORANGE', 'ORANGE enqueued first.');
    DBMS_AQ.ENQUEUE(
        queue_name => 'test.obj_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => message_handle);
    message := test.message_typ(001, 'ORANGE', 'ORANGE also enqueued second. ');
    DBMS_AQ.ENQUEUE(
        queue_name => 'test.obj_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => message_handle);
    message := test.message_typ(001, 'YELLOW', 'YELLOW enqueued third.' );
    DBMS_AQ.ENQUEUE(
        queue_name => 'test.obj_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => message_handle);
```
message := test.message_typ(001, 'VIOLET', 'VIOLET enqueued fourth.');
DBMS_AQ.ENQUEUE(
    queue_name => 'test.obj_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
message := test.message_typ(001, 'PURPLE', 'PURPLE enqueued fifth.');
DBMS_AQ.ENQUEUE(
    queue_name => 'test.obj_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
message := test.message_typ(001, 'PINK', 'PINK enqueued sixth.');
DBMS_AQ.ENQUEUE(
    queue_name => 'test.obj_queue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => message_handle);
COMMIT;
END;
/

Enqueuing Multiple Messages to a Multiconsumer Queue

Example 10–6 requires that you connect as user 'test_adm' to add subscribers RED and GREEN to queue test.multiconsumer_queue. The subscribers are required for Example 10–7.

Example 10–6 Adding Subscribers RED and GREEN

DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('RED', NULL, NULL);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'test.multiconsumer_queue',
        subscriber => subscriber);
    subscriber := sys.aq$_agent('GREEN', NULL, NULL);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'test.multiconsumer_queue',
        subscriber => subscriber);
END;
/

Example 10–7 enqueues multiple messages from sender 001. MESSAGE 1 is intended for all queue subscribers. MESSAGE 2 is intended for RED and BLUE. These messages are dequeued in Example 10–17 on page 10-18.

Example 10–7 Enqueuing Multiple Messages to a Multiconsumer Queue

DECLARE
    enqueue_options DBMS_AQ.enqueue_options_t;
    message_properties DBMS_AQ.message_properties_t;
    recipients     DBMS_AQ.aq$_recipient_list_t;
    message_handle RAW(16);
message             test.message_typ;
BEGIN
message := test.message_typ(001, 'MESSAGE 1', 'For queue subscribers');
DBMS_AQ.ENQUEUE(
  queue_name => 'test.multiconsumer_queue',
  enqueue_options => enqueue_options,
  message_properties => message_properties,
  payload => message,
  msgid => message_handle);
message := test.message_typ(001, 'MESSAGE 2', 'For two recipients');
recipients(1) := sys.aq$_agent('RED', NULL, NULL);
recipients(2) := sys.aq$_agent('BLUE', NULL, NULL);
message_properties.recipient_list := recipients;
DBMS_AQ.ENQUEUE(
  queue_name => 'test.multiconsumer_queue',
  enqueue_options => enqueue_options,
  message_properties => message_properties,
  payload => message,
  msgid => message_handle);
COMMIT;
END;
/

Enqueuing Grouped Messages

Example 10–8 enqueues three groups of messages, with three messages in each group. These messages are dequeued in Example 10–16 on page 10-17.

Example 10–8 Enqueuing Grouped Messages

DECLARE
  enqueue_options     DBMS_AQ.enqueue_options_t;
  message_properties  DBMS_AQ.message_properties_t;
  message_handle      RAW(16);
  message             test.message_typ;
BEGIN
  FOR groupno in 1..3 LOOP
    FOR msgno in 1..3 LOOP
      message := test.message_typ(001, 'GROUP ' || groupno, 'Message ' || msgno || ' in group ' || groupno);
      DBMS_AQ.ENQUEUE(
        queue_name => 'test.group_queue',
        enqueue_options => enqueue_options,
        message_properties => message_properties,
        payload => message,
        msgid => message_handle);
    END LOOP;
  END LOOP;
COMMIT;
END;
/

Enqueuing a Message with Delay and Expiration

In Example 10–9, an application wants a message to be dequeued no earlier than a week from now, but no later than three weeks from now. Because expiration is
calculated from the earliest dequeue time, this requires setting the expiration time for two weeks.

**Example 10–9  **Enqueuing a Message, Specifying Delay and Expiration

```sql
DECLARE
    enqueue_options     DBMS_AQ.enqueue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             test.message_typ;
BEGIN
    message := test.message_typ(001, 'DELAYED', 'Message is delayed one week.');
    message_properties.delay      := 7*24*60*60;
    message_properties.expiration := 2*7*24*60*60;
    DBMS_AQ.ENQUEUE(
        queue_name           => 'test.obj_queue',
        enqueue_options      => enqueue_options,
        message_properties   => message_properties,
        payload              => message,
        msgid                => message_handle);
    COMMIT;
END;
/
```

**Example 10–10  **Enqueuing a Message, Specifying a Transformation

```sql
DECLARE
    enqueue_options     DBMS_AQ.enqueue_options_t;
    message_properties  DBMS_AQ.message_properties_t;
    message_handle      RAW(16);
    message             test.message_typ;
BEGIN
    message := test.message_typ(001, 'NORMAL MESSAGE', 'enqueued to obj_queue');
    enqueue_options.transformation := 'message_order_transform';
    DBMS_AQ.ENQUEUE(
        queue_name           => 'test.priority_queue',
        enqueue_options      => enqueue_options,
        message_properties   => message_properties,
        payload              => message,
        msgid                => message_handle);
    COMMIT;
END;
/
```

**See Also:**  "Using Advanced Queuing Interfaces" in *Oracle Objects for OLE Developer’s Guide* for OO4O message-enqueuing examples

---

**Enqueuing an Array of Messages**

```sql
DBMS_AQ.ENQUEUE_ARRAY(
    queue_name   IN   VARCHAR2,  
    enqueue_options   IN   enqueue_options_t,  
    array_size IN   PLS_INTEGER,  
    message_properties_array IN   message_properties_array_t,  
    payload_array   IN   VARRAY,  
    msid_array OUT   msgid_array_t)
RETURN PLS_INTEGER;
```
Use the `ENQUEUE_ARRAY` function to enqueue an array of payloads using a corresponding array of message properties. The output is an array of message identifiers of the enqueued messages. The function returns the number of messages successfully enqueued.

Array enqueuing is not supported for buffered messages, but you can still use `DBMS_AQ.ENQUEUE_ARRAY()` to enqueue buffered messages by setting `array_size` to 1.

See Also: "Enqueue Options" on page 10-2

The `message_properties_array` parameter is an array of message properties. Each element in the payload array must have a corresponding element in this record. All messages in an array have the same delivery mode.

See Also: "Message Properties" on page 10-3

The payload structure can be a VARRAY or nested table. The message IDs are returned into an array of `RAW(16)` entries of type `DBMS_AQ.msgid_array_t`.

As with array operations in the relational world, it is not possible to provide a single optimum array size that will be correct in all circumstances. Application developers must experiment with different array sizes to determine the optimal value for their particular applications.

Example 10–11 Enqueuing an Array of Messages

```sql
DECLARE
    enqueue_options       DBMS_AQ.enqueue_options_t;
    msg_prop_array        DBMS_AQ.message_properties_array_t;
    msg_prop              DBMS_AQ.message_properties_t;
    payload_array         test.msg_table;
    msgid_array           DBMS_AQ.msgid_array_t;
    retval                PLS_INTEGER;
BEGIN
    payload_array  := msg_table(
        message_typ(001, 'MESSAGE  1', 'array enqueued to obj_queue'),
        message_typ(001, 'MESSAGE  2', 'array enqueued to obj_queue'));
    msg_prop_array := DBMS_AQ.message_properties_array_t(msg_prop, msg_prop);
    retval := DBMS_AQ.ENQUEUE_ARRAY(
        queue_name               => 'test.obj_queue',
        enqueue_options          => enqueue_options,
        array_size               => 2,
        message_properties_array => msg_prop_array,
        payload_array            => payload_array,
        msgid_array              => msgid_array);
    COMMIT;
END;
/
```

Listening to One or More Queues

```sql
DBMS_AQ.LISTEN(
    agent_list             IN    aq$_agent_list_t,
    wait                   IN    BINARY_INTEGER DEFAULT FOREVER,
    listen_delivery_mode   IN    PLS_INTEGER DEFAULT PERSISTENT,
    agent                  OUT   sys.aq$_agent
    message_delivery_mode  OUT   PLS_INTEGER);
```

Oracle® Database Operations Using PL/SQL 10-11
TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;

This procedure specifies which queue or queues to monitor.
This call takes a list of agents as an argument. Each agent is identified by a unique combination of name, address, and protocol.

See Also: "AQ Agent Type" on page 2-2

You specify the queue to be monitored in the address field of each agent listed. Agents must have dequeue privileges on each monitored queue. You must specify the name of the agent when monitoring multiconsumer queues; but you must not specify an agent name for single-consumer queues. Only local queues are supported as addresses. Protocol is reserved for future use.

Note: Listening to multiconsumer queues is not supported in the Java API.

The listen_delivery_mode parameter specifies what types of message interest the agent. If it is the default PERSISTENT, then the agent is informed about persistent messages only. If it is set to BUFFERED, then the agent is informed about buffered messages only. If it is set to PERSISTENT_OR_BUFFERED, then the agent is informed about both types.

This is a blocking call that returns the agent and message type when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, then only the first agent listed is returned. If there are no messages found when the wait time expires, then an error is raised.

A successful return from the listen call is only an indication that there is a message for one of the listed agents in one of the specified queues. The interested agent must still dequeue the relevant message.

Note: You cannot call LISTEN on nonpersistent queues.

Example 10–12 Listening to a Single-Consumer Queue with Zero Timeout

SET SERVEROUTPUT ON
DECLARE
    agent            sys.aq$_agent;
    test_agent_list  DBMS_AQ.aq$_agent_list_t;
BEGIN
    test_agent_list(1) := sys.aq$_agent(NULL, 'test.obj_queue', NULL);
    test_agent_list(2) := sys.aq$_agent(NULL, 'test.priority_queue', NULL);
    DBMS_AQ.LISTEN(
        agent_list   =>   test_agent_list,
        wait         =>   0,
        agent        =>   agent);
    DBMS_OUTPUT.PUT_LINE('Message in Queue: ' || agent.address);
END;
/

Even though both test.obj_queue and test.priority_queue contain messages (enqueued in Example 10–1 and Example 10–2 respectively) Example 10–12 returns only:

Message in Queue: "TEST"."OBJ_QUEUE"
If the order of agents in `test_agent_list` is reversed, so `test.priority_queue` appears before `test.obj_queue`, then the example returns:

Message in Queue: "TEST"."PRIORITY_QUEUE"

Dequeueing Messages

```sql
DBMS_AQ.DEQUEUE(
    queue_name IN VARCHAR2,
    dequeue_options IN dequeue_options_t,
    message_properties OUT message_properties_t,
    payload OUT "type_name",
    msgid OUT RAW);
```

This procedure dequeues a message from the specified queue. Beginning with Oracle® Database 10g Release 2 (10.2), you can choose to dequeue only persistent messages, only buffered messages, or both. See `delivery_mode` in the following list of dequeue options.

**See Also:** "Message Properties" on page 10-3

**Dequeue Options**

The `dequeue_options` parameter specifies the options available for the dequeue operation. It has the following attributes:

- **consumer_name**

  A consumer can dequeue a message from a queue by supplying the name that was used in the `AQ$_AGENT` type of the `DBMS_AQADM.ADD_SUBSCRIBER` procedure or the recipient list of the message properties. If a value is specified, then only those messages matching `consumer_name` are accessed. If a queue is not set up for multiple consumers, then this field must be set to NULL (the default).

- **dequeue_mode**

  The `dequeue_mode` attribute specifies the locking behavior associated with the dequeue. If `BROWSE` is specified, then the message is dequeued without acquiring any lock. If `LOCKED` is specified, then the message is dequeued with a write lock that lasts for the duration of the transaction. If `REMOVE` is specified, then the message is dequeued and deleted (the default). The message can be retained in the queue table based on the retention properties. If `REMOVE_NO_DATA` is specified, then the message is marked as updated or deleted.

- **navigation**

  The `navigation` attribute specifies the position of the dequeued message. If `FIRST_MESSAGE` is specified, then the first available message matching the search criteria is dequeued. If `NEXT_MESSAGE` is specified, then the next available message matching the search criteria is dequeued (the default). If the previous message belongs to a message group, then the next available message matching the search criteria in the message group is dequeued.

  If `NEXT_TRANSACTION` is specified, then any messages in the current transaction group are skipped and the first message of the next transaction group is dequeued. This setting can only be used if message grouping is enabled for the queue.

- **visibility**
Dequeuing Messages

The visibility attribute specifies when the new message is dequeued. If ON_COMMIT is specified, then the dequeue is part of the current transaction (the default). If IMMEDIATE is specified, then the dequeue operation is an autonomous transaction that commits at the end of the operation. The visibility attribute is ignored in BROWSE dequeue mode.

Visibility must always be IMMEDIATE when dequeuing messages with delivery mode DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT_OR_BUFFERED.

- wait
  The wait attribute specifies the wait time if there is currently no message available matching the search criteria. If a number is specified, then the operation waits that number of seconds. If FOREVER is specified, then the operation waits forever (the default). If NO_WAIT is specified, then the operation does not wait.

-msgid
  The msgid attribute specifies the message identifier of the dequeued message. Only messages in the READY state are dequeued unless msgid is specified.

- correlation
  The correlation attribute specifies the correlation identifier of the dequeued message. The correlation identifier cannot be changed between successive dequeue calls without specifying the FIRST_MESSAGE navigation option.

  Correlation identifiers are application-defined identifiers that are not interpreted by Oracle® Database. You can use special pattern matching characters, such as the percent sign and the underscore. If more than one message satisfies the pattern, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

  **Note:** Although dequeue options correlation and deq_condition are both supported for buffered messages, it is not possible to create indexes to optimize these queries.

- deq_condition
  The deq_condition attribute is a Boolean expression similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions.

  To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with `tab.user_data` as a qualifier to indicate the specific column of the queue table that stores the payload.

  The deq_condition attribute cannot exceed 4000 characters. If more than one message satisfies the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

- transformation
  The transformation attribute specifies a transformation that will be applied after the message is dequeued but before returning the message to the caller.

- delivery_mode
The `delivery_mode` attribute specifies what types of messages to dequeue. If it is set to `DBMS_AQ.PERSISTENT`, then only persistent messages are dequeued. If it is set to `DBMS_AQ.BUFFERED`, then only buffered messages are dequeued.

If it is the default `DBMS_AQ.PERSISTENT_OR_BUFFERED`, then both persistent and buffered messages are dequeued. The `delivery_mode` attribute in the message properties of the dequeued message indicates whether the dequeued message was buffered or persistent.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the message identifier and correlation identifier in dequeue options.

The database consistent read mechanism is applicable for queue operations. For example, a `BROWSE` call may not see a message that is enqueued after the beginning of the browsing transaction.

In a *commit-time queue*, a new feature of Oracle® Database 10g Release 2 (10.2), messages are not visible to `BROWSE` or `DEQUEUE` calls until a deterministic order can be established among them based on an approximate CSCN.

### See Also:

- "Commit-Time Queues" in *Oracle Streams Concepts and Administration*
- "Dequeue Modes" on page 1-22

If the `navigation` attribute of the `dequeue_conditions` parameter is `NEXT_MESSAGE` (the default), then subsequent dequeues retrieve messages from the queue based on the snapshot obtained in the first dequeue. A message enqueued after the first dequeue command, therefore, will be processed only after processing all remaining messages in the queue. This is not a problem if all the messages have already been enqueued or if the queue does not have priority-based ordering. But if an application must process the highest-priority message in the queue, then it must use the `FIRST_MESSAGE` navigation option.

### Note:

It can also be more efficient to use the `FIRST_MESSAGE` navigation option when there are messages being concurrently enqueued. If the `FIRST_MESSAGE` option is not specified, then Oracle® Database continually generates the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST_MESSAGE` option is specified, then Oracle® Database uses a new snapshot for every dequeue command.

Messages enqueued in the same transaction into a queue that has been enabled for message grouping form a group. If only one message is enqueued in the transaction, then this effectively forms a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group locks the entire group. This is useful when all the messages in a group must be processed as a unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use `NEXT_TRANSACTION` to start dequeuing messages from the next available...
Dequeuing Messages

In the event that no groups are available, the dequeue times out after the period specified in the wait attribute of dequeue_options.

Typically, you expect the consumer of messages to access messages using the dequeue interface. You can view processed messages or messages still to be processed by browsing by message ID or by using SELECT commands.

Example 10–13 returns the message enqueued in Example 10–1 on page 10-5. It returns:

From Sender No.1
Subject: TEST MESSAGE
Text: First message to obj_queue

Example 10–13  Dequeuing Object Type Messages

```
SET SERVEROUTPUT ON
DECLARE
dequeue_options     DBMS_AQ.dequeue_options_t;
message_properties  DBMS_AQ.message_properties_t;
message_handle      RAW(16);
message             test.message_typ;
BEGIN
  dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
  DBMS_AQ.DEQUEUE(
    queue_name        => 'test.obj_queue',
    dequeue_options   => dequeue_options,
    message_properties => message_properties,
    payload           => message,
    msgid             => message_handle);
  DBMS_OUTPUT.PUT_LINE('From Sender No.'|| message.sender_id);
  DBMS_OUTPUT.PUT_LINE('Subject: '||message.subject);
  DBMS_OUTPUT.PUT_LINE('Text: '||message.text);
  COMMIT;
END;
/
```

Dequeuing LOB Type Messages

Example 10–14 creates procedure blobdequeue() to dequeue the LOB type messages enqueued in Example 10–4 on page 10-7. The actual dequeue is shown in Example 10–15. It returns:

Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000

Example 10–14  Creating a Dequeue Procedure for LOB Type Messages

```
CREATE OR REPLACE PROCEDURE blobdequeue(msgno IN NUMBER) AS
dequeue_options     DBMS_AQ.dequeue_options_t;
message_properties  DBMS_AQ.message_properties_t;
msgid               RAW(16);
payload             test.lob_typ;
lob_loc             BLOB;
amount              BINARY_INTEGER;
buffer              RAW(4096);
BEGIN
  DBMS_AQ.DEQUEUE(
    queue_name        => 'test.lob_queue',
    dequeue_options   => dequeue_options,
    message_properties => message_properties,
    payload           => payload,
    msgid             => msgid,
    lob_loc           => lob_loc,
    amount            => amount,
    buffer            => buffer);
```
Example 10–15  Dequeuing LOB Type Messages

BEGIN
  FOR i IN 1..5 LOOP
    blobdequeue(i);
  END LOOP;
END;
/

Dequeuing Grouped Messages

You can dequeue the grouped messages enqueued in Example 10–8 on page 10-9 by
running Example 10–16. It returns:

GROUP 1: Message 1 in group 1
GROUP 1: Message 2 in group 1
GROUP 1: Message 3 in group 1
Finished GROUP 1
GROUP 2: Message 1 in group 2
GROUP 2: Message 2 in group 2
GROUP 2: Message 3 in group 2
Finished GROUP 2
GROUP 3: Message 1 in group 3
GROUP 3: Message 2 in group 3
GROUP 3: Message 3 in group 3
Finished GROUP 3
No more messages

Example 10–16  Dequeuing Grouped Messages

SET SERVEROUTPUT ON
DECLARE
  dequeue_options DBMS_AQ.dequeue_options_t;
  message_properties DBMS_AQ.message_properties_t;
  message_handle RAW(16);
  message test.message_typ;
  no_messages exception;
  end_of_group exception;
  PRAGMA EXCEPTION_INIT (no_messages, -25228);
  PRAGMA EXCEPTION_INIT (end_of_group, -25235);
BEGIN
  dequeue_options.wait := DBMS_AQ.NO_WAIT;
  dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
  LOOP
    BEGIN
      DBMS_AQ.DEQUEUE{
        queue_name => 'test.group_queue',
        dequeue_options => dequeue_options,
        message_properties => message_properties,
        payload => payload,
        msgid => msgid);
        lob_loc := payload.data;
        amount := 2000;
        DBMS_LOB.READ(lob_loc, amount, 1, buffer);
        DBMS_OUTPUT.PUT_LINE('Amount of data read: '| amount);
        COMMIT;
      END;
    END LOOP;
END;
Debe que los mensajes reenviados para RED en el Ejemplo 10-7 en la página 10-8 se pueden desencadenar ejecutando el Ejemplo 10-17. Si cambia RED a GREEN y luego a BLUE, se puede usarlo para desencadenar sus mensajes también. El resultado del ejemplo será diferente en cada caso.

RED es un suscriptor de la cola de consumidores múltiples y también un destinatario especificado de MESSAGE 2, por lo que obtiene ambos mensajes:

Mensaje: MESSAGE 1 .. Para suscriptores de cola
Mensaje: MESSAGE 2 .. Para dos destinatarios
No más mensajes para RED

GREEN es solo un suscriptor, por lo que obtiene solo aquellos mensajes en la cola para los que no se han especificado destinatarios (en este caso, MESSAGE 1):

Mensaje: MESSAGE 1 .. Para suscriptores de cola
No más mensajes para GREEN

BLUE, aunque no un suscriptor de la cola, se especifica como destinatario de MESSAGE 2.

Mensaje: MESSAGE 2 .. Para dos destinatarios
No más mensajes para BLUE

**Ejemplo 10-17 Desencadenar mensajes para RED de una cola de consumidores múltiples**

```sql
SET SERVEROUTPUT ON
DECLARE
    dequeue_options       DBMS_AQ.dequeue_options_t;
    message_properties    DBMS_AQ.message_properties_t;
    message_handle        RAW(16);
    message               test.message_typ;
    no_messages           exception;
    PRAGMA EXCEPTION_INIT (no_messages, -25228);
BEGIN
    dequeue_options.wait          := DBMS_AQ.NO_WAIT;
    dequeue_options.consumer_name := 'RED';
    dequeue_options.navigation    := DBMS_AQ.FIRSTMESSAGE;
    LOOP
        BEGIN
            DBMS_AQ.DEQUEUE(
                queue_name              => 'test.multiconsumer_queue',
                message_properties      => message_properties,
                payload                  => message,
                msgid                    => message_handle);
            DBMS_OUTPUT.PUT_LINE(message.subject || ': ' || message.text);
            dequeue_options.navigation := DBMS_AQ.NEXTMESSAGE;
            EXCEPTION
                WHEN end_of_group THEN
                    DBMS_OUTPUT.PUT_LINE('Finished ' || message.subject);
                    COMMIT;
                    dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
            END LOOP;
            EXCEPTION
                WHEN no_messages THEN
                    DBMS_OUTPUT.PUT_LINE('No more messages');
            END;
        END;
    /
```

**Desencadenamiento de mensajes para RED de una cola de consumidores múltiples**

Puede desencadenar los mensajes encolados para RED en el Ejemplo 10-7 en la página 10-8 ejecutando el Ejemplo 10-17. Si cambia RED a GREEN y luego a BLUE, puede usarlo para desencadenar sus mensajes también. El resultado del ejemplo será diferente en cada caso.

RED es un suscriptor de la cola de consumidores múltiples y también un destinatario especificado de MESSAGE 2, por lo que obtiene ambos mensajes:

Mensaje: MESSAGE 1 .. Para suscriptores de cola
Mensaje: MESSAGE 2 .. Para destinatarios especificados
No más mensajes para RED

GREEN sólo es un suscriptor, por lo que obtiene solo aquellos mensajes en la cola para los que no se han especificado destinatarios (en este caso, MESSAGE 1):

Mensaje: MESSAGE 1 .. Para suscriptores de cola
No más mensajes para GREEN

BLUE, que aunque no es un suscriptor de la cola, se especifica como destinatario de MESSAGE 2.

Mensaje: MESSAGE 2 .. Para destinatarios especificados
No más mensajes para BLUE

---

**Ejemplo 10-17 Desencadenar mensajes para RED de una cola de consumidores múltiples**

```
SET SERVEROUTPUT ON
DECLARE
    dequeue_options       DBMS_AQ.dequeue_options_t;
    message_properties    DBMS_AQ.message_properties_t;
    message_handle        RAW(16);
    message               test.message_typ;
    no_messages           exception;
    PRAGMA EXCEPTION_INIT (no_messages, -25228);
BEGIN
    dequeue_options.wait          := DBMS_AQ.NO_WAIT;
    dequeue_options.consumer_name := 'RED';
    dequeue_options.navigation    := DBMS_AQ.FIRSTMESSAGE;
    LOOP
        BEGIN
            DBMS_AQ.DEQUEUE(
                queue_name              => 'test.multiconsumer_queue',
                message_properties      => message_properties,
                payload                  => message,
                msgid                    => message_handle);
            DBMS_OUTPUT.PUT_LINE(message.subject || ': ' || message.text);
            dequeue_options.navigation := DBMS_AQ.NEXTMESSAGE;
            EXCEPTION
                WHEN end_of_group THEN
                    DBMS_OUTPUT.PUT_LINE('Finished ' || message.subject);
                    COMMIT;
                    dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
            END LOOP;
            EXCEPTION
                WHEN no_messages THEN
                    DBMS_OUTPUT.PUT_LINE('No more messages');
            END;
        END;
    /
```
deque_options    => dequeue_options,
message_properties => message_properties,
payload            => message,
msgid              => message_handle);
DBMS_OUTPUT.PUT_LINE('Message: '|| message.subject ||' .. '|| message.text );
dequeue_options.navigation := DBMS_AQ.NEXT_MESSAGE;
END;
END LOOP;
EXCEPTION
  WHEN no_messages THEN
    DBMS_OUTPUT.PUT_LINE ('No more messages for RED');
    COMMIT;
END;
/

Example 10–18 browses messages enqueued in Example 10–5 until it finds PINK, which it removes. The example returns:

Browsed Message Text: ORANGE enqueued first.
Browsed Message Text: ORANGE also enqueued second.
Browsed Message Text: YELLOW enqueued third.
Browsed Message Text: VIOLET enqueued fourth.
Browsed Message Text: PURPLE enqueued fifth.
Browsed Message Text: PINK enqueued sixth.
Removed Message Text: PINK enqueued sixth.

Dequeu e Modes

Example 10–18 Dequeue in Browse Mode and Remove Specified Message

SET SERVEROUTPUT ON
DECLARE
  dequeue_options     DBMS_AQ.dequeue_options_t;
  message_properties  DBMS_AQ.message_properties_t;
  message_handle      RAW(16);
  message             test.message_typ;
BEGIN
  dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;
  LOOP
    DBMS_AQ.DEQUEUE(
      queue_name => 'test.obj_queue',
      dequeue_options => dequeue_options,
      message_properties => message_properties,
      payload       => message,
      msgid         => message_handle);
    DBMS_OUTPUT.PUT_LINE ('Browsed Message Text: ' || message.text);
    EXIT WHEN message.subject = 'PINK';
  END LOOP;
  dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
  dequeue_options.msgid        := message_handle;
  DBMS_AQ.DEQUEUE(
    queue_name => 'test.obj_queue',
    dequeue_options => dequeue_options,
    message_properties => message_properties,
    payload       => message,
    msgid         => message_handle);
  DBMS_OUTPUT.PUT_LINE('Removed Message Text: ' || message.text);
  COMMIT;
END;
/
Example 10–19 previews in locked mode the messages enqueued in Example 10–5 until it finds PURPLE, which it removes. The example returns:

Locked Message Text: ORANGE enqueued first.
Locked Message Text: ORANGE also enqueued second.
Locked Message Text: YELLOW enqueued third.
Locked Message Text: VIOLET enqueued fourth.
Locked Message Text: PURPLE enqueued fifth.
Removed Message Text: PURPLE enqueued fifth.

Example 10–19  Dequeue in Locked Mode and Remove Specified Message

SET SERVEROUTPUT ON
DECLARE
  dequeue_options     DBMS_AQ.dequeue_options_t;
  message_properties  DBMS_AQ.message_properties_t;
  message_handle      RAW(16);
  message             test.message_typ;
BEGIN
  dequeue_options.dequeue_mode := DBMS_AQ.LOCKED;
  LOOP
    DBMS_AQ.dequeue(
      queue_name         => 'test.obj_queue',
      dequeue_options    => dequeue_options,
      message_properties => message_properties,
      payload            => message,
      msgid              => message_handle);
    DBMS_OUTPUT.PUT_LINE('Locked Message Text: ' || message.text);
    EXIT WHEN message.subject = 'PURPLE';
  END LOOP;
  dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
  dequeue_options.msgid        := message_handle;
  DBMS_AQ.DEQUEUE(
    queue_name         => 'test.obj_queue',
    dequeue_options    => dequeue_options,
    message_properties => message_properties,
    payload            => message,
    msgid              => message_handle);
  DBMS_OUTPUT.PUT_LINE('Removed Message Text: ' || message.text);
  COMMIT;
END;
/

See Also:  "Using Advanced Queuing Interfaces" in Oracle Objects for OLE Developer’s Guide for OO4O message-dequeuing examples

Dequeuing an Array of Messages

DBMS_AQ.DEQUEUE_ARRAY(
  queue_name              IN   VARCHAR2,
  dequeue_options         IN   dequeue_options_t,
  array_size              IN   PLS_INTEGER,
  message_properties_array OUT  message_properties_array_t,
  payload_array           OUT  VARRAY,
  msgid_array             OUT  msgid_array_t)
RETURN PLS_INTEGER;
Use the `DEQUEUE_ARRAY` function to dequeue an array of payloads and a corresponding array of message properties. The output is an array of payloads, message IDs, and message properties of the dequeued messages. The function returns the number of messages successfully dequeued.

Array dequeuing is not supported for buffered messages, but you can still use `DBMS_AQ.DEQUEUE_ARRAY` to dequeue buffered messages by setting `array_size` to 1.

The payload structure can be a VARRAY or nested table. The message identifiers are returned into an array of RAW(16) entries of type `DBMS_AQ.msgid_array_t`. The message properties are returned into an array of type `DBMS_AQ.message_properties_array_t`.

As with array operations in the relational world, it is not possible to provide a single optimum array size that will be correct in all circumstances. Application developers must experiment with different array sizes to determine the optimal value for their particular applications.

All dequeue options available with `DBMS_AQ.DEQUEUE` are also available with `DBMS_AQ.DEQUEUE_ARRAY`. Beginning with Oracle Database 10g Release 2 (10.2), you can choose to dequeue only persistent messages, only buffered messages, or both. In addition, the `navigation` attribute of `dequeue_options` offers two options specific to `DBMS_AQ.DEQUEUE_ARRAY`.

See Also: "Dequeueing Messages" on page 10-13

When dequeuing messages, you might want to dequeue all the messages for a transaction group with a single call. You might also want to dequeue messages that span multiple transaction groups. You can specify either of these methods by using one of the following navigation methods:

- NEXT_MESSAGE_ONE_GROUP
- FIRST_MESSAGE_ONE_GROUP
- NEXT_MESSAGE_MULTI_GROUP
- FIRST_MESSAGE_MULTI_GROUP

Navigation method `NEXT_MESSAGE_ONE_GROUP` dequeues messages that match the search criteria from the next available transaction group into an array. Navigation method `FIRST_MESSAGE_ONE_GROUP` resets the position to the beginning of the queue and dequeues all the messages in a single transaction group that are available and match the search criteria.

The number of messages dequeued is determined by an array size limit. If the number of messages in the transaction group exceeds `array_size`, then multiple calls to `DEQUEUE_ARRAY` must be made to dequeue all the messages for the transaction group.

Navigation methods `NEXT_MESSAGE_MULTI_GROUP` and `FIRST_MESSAGE_MULTI_GROUP` work like their `ONE_GROUP` counterparts, but they are not limited to a single transaction group. Each message that is dequeued into the array has an associated set of message properties. Message property `transaction_group` determines which messages belong to the same transaction group.

Example 10–20 dequeues the messages enqueued in Example 10–11 on page 10-11. It returns:

```
Number of messages dequeued: 2
```
Example 10–20  Dequeuing an Array of Messages

```
SET SERVEROUTPUT ON
DECLARE
  dequeue_options       DBMS_AQ.dequeue_options_t;
  msg_prop_array        DBMS_AQ.message_properties_array_t :=
                         DBMS_AQ.message_properties_array_t();
  payload_array         test.msg_table;
 msgid_array           DBMS_AQ.msgid_array_t;
  retval                PLS_INTEGER;
BEGIN
  retval := DBMS_AQ.DEQUEUE_ARRAY(
    queue_name               => 'test.obj_queue',
    dequeue_options          => dequeue_options,
    array_size               => 2,
    message_properties_array => msg_prop_array,
    payload_array            => payload_array,
   msgid_array              => msgid_array);
  DBMS_OUTPUT.PUT_LINE('Number of messages dequeued: ' || retval);
END;
/
```

Registering for Notification

```
DBMS_AQ.REGISTER(
  reg_list     IN SYS.AQ$_REG_INFO_LIST,
  reg_count    IN NUMBER);
```

This procedure registers an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

---

**Note:** In releases before Oracle Database 10g Release 2 (10.2), the Oracle® Database notification feature was not supported for queues with names longer than 30 characters. This restriction no longer applies. The 24-character limit on names of user-generated queues still applies. See "Creating a Queue" on page 8-12.

---

The `reg_list` parameter is a list of `SYS.AQ$_REG_INFO` objects. You can specify notification quality of service, a new feature in Oracle® Database 10g Release 2 (10.2), with the `qosflags` attribute of `SYS.AQ$_REG_INFO`.

**See Also:** "AQ Registration Information Type" on page 2-3 for more information on `SYS.AQ$_REG_INFO` objects.

The `reg_count` parameter specifies the number of entries in the `reg_list`. Each subscription requires its own `reg_list` entry. Interest in several subscriptions can be registered at one time.

When PL/SQL notification is received, the Oracle® Database message properties descriptor that the callback is invoked with specifies the `delivery_mode` of the message notified as `DBMS_AQ.PERSISTENT` or `DBMS_AQ.BUFFERED`.

**See Also:** "AQ Notification Descriptor Type" on page 2-5 for more information on the message properties descriptor.
If you register for e-mail notifications, then you must set the host name and port name for the SMTP server that will be used by the database to send e-mail notifications. If required, you should set the send-from e-mail address, which is set by the database as the sent from field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, then you might want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

An internal queue called `SYS.AQ_SRVNTFN_TABLE_Q` stores the notifications to be processed by the job queue processes. If notification fails, then Oracle® Database retries the failed notification up to MAX_RETRIES attempts.

---

**Note:** You can change the MAX_RETRIES and RETRY_DELAY properties of `SYS.AQ_SRVNTFN_TABLE_Q`. The new settings are applied across all notifications.

---

### Example 10–21 Registering for Notifications

```plsql
DECLARE
    reginfo   sys.aq$$_reg_info;
    reg_list  sys.aq$$_reg_info_list;
BEGIN
    reginfo := sys.aq$$_reg_info('test.obj_queue',
                                 DBMS_AQ.NAMESPACE_ANONYMOUS,
                                 'http://www.company.com:8080',
                                 HEXTORAW('FF'));
    reg_list := sys.aq$$_reg_info_list(reginfo);
    DBMS_AQ.REGISTER(
        reg_list     => reg_list,
        reg_count    => 1);
    COMMIT;
END;
/
```

### Unregistering for Notification

```plsql
DBMS_AQ.UNREGISTER(
    reg_list IN SYS.AQ$$_REG_INFO_LIST,
    reg_count IN NUMBER);
```

This procedure unregisters an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

### Posting for Subscriber Notification

```plsql
DBMS_AQ.POST(
    post_list IN SYS.AQ$$_POST_INFO_LIST,
    post_count IN NUMBER);
```

This procedure posts to a list of anonymous subscriptions, allowing all clients who are registered for the subscriptions to get notifications of persistent messages. This feature is not supported with buffered messages.
The `count` parameter specifies the number of entries in the `post_list`. Each posted subscription must have its own entry in the `post_list`. Several subscriptions can be posted to at one time.

The `post_list` parameter specifies the list of anonymous subscriptions to which you want to post. It has three attributes:

- **name**
  
  The `name` attribute specifies the name of the anonymous subscription to which you want to post.

- **namespace**
  
  The `namespace` attribute specifies the namespace of the subscription. To receive notifications from other applications through `DBMS_AQ.POST` the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`.

- **payload**
  
  The `payload` attribute specifies the payload to be posted to the anonymous subscription. It is possible for no payload to be associated with this call.

This call provides a best-effort guarantee. A notification goes to registered clients at most once. This call is primarily used for lightweight notification. If an application needs more rigid guarantees, then it can enqueue to a queue.

**Example 10–22 Posting Object-Type Messages**

```sql
DECLARE
    postinfo    sys.aq$_post_info;
    post_list   sys.aq$_post_info_list;
BEGIN
    postinfo := sys.aq$_post_info('test.obj_queue',0,HEXTORAW('FF'));
    post_list := sys.aq$_post_info_list(postinfo);
    DBMS_AQ.POST(
        post_list  => post_list,
        post_count => 1);
    COMMIT;
END;
/
```

**Adding an Agent to the LDAP Server**

```sql
DBMS_AQ.BIND_AGENT(
    agent   IN SYS.AQ$_AGENT,
    certificate IN VARCHAR2 default NULL);
```

This procedure creates an entry for an Oracle® Database agent in the Lightweight Directory Access Protocol (LDAP) server.

The `agent` parameter specifies the Oracle® Database Agent that is to be registered in LDAP server.

**See Also:**  "AQ Agent Type" on page 2-2

The `certificate` parameter specifies the location (LDAP distinguished name) of the `OrganizationalPerson` entry in LDAP whose digital certificate (attribute `usercert`) is to be used for this agent. For example, "cn=OE, cn=ACME, cn=com" is a distinguished name for a `OrganizationalPerson OE` whose certificate
will be used with the specified agent. If the agent does not have a digital certificate, then this parameter is defaulted to null.

Removing an Agent from the LDAP Server

```sql
DBMS_AQ.UNBIND_AGENT(
    agent    IN SYS.AQ$_AGENT);
```

This procedure removes the entry for an Oracle® Database agent from the LDAP server.
This chapter describes the Oracle Java Message Service (JMS) interface to Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- General Features of JMS and Oracle JMS
- Structured Payload/Message Types in JMS
- JMS Point-to-Point Model Features
- JMS Publish/Subscribe Model Features
- JMS MessageProducer Features
- JMS Message Consumer Features
- JMS Propagation
- Message Transformation with JMS AQ
- J2EE Compliance

General Features of JMS and Oracle JMS

This section contains these topics:

- JMS Connection and Session
- JMS Destination
- System-Level Access Control in JMS
- Destination-Level Access Control in JMS
- Retention and Message History in JMS
- Supporting Oracle Real Application Clusters in JMS
- Supporting Statistics Views in JMS

JMS Connection and Session

This section contains these topics:

- ConnectionFactory Objects
- Using AQ.jmsFactory to Obtain ConnectionFactory Objects
- Using JNDI to Look Up ConnectionFactory Objects
- JMS Connection
JMS Session

ConnectionFactory Objects
A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider. In this case Oracle JMS, part of Oracle Database, is the JMS provider.

The three types of ConnectionFactory objects are:
- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory

Using AQjmsFactory to Obtain ConnectionFactory Objects
You can use the AQjmsFactory class to obtain a handle to a ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory object.

To obtain a ConnectionFactory, which supports both point-to-point and publish/subscribe operations, use AQjmsFactory.getConnectionFactory(). To obtain a QueueConnectionFactory, use AQjmsFactory.getQueueConnectionFactory(). To obtain a TopicConnectionFactory, use AQjmsFactory.getTopicConnectionFactory().

The ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory can be created using hostname, port number, and SID driver or by using JDBC URL and properties.

Using JNDI to Look Up ConnectionFactory Objects
A JMS administrator can register ConnectionFactory objects in a Lightweight Directory Access Protocol (LDAP) server. The following setup is required to enable Java Naming and Directory Interface (JNDI) lookup in JMS:

1. Register Database

When the Oracle Database server is installed, the database must be registered with the LDAP server. This can be accomplished using the Database Configuration Assistant (DBCA). Figure 11–1 shows the structure of Oracle® Database entries in the LDAP server. ConnectionFactory information is stored under <cn=OracleDBConnections>, while topics and queues are stored under <cn=OracleDBQueues>. 
2. Set Parameter `GLOBAL_TOPIC_ENABLED`.

The `GLOBAL_TOPIC_ENABLED` system parameter for the database must be set to `TRUE`. This ensures that all queues and topics created in Oracle® Database are automatically registered with the LDAP server. This parameter can be set by using `ALTER SYSTEM SET GLOBAL_TOPIC_ENABLED = TRUE`.

3. Register ConnectionFactory Objects

After the database has been set up to use an LDAP server, the JMS administrator can register `ConnectionFactory`, `QueueConnectionFactory`, and `TopicConnectionFactory` objects in LDAP by using `AQjmsFactory.registerConnectionFactory()`.

The registration can be accomplished in one of the following ways:

- **Connect directly to the LDAP server**
  
  The user must have the `GLOBAL_AQ_USER_ROLE` to register connection factories in LDAP.

  To connect directly to LDAP, the parameters for the `registerConnectionFactory` method include the LDAP context, the name of the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory`, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

- **Connect to LDAP through the database server**
  
  The user can log on to Oracle Database first and then have the database update the LDAP entry. The user that logs on to the database must have the `AQ_ADMINISTRATOR_ROLE` to perform this operation.

  To connect to LDAP through the database server, the parameters for the `registerConnectionFactory` method include a JDBC connection (to a user having `AQ_ADMINISTRATOR_ROLE`), the name of the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory`, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

**JMS Connection**

A **JMS Connection** is an active connection between a client and its JMS provider. A **JMS Connection** performs several critical services:
- Encapsulates either an open connection or a pool of connections with a JMS provider
- Typically represents an open TCP/IP socket (or a set of open sockets) between a client and a provider’s service daemon
- Provides a structure for authenticating clients at the time of its creation
- Creates Sessions
- Provides connection metadata
- Supports an optional ExceptionListener

A JMS Connection to the database can be created by invoking createConnection(),createQueueConnection(), or createTopicConnection() and passing the parameters username and password on the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory object respectively.

Some of the methods that are supported on the Connection object are:

- start()
  This method starts or restart delivery of incoming messages.
- stop()
  This method temporarily stops delivery of incoming messages. When a Connection object is stopped, delivery to all of its message consumers is inhibited. Also, synchronous receive's block and messages are not delivered to message listener.
- close()
  This method closes the JMS session and releases all associated resources.
- createSession(true, 0)
  This method creates a JMS Session using a JMS Connection instance.
- createQueueSession(true, 0)
  This method creates a QueueSession.
- createTopicSession(true, 0)
  This method creates a TopicSession.
- setExceptionListener(ExceptionListener)
  This method sets an exception listener for the Connection. This allows a client to be notified of a problem asynchronously. If a Connection only consumes messages, then it has no other way to learn it has failed.
- getExceptionListener()
  This method gets the ExceptionListener for this Connection.

A JMS client typically creates a Connection, a Session and a number of MessageProducer and MessageConsumer objects. In the current version only one open Session for each Connection is allowed, except in the following cases:

- If the JDBC oci8 driver is used to create the JMS connection
- If the user provides an OracleOCIConnectionPool instance during JMS connection creation
When a Connection is created it is in stopped mode. In this state no messages can be delivered to it. It is typical to leave the Connection in stopped mode until setup is complete. At that point the Connection start() method is called and messages begin arriving at the Connection consumers. This setup convention minimizes any client confusion that can result from asynchronous message delivery while the client is still in the process of setup.

It is possible to start a Connection and to perform setup subsequently. Clients that do this must be prepared to handle asynchronous message delivery while they are still in the process of setting up. A MessageProducer can send messages while a Connection is stopped.

**JMS Session**

A JMS Session is a single threaded context for producing and consuming messages. Although it can allocate provider resources outside the Java Virtual Machine (JVM), it is considered a lightweight JMS object. A Session serves several purposes:

- Constitutes a factory for MessageProducer and MessageConsumer objects
- Provides a way to get a handle to destination objects (queues/topics)
- Supplies provider-optimized message factories
- Supports a single series of transactions that combines work spanning session MessageProducer and MessageConsumer objects, organizing these into units
- Defines a serial order for the messages it consumes and the messages it produces
- Serializes execution of MessageListener objects registered with it

In Oracle Database 10g, you can create as many JMS Sessions as resources allow using a single JMS Connection, when using either JDBC thin or JDBC thick (OCI) drivers.

Because a provider can allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough. The same is true for MessageProducer and MessageConsumer objects created by a Session.

Methods on the Session object include:

- **commit()**
  This method commits all messages performed in the transaction and releases locks currently held.

- **rollback()**
  This method rolls back any messages accomplished in the transaction and release locks currently held.

- **close()**
  This method closes the Session.

- **getDBConnection()**
  This method gets a handle to the underlying JDBC connection. This handle can be used to perform other SQL DML operations as part of the same Session. The method is specific to Oracle JMS.

- **acknowledge()**
This method acknowledges message receipt in a nontransactional session.

- **recover()**
  This method restarts message delivery in a nontransactional session. In effect, the series of delivered messages in the session is reset to the point after the last acknowledged message.

The following are some Oracle JMS extensions:

- **createQueueTable()**
  This method creates a **queue table**.

- **getQueueTable()**
  This method gets a handle to an existing queue table.

- **createQueue()**
  This method creates a queue.

- **getQueue()**
  This method gets a handle to an existing queue.

- **createTopic()**
  This method creates a topic.

- **getTopic()**
  This method gets a handle to an existing topic.

The `Session` object must be cast to `AQjmsSession` to use any of the extensions.

---

**Note:** The JMS specification expects providers to return null messages when receives are accomplished on a JMS `Connection` instance that has not been started.

After you create a `javax.jms.Connection` instance, you must call the `start()` method on it before you can receive messages. If you add a line like `t_conn.start();` any time after the connection has been created, but before the actual receive, then you can receive your messages.

---

**JMS Destination**

A **Destination** is an object a client uses to specify the destination where it sends messages, and the source from which it receives messages. A **Destination** object can be a **Queue** or a **Topic**. In Oracle® Database, these map to a `schema.queue` at a specific database. **Queue** maps to a single-consumer queue, and **Topic** maps to a multiconsumer queue.

**Using a JMS Session to Obtain Destination Objects**

Destination objects are created from a `Session` object using the following domain-specific `Session` methods:

- **AQjmsSession.getQueue(queue_owner, queue_name)**
  This method gets a handle to a JMS queue.

- **AQjmsSession.getTopic(topic_owner, topic_name)**
This method gets a handle to a JMS topic.

Using JNDI to Look Up Destination Objects
The database can be configured to register schema objects with an LDAP server. If a database has been configured to use LDAP and the GLOBAL_TOPIC_ENABLED parameter has been set to TRUE, then all JMS queues and topics are automatically registered with the LDAP server when they are created. The administrator can also create aliases to the queues and topics registered in LDAP. Queues and topics that are registered in LDAP can be looked up through JNDI using the name or alias of the queue or topic.

See Also: "Adding an Alias to the LDAP Server" on page 8-30

JMS Destination Methods
Methods on the Destination object include:
- alter()
  This method alters a Queue or a Topic.
- schedulePropagation()
  This method schedules propagation from a source to a destination.
- unschedulePropagation()
  This method unschedules a previously scheduled propagation.
- enablePropagationSchedule()
  This method enables a propagation schedule.
- disablePropagationSchedule()
  This method disables a propagation schedule.
- start()
  This method starts a Queue or a Topic. The queue can be started for enqueue or dequeue. The topic can be started for publish or subscribe.
- stop()
  This method stops a Queue or a Topic. The queue is stopped for enqueue or dequeue. The topic is stopped for publish or subscribe.
- drop()
  This method drops a Queue or a Topic.

System-Level Access Control in JMS
Oracle8i or higher supports system-level access control for all queuing operations. This feature allows an application designer or DBA to create users as queue administrators. A queue administrator can invoke administrative and operational JMS interfaces on any queue in the database. This simplifies administrative work, because all administrative scripts for the queues in a database can be managed under one schema.

See Also: "Oracle Enterprise Manager Support" on page 4-6

When messages arrive at the destination queues, sessions based on the source queue schema name are used for enqueuing the newly arrived messages into the destination.
queues. This means that you must grant enqueue privileges for the destination queues to schemas of the source queues.

To propagate to a remote destination queue, the login user (specified in the database link in the address field of the agent structure) should either be granted the `ENQUEUE_ANY` privilege, or be granted the rights to enqueue to the destination queue. However, you are not required to grant any explicit privileges if the login user in the database link also owns the queue tables at the destination.

**Destination-Level Access Control in JMS**

Oracle8i or higher supports access control for enqueue and dequeue operations at the queue or topic level. This feature allows the application designer to protect queues and topics created in one schema from applications running in other schemas. You can grant only minimal access privileges to the applications that run outside the schema of the queue or topic. The supported access privileges on a queue or topic are `ENQUEUE`, `DEQUEUE` and `ALL`.

**Retention and Message History in JMS**

Messages are often related to each other. For example, if a message is produced as a result of the consumption of another message, then the two are related. As the application designer, you may want to keep track of such relationships. Oracle® Database allows users to retain messages in the queue table, which can then be queried in SQL for analysis.

Along with retention and message identifiers, Oracle® Database lets you automatically create message journals, also called tracking journals or event journals. Taken together, retention, message identifiers and SQL queries make it possible to build powerful message warehouses.

**Supporting Oracle Real Application Clusters in JMS**

Oracle Real Application Clusters (RAC) can be used to improve Oracle® Database performance by allowing different queues to be managed by different instances. You do this by specifying different instance affinities (preferences) for the queue tables that store the queues. This allows queue operations (enqueue/dequeue) or topic operations (publish/subscribe) on different queues or topics to occur in parallel.

The Oracle® Database queue monitor process continuously monitors the instance affinities of the queue tables. The queue monitor assigns ownership of a queue table to the specified primary instance if it is available, failing which it assigns it to the specified secondary instance.

If the owner instance of a queue table terminates, then the queue monitor changes ownership to a suitable instance such as the secondary instance.

Oracle® Database propagation is able to make use of Real Application Clusters, although it is transparent to the user. The affinities for jobs submitted on behalf of the propagation schedules are set to the same values as that of the affinities of the respective queue tables. Thus, a `job_queue_process` associated with the owner instance of a queue table is handling the propagation from queues stored in that queue table, thereby minimizing pinging.
Supporting Statistics Views in JMS

Each instance keeps its own Oracle® Database statistics information in its own System Global Area (SGA), and does not have knowledge of the statistics gathered by other instances. Then, when a GV$AQ view is queried by an instance, all other instances funnel their statistics information to the instance issuing the query.

The GV$AQ view can be queried at any time to see the number of messages in waiting, ready or expired state. The view also displays the average number of seconds messages have been waiting to be processed.

See Also:  "(G)V$AQ: Number of Messages in Different States in Database" on page 9-8

Structured Payload/Message Types in JMS

JMS messages are composed of a header, properties, and a body. The header consists of header fields, which contain values used by both clients and providers to identify and route messages. All messages support the same set of header fields.

Properties are optional header fields. In addition to standard properties defined by JMS, there can be provider-specific and application-specific properties.

The body is the message payload. JMS defines various types of message payloads, and a type that can store JMS messages of any or all JMS-specified message types.

This section contains these topics:

- JMS Message Headers
- JMS Message Properties
- JMS Message Bodies
- Using Message Properties with Different Message Types
- Buffered Messaging with Oracle JMS

JMS Message Headers

A JMS message header contains the following fields:

- **JMSDestination**

  This field contains the destination to which the message is sent. In Oracle® Database this corresponds to the destination queue/topic. It is a Destination type set by JMS after the Send method has completed.

- **JMSDeliveryMode**

  This field determines whether the message is logged or not. JMS supports PERSISTENT delivery (where messages are logged to stable storage) and NONPERSISTENT delivery (messages not logged). It is a INTEGER set by JMS after
the Send method has completed. JMS permits an administrator to configure JMS to override the client-specified value for JMSDeliveryMode.

- **JMSMessageID**
  This field uniquely identifies a message in a provider. All message IDs must begin with the string ID:. It is a String type set by JMS after the Send method has completed.

- **JMSTimeStamp**
  This field contains the time the message was handed over to the provider to be sent. This maps to Oracle® Database message enqueue time. It is a Long type set by JMS after the Send method has completed.

- **JMSCorrelationID**
  This field can be used by a client to link one message with another. It is a String type set by the JMS client.

- **JMSReplyTo**
  This field contains a Destination type supplied by a client when a message is sent. Clients can use oracle.jms.AQ.jmsAgent; javax.jms.Queue; or javax.jms.Topic.

- **JMSType**
  This field contains a message type identifier supplied by a client at send time. It is a String type. For portability Oracle recommends that the JMSType be symbolic values.

- **JMSExpiration**
  This field is the sum of the enqueue time and the TimeToLive in non-J2EE compliance mode. In compliant mode, the JMSExpiration header value in a dequeued message is the sum of JMSTimeStamp when the message was enqueued (Greenwich Mean Time, in milliseconds) and the TimeToLive (in milliseconds). It is a Long type set by JMS after the Send method has completed. JMS permits an administrator to configure JMS to override the client-specified value for JMSExpiration.

- **JMSPriority**
  This field contains the priority of the message. It is a INTEGER set by JMS after the Send method has completed. In J2EE-compliance mode, the permitted values for priority are 0–9, with 9 the highest priority and 4 the default, in conformance with the Sun Microsystems JMS 1.1 standard. Noncompliant mode is the default. JMS permits an administrator to configure JMS to override the client-specified value for JMSPriority.

- **JMSRedelivered**
  This field is a Boolean set by the JMS provider.

**See Also:** "J2EE Compliance" on page 11-29

### JMS Message Properties

JMS properties are set either explicitly by the client or automatically by the JMS provider (these are generally read-only). Some JMS properties are set using the parameters specified in Send and Receive operations.
Properties add optional header fields to a message. Properties allow a client, using a messageSelector, to have a JMS provider select messages on its behalf using application-specific criteria. Property names are strings and values can be: Boolean, byte, short, int, long, float, double, and string.

JMS-defined properties, which all begin with "JMSX", include the following:

- **JMSXUserID**
  This field is the identity of the user sending the message. It is a String type set by JMS after the Send method has completed.

- **JMSXAppID**
  This field is the identity of the application sending the message. It is a String type set by JMS after the Send method has completed.

- **JMSXDeliveryCount**
  This field is the number of message delivery attempts. It is an Integer set by JMS after the Send method has completed.

- **JMSXGroupid**
  This field is the identity of the message group that this message belongs to. It is a String type set by the JMS client.

- **JMSXGroupSeq**
  This field is the sequence number of a message within a group. It is an Integer set by the JMS client.

- **JMSXRcvTimeStamp**
  This field is the time the message was delivered to the consumer (dequeue time). It is a String type set by JMS after the Receive method has completed.

- **JMSXState**
  This field is the message state, set by the provider. The message state can be WAITING, READY, EXPIRED, or RETAINED.

Oracle-specific JMS properties, which all begin with JMS_Oracle, include the following:

- **JMS_OracleExcpQ**
  This field is the queue name to send the message to if it cannot be delivered to the original destination. It is a String type set by the JMS client. Only destinations of type EXCEPTION can be specified in the JMS_OracleExcpQ property.

- **JMS_OracleDelay**
  This field is the time in seconds to delay the delivery of the message. It is an Integer set by the JMS client. This can affect the order of message delivery.

- **JMS_OracleOriginalMessageId**
  This field is set to the message identifier of the message in the source if the message is propagated from one destination to another. It is a String type set by the JMS provider. If the message is not propagated, then this property has the same value as JMSMessageId.

A client can add additional header fields to a message by defining properties. These properties can then be used in a messageSelector to select specific messages.
Structured Payload/Message Types in JMS

JMS Message Bodies

JMS provides five forms of message body:

- StreamMessage
- BytesMessage
- MapMessage
- TextMessage
- ObjectMessage
- AdtMessage

StreamMessage

A StreamMessage object is used to send a stream of Java primitives. It is filled and read sequentially. It inherits from Message and adds a StreamMessage body. Its methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects. To use StreamMessage objects, create the queue table with the SYS.AQ$JMS_STREAM_MESSAGE or AQ$JMS_MESSAGE payload types.

StreamMessage objects support the conversions shown in Table 11–1. A value written as the row type can be read as the column type.

<table>
<thead>
<tr>
<th>Table 11–1 StreamMessage Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Boolean</td>
</tr>
<tr>
<td>byte</td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>long</td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td>double</td>
</tr>
<tr>
<td>string</td>
</tr>
<tr>
<td>byte[]</td>
</tr>
</tbody>
</table>

BytesMessage

A BytesMessage object is used to send a message containing a stream of uninterpreted bytes. It inherits Message and adds a BytesMessage body. The receiver of the message interprets the bytes. Its methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

This message type is for client encoding of existing message formats. If possible, one of the other self-defining message types should be used instead.

The primitive types can be written explicitly using methods for each type. They can also be written generically as objects. To use BytesMessage objects, create the queue table with SYS.AQ$JMS_BYTES_MESSAGE or AQ$JMS_MESSAGE payload types.
MapMessage

A MapMessage object is used to send a set of name-value pairs where the names are String types, and the values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from Message and adds a MapMessage body. The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects.

To use MapMessage objects, create the queue table with the SYS.AQ$_JMS_MAP_ MESSAGE or AQ$_JMS_MESSAGE payload types. MapMessage objects support the conversions shown in Table 11–2. An “X” in the table means that a value written as the row type can be read as the column type.

<table>
<thead>
<tr>
<th>Input</th>
<th>Boolean</th>
<th>byte</th>
<th>short</th>
<th>char</th>
<th>int</th>
<th>long</th>
<th>float</th>
<th>double</th>
<th>String</th>
<th>byte[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>byte</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>short</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>char</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>int</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>long</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>float</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>double</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>string</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>byte[]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

TextMessage

A TextMessage object is used to send a message containing a java.lang.StringBuffer. It inherits from Message and adds a TextMessage body. The text information can be read or written using methods getText() and setText(...). To use TextMessage objects, create the queue table with the SYS.AQ$_JMS_TEXT_MESSAGE or AQ$_JMS_MESSAGE payload types.

ObjectMessage

An ObjectMessage object is used to send a message that contains a serializable Java object. It inherits from Message and adds a body containing a single Java reference. Only serializable Java objects can be used. If a collection of Java objects must be sent, then one of the collection classes provided in JDK 1.4 can be used. The objects can be read or written using the methods getObject() and setObject(...). To use ObjectMessage objects, create the queue table with the SYS.AQ$_JMS_OBJECT_ MESSAGE or AQ$_JMS_MESSAGE payload types.

AdtMessage

An AdtMessage object is used to send a message that contains a Java object that maps to an Oracle object type. These objects inherit from Message and add a body containing a Java object that implements the CustomDatum or ORAData interface.

See Also: Oracle Database Java Developer’s Guide for information about the CustomDatum and ORAData interfaces.
To use `AdtMessage` objects, create the queue table with payload type as the Oracle object type. The `AdtMessage` payload can be read and written using the `getAdtPayload` and `setAdtPayload` methods.

You can also use an `AdtMessage` object to send messages to queues of type `SYS.XMLType`. You must use the `oracle.xdb.XMLType` class to create the message.

For `AdtMessage` objects, the client can get:

- `JMSXDeliveryCount`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

**Using Message Properties with Different Message Types**

The following message properties can be set by the client using the `setProperty` call. For `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, and `MapMessage` objects, the client can set:

- `JMSXAppID`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

For `AdtMessage` objects, the client can set:

- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

The following message properties can be obtained by the client using the `getProperty` call. For `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, and `MapMessage` objects, the client can get:

- `JMSXuserID`
- `JMSXAppID`
- `JMSXDeliveryCount`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`
- `JMS_OracleOriginalMessageID`
Buffered Messaging with Oracle JMS

Users can send a nonpersistent JMS message by specifying the deliveryMode to be NON_PERSISTENT when sending a message. JMS nonpersistent messages are not required to be logged to stable storage, so they can be lost after a JMS system failure. JMS nonpersistent messages are similar to the buffered messages now available in Oracle® Database, but there are also important differences between the two.

Note: Do not confuse Oracle JMS nonpersistent messages with Oracle® Database nonpersistent queues, which are deprecated in Oracle Database 10g Release 2 (10.2).

See Also:
- "Buffered Messaging" on page 1-12
- Appendix A, "Nonpersistent Queues"

Transaction Commits and Client Acknowledgments

The JMS deliveryMode is orthogonal to the transaction attribute of a message. JMS nonpersistent messages can be sent and received by either a transacted session or a nontransacted session. If a JMS nonpersistent message is sent and received by a transacted session, then the effect of the JMS operation is only visible after the transacted session commits. If it is received by a nontransacted session with CLIENT_ACKNOWLEDGE acknowledgment mode, then the effect of receiving this message is only visible after the client acknowledges the message. Without the acknowledgment, the message is not removed and will be redelivered if the client calls Session.recover.

Oracle® Database buffered messages, on the other hand, do not support these transaction or acknowledgment concepts. Both sending and receiving a buffered message must be in the IMMEDIATE visibility mode. The effects of the sending and receiving operations are therefore visible to the user immediately, no matter whether the session is committed or the messages are acknowledged.

Different APIs

Messages sent with the regular JMS send and publish methods are treated by Oracle® Database as persistent messages. The regular JMS receive methods receive only AQ persistent messages. To send and receive buffered messages, you must use the Oracle extension APIs bufferSend, bufferPublish, and bufferReceive.

See Also: Oracle Streams Advanced Queuing Java API Reference for more information on bufferSend, bufferPublish, and bufferReceive

Payload Limits

The Oracle® Database implementation of buffered messages does not support LOB attributes. This places limits on the payloads for the five types of standard JMS messages:

- JMS TextMessage payloads cannot exceed 4000 bytes.

This limit might be even lower with some database character sets, because during the Oracle JMS character set conversion, Oracle JMS sometimes must make a conservative choice of using CLOB instead of VARCHAR to store the text payload in the database.
JMS Point-to-Point Model Features

- JMS BytesMessage payloads cannot exceed 2000 bytes.
- For all other Oracle JMS ADT messages, the corresponding Oracle database ADT cannot contain LOB attributes.

Different Constants

The Oracle® Database and Oracle JMS APIs use different numerical values to designate buffered and persistent messages, as shown in Table 11–3.

<table>
<thead>
<tr>
<th>API</th>
<th>Persistent Message</th>
<th>Buffered Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle® Database</td>
<td>PERSISTENT := 1</td>
<td>BUFFERED := 2</td>
</tr>
<tr>
<td>Oracle JMS</td>
<td>PERSISTENT := 2</td>
<td>NON_PERSISTENT := 1</td>
</tr>
</tbody>
</table>

JMS Point-to-Point Model Features

In the point-to-point model, clients exchange messages from one point to another. Message producers and consumers send and receive messages using single-consumer queues. An administrator creates the single-consumer queues with the createQueue method in AQjmsSession. Before they can be used, the queues must be enabled for enqueue/dequeue using the start call in AQjmsDestination. Clients obtain a handle to a previously created queue using the getQueue method on AQjmsSession.

In a single-consumer queue, a message can be consumed exactly once by a single consumer. If there are multiple processes or operating system threads concurrently dequeuing from the same queue, then each process dequeues the first unlocked message at the head of the queue. A locked message cannot be dequeued by a process other than the one that has created the lock.

After processing, the message is removed if the retention time of the queue is 0, or it is retained for a specified retention time. As long as the message is retained, it can be either queried using SQL on the queue table view or dequeued by specifying the message identifier of the processed message in a QueueBrowser.

QueueSender

A client uses a QueueSender to send messages to a queue. It is created by passing a queue to the createSender method in a client Session. A client also has the option of creating a QueueSender without supplying a queue. In that case a queue must be specified on every send operation.

A client can specify a default delivery mode, priority and TimeToLive for all messages sent by the QueueSender. Alternatively, the client can define these options for each message.

QueueReceiver

A client uses a QueueReceiver to receive messages from a queue. It is created using the createQueueReceiver method in a client Session. It can be created with or without a messageSelector.
QueueBrowser

A client uses a QueueBrowser to view messages on a queue without removing them. The browser method returns a `java.util.Enumeration` that is used to scan messages in the queue. The first call to `nextElement` gets a snapshot of the queue. A QueueBrowser can be created with or without a messageSelector.

A QueueBrowser can also optionally lock messages as it is scanning them. This is similar to a "SELECT... for UPDATE" command on the message. This prevents other consumers from removing the message while they are being scanned.

MessageSelector

A messageSelector allows the client to restrict messages delivered to the consumer to those that match the messageSelector expression. A messageSelector for queues containing payloads of type `TextMessage`, `StreamMessage`, `BytesMessage`, `ObjectMessage`, or `MapMessage` can contain any expression that has one or more of the following:

- **JMS message identifier prefixed with "ID:"**
  
  ```
 JMISMessageID = 'ID:23452345'
  ```

- **JMS message header fields or properties**
  
  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

  ```
  JMSCorrelationID LIKE 'RE%'
  ```

- **User-defined message properties**
  
  ```
  color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
  ```

The messageSelector for queues containing payloads of type `AdtMessage` can contain any expression that has one or more of the following:

- **Message identifier without the "ID:" prefix**
  
  ```
  msgid = '23434556567676767'
  ```

- **Priority, correlation identifier, or both**
  
  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- **Message payload**
  
  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

JMS Publish/Subscribe Model Features

This section contains these topics:

- **JMS Publish/Subscribe Overview**
- **DurableSubscriber**
- **RemoteSubscriber**
- **TopicPublisher**
- **Recipient Lists**
- **TopicReceiver**
JMS Publish/Subscribe Model Features

- **TopicBrowser**
- **Setting Up JMS Publish/Subscribe Operations**

**JMS Publish/Subscribe Overview**

JMS enables flexible and dynamic communication between applications functioning as publishers and applications playing the role of subscribers. The applications are not coupled together; they interact based on messages and message content.

In distributing messages, publisher applications are not required to handle or manage message recipients explicitly. This allows new subscriber applications to be added dynamically without changing any publisher application logic.

Similarly, subscriber applications receive messages based on message content without regard to which publisher applications are sending messages. This allows new publisher applications to be added dynamically without changing any subscriber application logic.

Subscriber applications specify interest by defining a rule-based subscription on message properties or the message content of a topic. The system automatically routes messages by computing recipients for published messages using the rule-based subscriptions.

In the publish/subscribe model, messages are published to and received from topics. A topic is created using the `CreateTopic()` method in an `AQjmsSession`. A client can obtain a handle to a previously-created topic using the `getTopic()` method in `AQjmsSession`.

**DurableSubscriber**

A client creates a **DurableSubscriber** with the `createDurableSubscriber()` method in a client `Session`. It can be created with or without a `messageSelector`.

A `messageSelector` allows the client to restrict messages delivered to the subscriber to those that match the selector. The syntax for the selector is described in detail in `createDurableSubscriber` in *Oracle Streams Advanced Queuing Java API Reference*.

See Also: "**MessageSelector**" on page 11-17

When subscribers use the same name, durable subscriber action depends on the J2EE compliance mode set for an **Oracle Java Message Service** (OJMS) client at runtime.

See Also: "**J2EE Compliance**" on page 11-29

In noncompliant mode, two durable **TopicSubscriber** objects with the same name can be active against two different topics. In compliant mode, durable subscribers with the same name are not allowed. If two subscribers use the same name and are created against the same topic, but the selector used for each subscriber is different, then the underlying Oracle® Database subscription is altered using the internal `DBMS_AQJMS.ALTER_SUBSCRIBER()` call.

If two subscribers use the same name and are created against two different topics, and if the client that uses the same subscription name also originally created the subscription name, then the existing subscription is dropped and the new subscription is created.

If two subscribers use the same name and are created against two different topics, and if a different client (a client that did not originate the subscription name) uses an
existing subscription name, then the subscription is not dropped and an error is thrown. Because it is not known if the subscription was created by JMS or PL/SQL, the subscription on the other topic should not be dropped.

**RemoteSubscriber**

Remote subscribers are defined using the `createRemoteSubscriber` call. The remote subscriber can be a specific consumer at the remote topic or all subscribers at the remote topic.

A remote subscriber is defined using the `AQjmsAgent` structure. An `AQjmsAgent` consists of a name and address. The name refers to the `consumer_name` at the remote topic. The address refers to the remote topic:

```
schema.topic_name[@dblink]
```

To publish messages to a particular consumer at the remote topic, the `subscription_name` of the `recipient` at the remote topic must be specified in the `name` field of `AQjmsAgent`. The remote topic must be specified in the `address` field of `AQjmsAgent`.

To publish messages to all subscribers of the remote topic, the `name` field of `AQjmsAgent` must be set to null. The remote topic must be specified in the `address` field of `AQjmsAgent`.

**TopicPublisher**

Messages are published using `TopicPublisher`, which is created by passing a `Topic` to a `createPublisher` method. A client also has the option of creating a `TopicPublisher` without supplying a `Topic`. In this case, a `Topic` must be specified on every publish operation. A client can specify a default delivery mode, priority and `TimeToLive` for all messages sent by the `TopicPublisher`. It can also specify these options for each message.

**Recipient Lists**

In the JMS publish/subscribe model, clients can specify explicit recipient lists instead of having messages sent to all the subscribers of the topic. These recipients may or may not be existing subscribers of the topic. The recipient list overrides the subscription list on the topic for this message. Recipient lists functionality is an Oracle extension to JMS.

**TopicReceiver**

If the recipient name is explicitly specified in the recipient list, but that recipient is not a subscriber to the queue, then messages sent to it can be received by creating a `TopicReceiver`. If the subscriber name is not specified, then clients must use durable subscribers at the remote site to receive messages. `TopicReceiver` is an Oracle extension to JMS.

A `TopicReceiver` can be created with a `messageSelector`. This allows the client to restrict messages delivered to the recipient to those that match the selector.

See Also: "MessageSelector" on page 11-17
**TopicBrowser**

A client uses a TopicBrowser to view messages on a topic without removing them. The browser method returns a `java.util.Enumeration` that is used to scan topic messages. Only durable subscribers are allowed to create a TopicBrowser. The first call to `nextElement` gets a snapshot of the topic.

**See Also:** "Creating a TopicBrowser for Standard JMS Messages" on page 14-19

A TopicBrowser can optionally lock messages as it is scanning them. This is similar to a `SELECT...FOR UPDATE` command on the message. This prevents other consumers from removing the message while it is being scanned.

**See Also:** "Creating a TopicBrowser for Standard JMS Messages, Locking Messages" on page 14-20

A TopicBrowser can be created with a messageSelector. This allows the client to restrict messages delivered to the browser to those that match the selector.

**See Also:** "MessageSelector" on page 11-17

TopicBrowser supports a purge feature. This allows a client using a TopicBrowser to discard all messages that have been seen during the current browse operation on the topic. A purge is equivalent to a destructive receive of all of the seen messages (as if performed using a TopicSubscriber).

For a purge, a message is considered seen if it has been returned to the client using a call to the `nextElement()` operation on the `java.lang.Enumeration` for the TopicBrowser. Messages that have not yet been seen by the client are not discarded during a purge. A purge operation can be performed multiple times on the same TopicBrowser.

The effect of a purge becomes stable when the JMS Session used to create the TopicBrowser is committed. If the operations on the session are rolled back, then the effects of the purge operation are also undone.

**See Also:** "Browsing Messages Using a TopicBrowser" on page 14-22

**Setting Up JMS Publish/Subscribe Operations**

Follow these steps to use the publish/subscribe model of communication in JMS:

1. Set up one or more topics to hold messages. These topics represent an area or subject of interest. For example, a topic can represent billed orders.

2. Enable enqueue/dequeue on the topic using the `start` call in `AQ.jmsDestination`.

3. Create a set of durable subscribers. Each subscriber can specify a `messageSelector` that selects the messages that the subscriber wishes to receive. A null `messageSelector` indicates that the subscriber wishes to receive all messages published on the topic.

Subscribers can be local or remote. Local subscribers are durable subscribers defined on the same topic on which the message is published. Remote subscribers are other topics, or recipients on other topics that are defined as subscribers to a particular queue. In order to use remote subscribers, you must set up propagation...
between the source and destination topics. Remote subscribers and propagation are Oracle extensions to JMS.

**See Also:** "Managing Propagations" on page 8-23

4. Create `TopicPublisher` objects using the `createPublisher()` method in the `publisher` `Session`. Messages are published using the `publish` call. Messages can be published to all subscribers to the topic or to a specified subset of recipients on the topic.

5. Subscribers receive messages on the topic by using the `receive` method.

6. Subscribers can also receive messages asynchronously by using message listeners.

**See Also:** "Listening to One or More Queues" on page 10-11

### JMS MessageProducer Features

- Priority and Ordering of Messages
- Specifying a Message Delay
- Specifying a Message Expiration
- Message Grouping

#### Priority and Ordering of Messages

Message ordering dictates the order in which messages are received from a queue or topic. The ordering method is specified when the queue table for the queue or topic is created. Currently, Oracle® Database supports ordering on message priority and enqueue time, producing four possible ways of ordering:

- **First-In, First-Out (FIFO)**
  
  If enqueue time was chosen as the ordering criteria, then messages are received in the order of the enqueue time. The enqueue time is assigned to the message by Oracle® Database at message publish/send time. This is also the default ordering.

- **Priority Ordering**
  
  If priority ordering was chosen, then each message is assigned a priority. Priority can be specified as a message property at publish/send time by the `MessageProducer`. The messages are received in the order of the priorities assigned.

- **FIFO Priority**
  
  If FIFO priority ordering was chosen, then the topic/queue acts like a priority queue. If two messages are assigned the same priority, then they are received in the order of their enqueue time.

- **Enqueue Time Followed by Priority**
  
  Messages with the same enqueue time are received according to their priorities. If the ordering criteria of two message is the same, then the order they are received is indeterminate. However, Oracle® Database does ensure that messages produced in one session with a particular ordering criteria are received in the order they were sent.
Specifying a Message Delay

Messages can be sent/published to a queue/topic with delay. The delay represents a time interval after which the message becomes available to the message consumer. A message specified with a delay is in a waiting state until the delay expires. Receiving by message identifier overrides the delay specification.

Delay is an Oracle® Database extension to JMS message properties. It requires the Oracle® Database background process queue monitor to be started.

Specifying a Message Expiration

Producers of messages can specify expiration limits, or TimeToLive for messages. This defines the period of time the message is available for a Message Consumer.

TimeToLive can be specified at send/publish time or using the set TimeToLive method of a MessageProducer, with the former overriding the latter. The Oracle® Database background process queue monitor must be running to implement TimeToLive.

Message Grouping

Messages belonging to a queue/topic can be grouped to form a set that can be consumed by only one consumer at a time. This requires the queue/topic be created in a queue table that is enabled for transactional message grouping. All messages belonging to a group must be created in the same transaction, and all messages created in one transaction belong to the same group.

Message grouping is an Oracle® Database extension to the JMS specification.

You can use this feature to divide a complex message into a linked series of simple messages. For example, an invoice directed to an invoices queue could be divided into a header message, followed by several messages representing details, followed by the trailer message.

Message grouping is also very useful if the message payload contains complex large objects such as images and video that can be segmented into smaller objects.

The priority, delay, and expiration properties for the messages in a group are determined solely by the message properties specified for the first message (head) of the group. Properties specified for subsequent messages in the group are ignored.

Message grouping is preserved during propagation. The destination topic must be enabled for transactional grouping.

See Also: "Dequeue Features" on page 1-21 for a discussion of restrictions you must keep in mind if message grouping is to be preserved while dequeuing messages from a queue enabled for transactional grouping.

JMS Message Consumer Features

This section contains these topics:

- Receiving Messages
- Message Navigation in Receive
- Browsing Messages
- Remove No Data
Receiving Messages

A JMS application can receive messages by creating a message consumer. Messages can be received synchronously using the `receive` call or asynchronously using a message listener.

There are three modes of receive:

- Block until a message arrives for a consumer
- Block for a maximum of the specified time
- Nonblocking

Message Navigation in Receive

If a consumer does not specify a navigation mode, then its first `receive` in a session retrieves the first message in the queue or topic, its second `receive` gets the next message, and so on. If a high priority message arrives for the consumer, then the consumer does not receive the message until it has cleared the messages that were already there before it.

To provide the consumer better control in navigating the queue for its messages, Oracle® Database offers several navigation modes as JMS extensions. These modes can be set at the `TopicSubscriber`, `QueueReceiver` or the `TopicReceiver`.

Two modes are available for ungrouped messages:

- **FIRST_MESSAGE**
  This mode resets the position to the beginning of the queue. It is useful for priority ordered queues, because it allows the consumer to remove the message on the top of the queue.

- **NEXT_MESSAGE**
  This mode gets whatever message follows the established position of the consumer. For example, a `NEXT_MESSAGE` applied when the position is at the fourth message will get the fifth message in the queue. This is the default action.

Three modes are available for grouped messages:

- **FIRST_MESSAGE**
  This mode resets the position to the beginning of the queue.

- **NEXT_MESSAGE**
  This mode sets the position to the next message in the same transaction.

- **NEXT_TRANSACTION**
  This mode sets the position to the first message in the next transaction.

The transaction grouping property can be negated if messages are received in the following ways:

- Receive by specifying a correlation identifier in the selector
- Receive by specifying a message identifier in the selector
■ Committing before all the messages of a transaction group have been received

If the consumer reaches the end of the queue while using the NEXT_MESSAGE or NEXT_TRANSACTION option, and you have specified a blocking receive(), then the navigating position is automatically changed to the beginning of the queue.

By default, a QueueReceiver, TopicReceiver, or TopicSubscriber uses FIRST_MESSAGE for the first receive call, and NEXT_MESSAGE for subsequent receive() calls.

**Browsing Messages**

Aside from the usual receive, which allows the dequeuing client to delete the message from the queue, JMS provides an interface that allows the JMS client to browse its messages in the queue. A QueueBrowser can be created using the createBrowser method from QueueSession.

If a message is browsed, then it remains available for further processing. That does not necessarily mean that the message will remain available to the JMS session after it is browsed, because a receive call from a concurrent session might remove it.

To prevent a viewed message from being removed by a concurrent JMS client, you can view the message in the locked mode. To do this, you must create a QueueBrowser with the locked mode using the Oracle® Database extension to the JMS interface. The lock on the message is released when the session performs a commit or a rollback.

To remove a message viewed by a QueueBrowser, the session must create a QueueReceiver and use the JMSmessageID as the selector.

**Remove No Data**

The consumer can remove a message from a queue or topic without retrieving it using the receiveNoData call. This is useful when the application has already examined the message, perhaps using a QueueBrowser. This mode allows the JMS client to avoid the overhead of retrieving a payload from the database, which can be substantial for a large message.

**Retry with Delay Interval**

If a transaction receiving a message from a queue/topic fails, then it is regarded as an unsuccessful attempt to remove the message. Oracle® Database records the number of failed attempts to remove the message in the message history.

An application can specify the maximum number of retries supported on messages at the queue/topic level. If the number of failed attempts to remove a message exceeds this maximum, then the message is moved to an exception queue.

Oracle® Database allows users to specify a retry_delay along with max_retries. This means that a message that has undergone a failed attempt at retrieving remains visible in the queue for dequeue after retry_delay interval. Until then it is in the WAITING state. The Oracle® Database background process time manager enforces the retry delay property.

The maximum retries and retry delay are properties of the queue/topic. They can be set when the queue/topic is created or by using the alter method on the queue/topic. The default value for MAX_RETRIES is 5.
Asynchronously Receiving Messages Using MessageListener

The JMS client can receive messages asynchronously by setting the MessageListener using the setMessageListener method.

When a message arrives for the consumer, the onMessage method of the message listener is invoked with the message. The message listener can commit or terminate the receipt of the message. The message listener does not receive messages if the JMS Connection has been stopped. The receive call must not be used to receive messages once the message listener has been set for the consumer.

The JMS client can receive messages asynchronously for all consumers in the session by setting the MessageListener at the session. No other mode for receiving messages must be used in the session once the message listener has been set.

Exception Queues

An exception queue is a repository for all expired or unserviceable messages. Applications cannot directly enqueue into exception queues. However, an application that intends to handle these expired or unserviceable messages can receive/remove them from the exception queue.

To retrieve messages from exception queues, the JMS client must use the point-to-point interface. The exception queue for messages intended for a topic must be created in a queue table with multiple consumers enabled. Like any other queue, the exception queue must be enabled for receiving messages using the start method in the AQOracleQueue class. You get an exception if you try to enable it for enqueue.

The exception queue is an Oracle-specific message property called "JMS_OracleExcpQ" that can be set with the message before sending/publishing it. If an exception queue is not specified, then the default exception queue is used. The default exception queue is automatically created when the queue table is created and is named AQ$_queue_table_name_E.

Messages are moved to the exception queue under the following conditions:

- The message was not dequeued within the specified timeToLive.
  For messages intended for more than one subscriber, the message is moved to the exception queue if one or more of the intended recipients is not able to dequeue the message within the specified timeToLive.

- The message was received successfully, but the application terminated the transaction that performed the receive because of an error while processing the message. The message is returned to the queue/topic and is available for any applications that are waiting to receive messages.
  A receive is considered rolled back or undone if the application terminates the entire transaction, or if it rolls back to a savepoint that was taken before the receive.
  Because this was a failed attempt to receive the message, its retry count is updated. If the retry count of the message exceeds the maximum value specified for the queue/topic where it resides, then it is moved to the exception queue.
  If a message has multiple subscribers, then the message is moved to the exception queue only when all the recipients of the message have exceeded the retry limit.
JMS Propagation

This section contains these topics:

- RemoteSubscriber
- Scheduling Propagation
- Enhanced Propagation Scheduling Capabilities
- Exception Handling During Propagation

RemoteSubscriber

Oracle® Database allows a subscriber at another database to subscribe to a topic. If a message published to the topic meets the criterion of the remote subscriber, then it is automatically propagated to the queue/topic at the remote database specified for the remote subscriber. Propagation is performed using database links and Oracle Net Services. This enables applications to communicate with each other without having to be connected to the same database.

There are two ways to implement remote subscribers:

- The createRemoteSubscriber method can be used to create a remote subscriber to/on the topic. The remote subscriber is specified as an instance of the class AQjmsAgent.
- The AQjmsAgent has a name and an address. The address consists of a queue/topic and the database link to the database of the subscriber.

There are two kinds of remote subscribers:

- The remote subscriber is a topic.
  This occurs when no name is specified for the remote subscriber in the AQjmsAgent object and the address is a topic. The message satisfying the subscriber's subscription is propagated to the remote topic. The propagated message is now available to all the subscriptions of the remote topic that it satisfies.

- A specific remote recipient is specified for the message.
  The remote subscription can be for a particular consumer at the remote database. If the name of the remote recipient is specified (in the AQjmsAgent object), then the message satisfying the subscription is propagated to the remote database for that recipient only. The recipient at the remote database uses the TopicReceiver interface to retrieve its messages. The remote subscription can also be for a point-to-point queue.

Scheduling Propagation

Propagation must be scheduled using the schedule_propagation method for every topic from which messages are propagated to target destination databases.

A schedule indicates the time frame during which messages can be propagated from the source topic. This time frame can depend on a number of factors such as network...
traffic, the load at the source database, the load at the destination database, and so on. The schedule therefore must be tailored for the specific source and destination. When a schedule is created, a job is automatically submitted to the job_queue facility to handle propagation.

The administrative calls for propagation scheduling provide great flexibility for managing the schedules. The duration or propagation window parameter of a schedule specifies the time frame during which propagation must take place. If the duration is unspecified, then the time frame is an infinite single window. If a window must be repeated periodically, then a finite duration is specified along with a next_time function that defines the periodic interval between successive windows.

**See Also:** "Scheduling a Propagation" on page 12-18

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue. In addition there are calls for temporarily disabling a schedule (instead of dropping the schedule) and enabling a disabled schedule. A schedule is active when messages are being propagated in that schedule. All the administrative calls can be made irrespective of whether the schedule is active or not. If a schedule is active, then it takes a few seconds for the calls to be executed.

Job queue processes must be started for propagation to take place. At least 2 job queue processes must be started. The database links to the destination database must also be valid. The source and destination topics of the propagation must be of the same message type. The remote topic must be enabled for enqueue. The user of the database link must also have enqueue privileges to the remote topic.

### Enhanced Propagation Scheduling Capabilities

Catalog views defined for propagation provide the following information about active schedules:

- Name of the background process handling the schedule
- SID (session and serial number) for the session handling the propagation
- Instance handling a schedule (if using RAC)
- Previous successful execution of a schedule
- Next planned execution of a schedule

The following propagation statistics are maintained for each schedule, providing useful information to queue administrators for tuning:

- The total number of messages propagated in a schedule
- Total number of bytes propagated in a schedule
- Maximum number of messages propagated in a window
- Maximum number of bytes propagated in a window
- Average number of messages propagated in a window
- Average size of propagated messages
- Average time to propagated a message

Propagation has built-in support for handling failures and reporting errors. For example, if the database link specified is invalid, or if the remote database is unavailable, or if the remote topic/queue is not enabled for enqueuing, then the appropriate error message is reported. Propagation uses an exponential backoff.
scheme for retrying propagation from a schedule that encountered a failure. If a schedule continuously encounters failures, then the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, then the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts are made after which the schedule is automatically disabled.

**Note:** Once a retry attempt slips to the next propagation window, it will always do so; the exponential backoff scheme no longer governs retry scheduling. If the date function specified in the `next_time` parameter of `DBMS_AQADM.SCHEDULE_PROPAGATION()` results in a short interval between windows, then the number of unsuccessful retry attempts can quickly reach 16, disabling the schedule.

When a schedule is disabled automatically due to failures, the relevant information is written into the alert log. It is possible to check at any time if there were failures encountered by a schedule and if so how many successive failures were encountered, the error message indicating the cause for the failure and the time at which the last failure was encountered. By examining this information, an administrator can fix the failure and enable the schedule.

If propagation is successful during a retry, then the number of failures is reset to 0.

Propagation has built-in support for Real Application Clusters and is transparent to the user and the administrator. The job that handles propagation is submitted to the same instance as the owner of the queue table where the source topic resides. If at any time there is a failure at an instance and the queue table that stores the topic is migrated to a different instance, then the propagation job is also automatically migrated to the new instance. This minimizes the pinging between instances and thus offers better performance. Propagation has been designed to handle any number of concurrent schedules.

The number of `job_queue_processes` is limited to a maximum of 1000 and some of these can be used to handle jobs unrelated to propagation. Hence, propagation has built in support for multitasking and load balancing. The propagation algorithms are designed such that multiple schedules can be handled by a single snapshot (`job_queue`) process. The propagation load on a `job_queue` processes can be skewed based on the arrival rate of messages in the different source topics. If one process is overburdened with several active schedules while another is less loaded with many passive schedules, then propagation automatically redistributes the schedules among the processes such that they are loaded uniformly.

**Exception Handling During Propagation**

When a system error such as a network failure occurs, Oracle® Database continues to attempt to propagate messages using an exponential back-off algorithm. In some situations that indicate application errors in queue-to-dblink propagations, Oracle® Database marks messages as `UNDELIVERABLE` and logs a message in `alert.log`. Examples of such errors are when the remote queue does not exist or when there is a type mismatch between the source queue and the remote queue. The trace files in the `background_dump_dest` directory can provide additional information about the error.

When a new job queue process starts, it clears the mismatched type errors so the types can be reverified. If you have capped the number of job queue processes and propagation remains busy, then you might not want to wait for the job queue process
to terminate and restart. Queue types can be reverified at any time using DBMS_AQADM.VERIFY_QUEUE_TYPES.

---

**Note:** When a type mismatch is detected in queue-to-queue propagation, propagation stops and throws an error. In such situations you must query the DBA_SCHEDULES view to determine the last error that occurred during propagation to a particular destination. The message is not marked as UNDELIVERABLE.

---

**Message Transformation with JMS AQ**

A **transformation** can be defined to map messages of one format to another. Transformations are useful when applications that use different formats to represent the same information must be integrated. Transformations can be SQL expressions and PL/SQL functions. Message transformation is an Oracle® Database extension to the standard JMS interface.

The transformations can be created using the DBMS_TRANSFORM.create_transformation procedure. Transformation can be specified for the following operations:

- Sending a message to a queue or topic
- Receiving a message from a queue or topic
- Creating a TopicSubscriber
- Creating a RemoteSubscriber. This enables propagation of messages between topics of different formats.

---

**J2EE Compliance**

In Oracle Database 10g, Oracle JMS conforms to the Sun Microsystems JMS 1.1 standard. You can define the J2EE compliance mode for an Oracle Java Message Service (OJMS) client at runtime. For compliance, set the Java property oracle.jms.j2eeCompliant to TRUE as a command line option. For noncompliance, do nothing. FALSE is the default value.

Features in Oracle® Database that support J2EE compliance (and are also available in the noncompliant mode) include:

- Nontransactional sessions
- Durable subscribers
- Temporary queues and topics
- Nonpersistent delivery mode
- Multiple JMS messages types on a single JMS queue or topic (using Oracle® Database queues of the AQ$_JMS_MESSAGE type)
- The noLocal option for durable subscribers
See Also:


- "JMS Message Headers" on page 11-9 for information on how the Java property `oracle.jms.j2eeCompliant` affects `JMSPriority` and `JMSExpiration`

- "DurableSubscriber" on page 11-18 for information on how the Java property `oracle.jms.j2eeCompliant` affects durable subscribers
This chapter describes the basic operational **Java Message Service** (JMS) administrative interface to Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:
- EXECUTE Privilege on DBMS_AQIN
- Registering a ConnectionFactory
- Unregistering a Queue/Topic ConnectionFactory
- Getting a QueueConnectionFactory or TopicConnectionFactory
- Getting a Queue or Topic in LDAP
- Creating a Queue Table
- Getting a Queue Table
- Creating a Queue
- Granting and Revoking Privileges
- Managing Destinations
- Propagation Schedules

**EXECUTE Privilege on DBMS_AQIN**

Users should never directly call methods in the DBMS_AQIN package, but they do need the EXECUTE privilege on DBMS_AQIN. Use the following syntax to accomplish this:

```
GRANT EXECUTE ON DBMS_AQIN to user;
```

**Registering a ConnectionFactory**

You can register a **ConnectionFactory** four ways:
- Registering Through the Database Using JDBC Connection Parameters
- Registering Through the Database Using a JDBC URL
- Registering Through LDAP Using JDBC Connection Parameters
- Registering Through LDAP Using a JDBC URL

**Registering Through the Database Using JDBC Connection Parameters**

```java
public static int registerConnectionFactory(java.sql.Connection connection, java.lang.String conn_name, ...
```

Oracle JMS Basic Operations 12-1
This method registers a QueueConnectionFactory or TopicConnectionFactory through the database to a Lightweight Directory Access Protocol (LDAP) server with JDBC connection parameters. This method is static and has the following parameters:

**Example 12–1 Registering Through the Database Using JDBC Connection Parameters**

```java
String url;
java.sql.Connection db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.registerConnectionFactory(
    db_conn, "queue_conn1", "sun-123", "db1", 1521, "thin", "queue");
```

### Registering Through the Database Using a JDBC URL

```java
public static int registerConnectionFactory(java.sql.Connection connection,
java.lang.String conn_name,
java.lang.String jdbc_url,
java.util.Properties info,
java.lang.String type)
throws JMSException
```

This method registers a QueueConnectionFactory or TopicConnectionFactory through the database with a JDBC URL to LDAP. It is static and has the following parameters:

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>JDBC connection used in registration</td>
</tr>
<tr>
<td>conn_name</td>
<td>Name of the connection to be registered</td>
</tr>
<tr>
<td>jdbc_url</td>
<td>URL to connect to</td>
</tr>
<tr>
<td>type</td>
<td>Connection factory type (QUEUE or TOPIC)</td>
</tr>
</tbody>
</table>
Registering a ConnectionFactory

Oracle JMS Basic Operations

12-3

The database connection passed to registerConnectionFactory must be granted AQ_ADMINISTRATOR_ROLE. After registration, you can look up the connection factory using JNDI.

Example 12–2 Registering Through the Database Using a JDBC URL

```java
String url;
java.sql.connection db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.registerConnectionFactory(
        db_conn, "topic_conn1", url, null, "topic");
```

Registering Through LDAP Using JDBC Connection Parameters

```java
public static int registerConnectionFactory(java.util.Hashtable env,
java.lang.String conn_name,
java.lang.String hostname,
java.lang.String oracle_sid,
int portno,
java.lang.String driver,
java.lang.String type)
throws JMSException
```

This method registers a QueueConnectionFactory or TopicConnectionFactory through LDAP with JDBC connection parameters to LDAP. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>env</td>
<td>Environment of LDAP connection</td>
</tr>
<tr>
<td>conn_name</td>
<td>Name of the connection to be registered</td>
</tr>
<tr>
<td>hostname</td>
<td>Name of the host running Oracle® Database</td>
</tr>
<tr>
<td>oracle_sid</td>
<td>Oracle system identifier</td>
</tr>
<tr>
<td>portno</td>
<td>Port number</td>
</tr>
<tr>
<td>driver</td>
<td>JDBC driver type</td>
</tr>
<tr>
<td>type</td>
<td>Connection factory type (QUEUE or TOPIC)</td>
</tr>
</tbody>
</table>

The hash table passed to registerConnectionFactory() must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted GLOBAL_AQ_USER_ROLE). After registration, look up the connection factory using JNDI.

Example 12–3 Registering Through LDAP Using JDBC Connection Parameters

```java
Hashtable env = new Hashtable(5, 0.75f);
```
Registering a ConnectionFactory

/* the following statements set in hashtable env:
   * service provider package
   * the URL of the ldap server
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put("searchbase", "cn=db1,cn=Oraclecontext,cn=acme,cn=com");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aqadmin,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");

AQjmsFactory.registerConnectionFactory(env,
   "queue_conn1",
   "sun-123",
   "db1",
   1521,
   "thin",
   "queue");

Registering Through LDAP Using a JDBC URL

public static int registerConnectionFactory(java.util.Hashtable env,
   java.lang.String conn_name,
   java.lang.String jdbc_url,
   java.util.Properties info,
   java.lang.String type)
   throws JMSException

This method registers a QueueConnectionFactory or TopicConnectionFactory through LDAP with JDBC connection parameters to LDAP. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>env</td>
<td>Environment of LDAP connection</td>
</tr>
<tr>
<td>conn_name</td>
<td>Name of the connection to be registered</td>
</tr>
<tr>
<td>jdbc_url</td>
<td>URL to connect to</td>
</tr>
<tr>
<td>info</td>
<td>Properties information</td>
</tr>
<tr>
<td>type</td>
<td>Connection factory type (QUEUE or TOPIC)</td>
</tr>
</tbody>
</table>

The hash table passed to registerConnectionFactory() must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted GLOBAL_AQ_USER_ROLE). After registration, look up the connection factory using JNDI.

Example 12-4  Registering Through LDAP Using a JDBC URL

```java
String url;
Hashtable env = new Hashtable(5, 0.75f);

/* the following statements set in hashtable env:
   * service provider package
```
You can unregister a queue/topic ConnectionFactory in LDAP two ways:

- **Unregistering Through the Database**
- **Unregistering Through LDAP**

### Unregistering Through the Database

```java
public static int unregisterConnectionFactory(java.sql.Connection connection, java.lang.String conn_name) throws JMSException
```

This method unregisters a QueueConnectionFactory or TopicConnectionFactory in LDAP. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>JDBC connection used in registration</td>
</tr>
<tr>
<td>conn_name</td>
<td>Name of the connection to be registered</td>
</tr>
</tbody>
</table>

The database connection passed to `unregisterConnectionFactory()` must be granted `AQ_ADMINISTRATOR_ROLE`.

**Example 12–5  Unregistering Through the Database**

```java
String url;
java.sql.Connection db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.unregisterConnectionFactory(db_conn, "topic_conn1");
```

### Unregistering Through LDAP

```java
public static int unregisterConnectionFactory(java.util.Hashtable env, java.lang.String conn_name) throws JMSException
```

This method unregisters a QueueConnectionFactory or TopicConnectionFactory in LDAP. It is static and has the following parameters:
The hash table passed to unregisterConnectionFactory() must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted GLOBAL_AQ_USER_ROLE).

**Example 12–6 Unregistering Through LDAP**

```
Hashtable env = new Hashtable(5, 0.75f);
/* the following statements set in hashtable env:
 * service provider package
 * the distinguished name of the database server
 * the authentication method (simple)
 * the LDAP username
 * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put("searchbase", "cn=db1,cn=Oraclecontext,cn=acme,cn=com");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aqadmin,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
url = "jdbc:oracle:thin:@sun-123:1521:db1";
AQjmsFactory.unregisterConnectionFactory(env, "queue_conn1");
```

### Getting a QueueConnectionFactory or TopicConnectionFactory

This section contains these topics:

- Getting a QueueConnectionFactory with JDBC URL
- Getting a QueueConnectionFactory with JDBC Connection Parameters
- Getting a TopicConnectionFactory with JDBC URL
- Getting a TopicConnectionFactory with JDBC Connection Parameters
- Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

#### Getting a QueueConnectionFactory with JDBC URL

```java
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
    java.lang.String jdbc_url,
    java.util.Properties info)
throws JMSException
```

This method gets a QueueConnectionFactory with JDBC URL. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc_url</td>
<td>URL to connect to</td>
</tr>
</tbody>
</table>
Getting a QueueConnectionFactory or TopicConnectionFactory

Example 12–7  Getting a QueueConnectionFactory with JDBC URL

```java
String url = "jdbc:oracle:oci10:internal/oracle"
Properties info = new Properties();
QueueConnectionFactory qc_fact;

info.put("internal_logon", "sysdba");
qc_fact = AQjmsFactory.getQueueConnectionFactory(url, info);
```

Getting a QueueConnectionFactory with JDBC Connection Parameters

```java
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
    java.lang.String hostname,
    java.lang.String oracle_sid,
    int portno,
    java.lang.String driver)
throws JMSException
```

This method gets a QueueConnectionFactory with JDBC connection parameters. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>Name of the host running Oracle® Database</td>
</tr>
<tr>
<td>oracle_sid</td>
<td>Oracle system identifier</td>
</tr>
<tr>
<td>portno</td>
<td>Port number</td>
</tr>
<tr>
<td>driver</td>
<td>JDBC driver type</td>
</tr>
</tbody>
</table>

Example 12–8  Getting a QueueConnectionFactory with JDBC Connection Parameters

```java
String host = "dlsun";
String ora_sid = "rdbms10i"
String driver = "thin";
int port = 5521;
QueueConnectionFactory qc_fact;

qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid, port, driver);
```

Getting a TopicConnectionFactory with JDBC URL

```java
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
    java.lang.String jdbc_url,
    java.util.Properties info)
throws JMSException
```

This method gets a TopicConnectionFactory with a JDBC URL. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc_url</td>
<td>URL to connect to</td>
</tr>
<tr>
<td>info</td>
<td>Properties information</td>
</tr>
</tbody>
</table>
Example 12–9  Getting a TopicConnectionFactory with JDBC URL

```java
String url = "jdbc:oracle:oci110:internal/oracle"
Properties info = new Properties();
TopicConnectionFactory tc_fact;

info.put("internal_logon", "sysdba");
tc_fact = AQjmsFactory.getTopicConnectionFactory(url, info);
```

Getting a TopicConnectionFactory with JDBC Connection Parameters

```java
public static javax.jms.TopicConnectionFactory getTopicConnectionFactory(
    java.lang.String hostname,
    java.lang.String oracle_sid,
    int portno,
    java.lang.String driver)
throws JMSException
```

This method gets a TopicConnectionFactory with JDBC connection parameters. It is static and has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>Name of the host running Oracle® Database</td>
</tr>
<tr>
<td>oracle_sid</td>
<td>Oracle system identifier</td>
</tr>
<tr>
<td>portno</td>
<td>Port number</td>
</tr>
<tr>
<td>driver</td>
<td>JDBC driver type</td>
</tr>
</tbody>
</table>

Example 12–10  Getting a TopicConnectionFactory with JDBC Connection Parameters

```java
String host = "dlsun";
String ora_sid = "rdbms10i"
String driver = "thin";
int port = 5521;
TopicConnectionFactory tc_fact;

tc_fact = AQjmsFactory.getTopicConnectionFactory(host, ora_sid, port, driver);
```

Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

This method gets a QueueConnectionFactory or TopicConnectionFactory from LDAP.

Example 12–11  Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

```java
Hashtable env = new Hashtable(5, 0.75f);
DirContext ctx;
queueConnectionFactory qc_fact;

/* the following statements set in hashtable env: *
* service provider package *
* the URL of the ldap server *
* the distinguished name of the database server *
* the authentication method (simple) *
* the LDAP username *
* the LDAP user password *
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
```
Creating a Queue Table

public oracle.AQ.AQQueueTable createQueueTable(
    java.lang.String owner,
    java.lang.String name,
    oracle.AQ.AQQueueTableProperty property)
throws JMSException

This method creates a queue table. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Queue table owner (schema)</td>
</tr>
<tr>
<td>name</td>
<td>Queue table name</td>
</tr>
<tr>
<td>property</td>
<td>Queue table properties</td>
</tr>
</tbody>
</table>
If the queue table is used to hold queues, then the queue table must not be multiconsumer enabled (default). If the queue table is used to hold topics, then the queue table must be multiconsumer enabled.

**CLOB, BLOB, and BFILE objects** are valid attributes for an Oracle® Database object type load. However, only CLOB and BLOB can be propagated using Oracle® Database propagation in Oracle8i and after.

### Example 12–13  Creating a Queue Table

```java
QueueSession q_sess    = null;
AQQueueTable q_table   = null;
AQQueueTableProperty qt_prop   = null;

qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");
q_table = ((AQjmsSession)q_sess).createQueueTable( 
    "boluser", "bol_ship_queue_table", qt_prop);
```

### Getting a Queue Table

This method gets a queue table. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Queue table owner (schema)</td>
</tr>
<tr>
<td>name</td>
<td>Queue table name</td>
</tr>
</tbody>
</table>

If the caller that opened the connection is not the owner of the queue table, then the caller must have Oracle® Database enqueue/dequeue privileges on queues/topics in the queue table. Otherwise the queue table is not returned.

### Example 12–14  Getting a Queue Table

```java
QueueSession q_sess;
AQQueueTable q_table;

q_table = ((AQjmsSession)q_sess).getQueueTable( 
    "boluser", "bol_ship_queue_table");
```

### Creating a Queue

This section contains these topics:

- Creating a Point-to-Point Queue
- Creating a Publish/Subscribe Topic

### Creating a Point-to-Point Queue

```java
public javax.jms.Queue createQueue(
    oracle.AQ.AQQueueTable q_table,
    java.lang.String queue_name,
    oracle.jms.AQjmsDestinationProperty dest_property) 
    throws JMSException
```
This method creates a queue in a specified queue table. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_table</td>
<td>Queue table in which the queue is to be created. The queue table must be single-consumer.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Name of the queue to be created</td>
</tr>
<tr>
<td>dest_property</td>
<td>Queue properties</td>
</tr>
</tbody>
</table>

This method is specific to OJMS. You cannot use standard Java `javax.jms.Session` objects with it. Instead, you must cast the standard type to the OJMS concrete class `oracle.jms.AQjmsSession`.

Example 12–15  Creating a Point-to-Point Queue

```java
QueueSession q_sess;
AQQueueTable q_table;
AQjmsDestinationProperty dest_prop;
Queue queue;
queue = ((AQjmsSession)q_sess).createQueue(q_table, "jms_q1", dest_prop);
```

Creating a Publish/Subscribe Topic

```java
public javax.jms.Topic createTopic(
    oracle.AQ.AQQueueTable q_table,
    java.lang.String topic_name,
    oracle.jms.AQjmsDestinationProperty dest_property)
    throws JMSException
```

This method creates a topic in the **publish/subscribe** model. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_table</td>
<td>Queue table in which the queue is to be created. The queue table must be multiconsumer.</td>
</tr>
<tr>
<td>queue_name</td>
<td>Name of the queue to be created</td>
</tr>
<tr>
<td>dest_property</td>
<td>Queue properties</td>
</tr>
</tbody>
</table>

This method is specific to OJMS. You cannot use standard Java `javax.jms.Session` objects with it. Instead, you must cast the standard type to the OJMS concrete class `oracle.jms.AQjmsSession`.

Example 12–16  Creating a Publish/Subscribe Topic

```java
TopicSession t_sess;
AQQueueTable q_table;
AQjmsDestinationProperty dest_prop;
Topic topic;
topic = ((AQjmsSession)t_sess).createTopic(q_table, "jms_t1", dest_prop);
```
In Example 12–17, if an order cannot be filled because of insufficient inventory, then the transaction processing the order is terminated. The bookedorders topic is set up with `max_retries = 4` and `retry_delay = 12` hours. Thus, if an order is not filled up in two days, then it is moved to an exception queue.

**Example 12–17  Specifying Max Retries and Max Delays in Messages**

```java
class BookOrder {

    public BolOrder process_booked_order(TopicSession jms_session) {
        try {
            Topic topic;
            TopicSubscriber tsubs;
            ObjectMessage obj_message;
            BolCustomer customer;
            BolOrder booked_order = null;
            String country;
            int i = 0;

            /* get a handle to the OE_bookedorders_topic */
            topic = ((AQjmsSession)jms_session).getTopic("WS",
                    "WS_bookedorders_topic");

            /* Create local subscriber - to track messages for Western Region */
            tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
                    "Region = 'Western' ",
                    false);

            /* wait for a message to show up in the topic */
            obj_message = (ObjectMessage)tsubs.receive(10);

            booked_order = (BolOrder)obj_message.getObject();
            customer = booked_order.getCustomer();
            country = customer.getCountry();

            if (country == "US")
                jms_session.commit();
            else
                jms_session.rollback();
        }
    }catch (JMSException ex)
    {
        System.out.println("Exception " + ex);}

    return booked_order;
}
```

**Granting and Revoking Privileges**

This section contains these topics:

- Granting Oracle® Database System Privileges
- Revoking Oracle® Database System Privileges
- Granting Publish/Subscribe Topic Privileges
Granting and Revoking Privileges

- Revoking Publish/Subscribe Topic Privileges
- Granting Point-to-Point Queue Privileges
- Revoking Point-to-Point Queue Privileges

Granting Oracle® Database System Privileges

```java
public void grantSystemPrivilege(java.lang.String privilege,
                                 java.lang.String grantee,
                                 boolean admin_option)
    throws JMSException
```

This method grants Oracle® Database system privileges to a user or role.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>ENQUEUE_ANY, DEQUEUE_ANY or MANAGE_ANY</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee (user, role, or PUBLIC)</td>
</tr>
<tr>
<td>admin_option</td>
<td>If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles</td>
</tr>
</tbody>
</table>

Initially only SYS and SYSTEM can use this procedure successfully. Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to dequeue messages from any queues in the database. Users granted the MANAGE_ANY privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

**Example 12–18  Granting Oracle® Database System Privileges**

```java
TopicSession             t_sess;
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "scott", false);
```

Revoking Oracle® Database System Privileges

```java
public void revokeSystemPrivilege(java.lang.String privilege,
                                   java.lang.String grantee)
    throws JMSException
```

This method revokes Oracle® Database system privileges from a user or role. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>ENQUEUE_ANY, DEQUEUE_ANY or MANAGE_ANY</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee (user, role, or PUBLIC)</td>
</tr>
</tbody>
</table>

Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to dequeue messages from any queues in the database. Users granted the MANAGE_ANY privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

**Example 12–19  Revoking Oracle® Database System Privileges**

```java
TopicSession             t_sess;
```

Parameter Description
```
((AQjmsSession)t_sess).revokeSystemPrivilege("ENQUEUE_ANY", "scott");

**Granting Publish/Subscribe Topic Privileges**

```java
public void grantTopicPrivilege(javax.jms.Session session,
                                java.lang.String privilege,
                                java.lang.String grantee,
                                boolean grant_option)
throws JMSException
```

This method grants a topic privilege in the publish/subscribe model. Initially only the queue table owner can use this procedure to grant privileges on the topic. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>privilege</td>
<td>ENQUEUE, DEQUEUE, or ALL (ALL means both.)</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee (user, role, or PUBLIC)</td>
</tr>
<tr>
<td>grant_option</td>
<td>If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles</td>
</tr>
</tbody>
</table>

**Example 12–20  Granting Publish/Subscribe Topic Privileges**

```java
TopicSession             t_sess;
Topic                    topic;
((AQjmsDestination)topic).grantTopicPrivilege(
  t_sess, "ENQUEUE", "scott", false);
```

**Revoking Publish/Subscribe Topic Privileges**

```java
public void revokeTopicPrivilege(javax.jms.Session session,
                                  java.lang.String privilege,
                                  java.lang.String grantee)
throws JMSException
```

This method revokes a topic privilege in the publish/subscribe model. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>privilege</td>
<td>ENQUEUE, DEQUEUE, or ALL (ALL means both.)</td>
</tr>
<tr>
<td>grantee</td>
<td>Revoked grantee (user, role, or PUBLIC)</td>
</tr>
</tbody>
</table>

**Example 12–21  Revoking Publish/Subscribe Topic Privileges**

```java
TopicSession             t_sess;
Topic                    topic;
((AQjmsDestination)topic).revokeTopicPrivilege(t_sess, "ENQUEUE", "scott");
```

**Granting Point-to-Point Queue Privileges**

```java
public void grantQueuePrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee,
                                 boolean grant_option)
throws JMSException
```
Granting and Revoking Privileges

```java
java.lang.String privilege,
java.lang.String grantee,
boolean grant_option)
throws JMSException
```

This method grants a queue privilege in the point-to-point model. Initially only the queue table owner can use this procedure to grant privileges on the queue. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>privilege</td>
<td>ENQUEUE, DEQUEUE, or ALL (ALL means both.)</td>
</tr>
<tr>
<td>grantee</td>
<td>Grantee (user, role, or PUBLIC)</td>
</tr>
<tr>
<td>grant_option</td>
<td>If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles</td>
</tr>
</tbody>
</table>

**Example 12–22  Granting Point-to-Point Queue Privileges**

```
QueueSession q_sess;
Queue queue;

((AQjmsDestination)queue).grantQueuePrivilege(
  q_sess, "ENQUEUE", "scott", false);
```

Revoking Point-to-Point Queue Privileges

```java
public void revokeQueuePrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee)
throws JMSException
```

This method revokes queue privileges in the point-to-point model. Initially only the queue table owner can use this procedure to grant privileges on the queue. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>privilege</td>
<td>ENQUEUE, DEQUEUE, or ALL (ALL means both.)</td>
</tr>
<tr>
<td>grantee</td>
<td>Revoked grantee (user, role, or PUBLIC)</td>
</tr>
</tbody>
</table>

To revoke a privilege, the revoker must be the original grantor of the privilege. Privileges propagated through the GRANT option are revoked if the grantor privilege is also revoked.

**Example 12–23  Revoking Point-to-Point Queue Privileges**

```
QueueSession q_sess;
Queue queue;

((AQjmsDestination)queue).revokeQueuePrivilege(q_sess, "ENQUEUE", "scott");
```
Managing Destinations

This section contains these topics:

- Starting a Destination
- Stopping a Destination
- Altering a Destination
- Dropping a Destination

Starting a Destination

```java
class (AQjmsDestination) {
    public void start(javax.jms.Session session,
                      boolean enqueue,
                      boolean dequeue)
        throws JMSException
    }
```

This method starts a destination. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>enqueue</td>
<td>If set to TRUE, then enqueue is enabled</td>
</tr>
<tr>
<td>dequeue</td>
<td>If set to TRUE, then dequeue is enabled</td>
</tr>
</tbody>
</table>

Example 12–24 Starting a Destination

```java
TopicSession t_sess;
QueueSession q_sess;
Topic topic;
Queue queue;

(AQjmsDestination) topic.start(t_sess, true, true);
(AQjmsDestination) queue.start(q_sess, true, true);
```

Stopping a Destination

```java
class (AQjmsDestination) {
    public void stop(javax.jms.Session session,
                     boolean enqueue,
                     boolean dequeue,
                     boolean wait)
        throws JMSException
    }
```

This method stops a destination. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>enqueue</td>
<td>If set to TRUE, then enqueue is disabled</td>
</tr>
<tr>
<td>dequeue</td>
<td>If set to TRUE, then dequeue is disabled</td>
</tr>
<tr>
<td>wait</td>
<td>If set to true, then pending transactions on the queue/topic are allowed to complete before the destination is stopped</td>
</tr>
</tbody>
</table>

Example 12–25 Stopping a Destination

```java
TopicSession t_sess;
```
Altering a Destination

```java
public void alter(javax.jms.Session session,
                     oracle.jms.AQjmsDestinationProperty dest_property)
    throws JMSException
```

This method alters a destination. It has the following properties:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>dest_property</td>
<td>New properties of the queue or topic</td>
</tr>
</tbody>
</table>

**Example 12–26  Altering a Destination**

```java
QueueSession q_sess;
Queue   queue;
TopicSession t_sess;
Topic   topic;
AQjmsDestinationProperty dest_prop1, dest_prop2;

((AQjmsDestination)queue).alter(dest_prop1);
((AQjmsDestination)topic).alter(dest_prop2);
```

Dropping a Destination

```java
public void drop(javax.jms.Session session)
    throws JMSException
```

This method drops a destination. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
</tbody>
</table>

**Example 12–27  Dropping a Destination**

```java
QueueSession q_sess;
Queue   queue;
TopicSession t_sess;
Topic   topic;

((AQjmsDestination)queue).drop(q_sess);
((AQjmsDestination)topic).drop(t_sess);
```

Propagation Schedules

This section contains these topics:

- Scheduling a Propagation
- Enabling a Propagation Schedule
- Altering a Propagation Schedule
- Disabling a Propagation Schedule
Scheduling a Propagation

```java
public void schedulePropagation(javax.jms.Session session,
                                java.lang.String destination,
                                java.util.Date start_time,
                                java.lang.Double duration,
                                java.lang.String next_time,
                                java.lang.Double latency)
    throws JMSException
```

This method schedules a propagation. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>destination</td>
<td>Database link of the remote database for which propagation is being scheduled. A null string means that propagation is scheduled for all subscribers in the database of the topic.</td>
</tr>
<tr>
<td>start_time</td>
<td>Time propagation starts</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of propagation</td>
</tr>
<tr>
<td>next_time</td>
<td>Next time propagation starts</td>
</tr>
<tr>
<td>latency</td>
<td>Latency in seconds that can be tolerated. Latency is the difference between the time a message was enqueued and the time it was propagated.</td>
</tr>
</tbody>
</table>

If a message has multiple recipients at the same destination in either the same or different queues, then it is propagated to all of them at the same time.

**Example 12–28  Scheduling a Propagation**

```java
TopicSession t_sess;
Topic topic;

((AQjmsDestination)topic).schedulePropagation(
    t_sess, null, null, null, null, new Double(0));
```

Enabling a Propagation Schedule

```java
public void enablePropagationSchedule(javax.jms.Session session,
                                      java.lang.String destination)
    throws JMSException
```

This method enables a propagation schedule. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>destination</td>
<td>Database link of the destination database. A null string means that propagation is to the local database.</td>
</tr>
</tbody>
</table>

**Example 12–29  Enabling a Propagation Schedule**

```java
TopicSession t_sess;
Topic topic;
```
Altering a Propagation Schedule

```java
public void alterPropagationSchedule(javax.jms.Session session,
        java.lang.String destination,
        java.lang.Double duration,
        java.lang.String next_time,
        java.lang.Double latency)
    throws JMSException
```

This method alters a propagation schedule. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>destination</td>
<td>Database link of the remote database for which propagation is being scheduled. A null string means that propagation is scheduled for all subscribers in the database of the topic.</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of propagation</td>
</tr>
<tr>
<td>next_time</td>
<td>Next time propagation starts</td>
</tr>
<tr>
<td>latency</td>
<td>Latency in seconds that can be tolerated. Latency is the difference between the time a message was enqueued and the time it was propagated.</td>
</tr>
</tbody>
</table>

**Example 12–30  Altering a Propagation Schedule**

```java
TopicSession t_sess;
Topic topic;

((AQjmsDestination)topic).alterPropagationSchedule(
    t_sess, null, 30, null, new Double(30));
```

Disabling a Propagation Schedule

```java
public void disablePropagationSchedule(javax.jms.Session session,
        java.lang.String destination)
    throws JMSException
```

This method disables a propagation schedule. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>destination</td>
<td>Database link of the destination database. A null string means that propagation is to the local database.</td>
</tr>
</tbody>
</table>

**Example 12–31  Disabling a Propagation Schedule**

```java
TopicSession t_sess;
Topic topic;

((AQjmsDestination)topic).disablePropagationSchedule(t_sess, "dbs1");
```
Unscheduling a Propagation

public void unschedulePropagation(javax.jms.Session session,
       java.lang.String destination)
       throws JMSException

This method unschedules a previously scheduled propagation. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>JMS session</td>
</tr>
<tr>
<td>destination</td>
<td>Database link of the destination database. A null string means that propagation is to the local database.</td>
</tr>
</tbody>
</table>

Example 12–32 Unscheduling a Propagation

TopicSession t_sess;
Topic topic;

((AQ.jmsDestination)topic).unschedulePropagation(t_sess, "dbs1");
This chapter describes the components of the Oracle Streams Advanced Queuing (AQ) Java Message Service (JMS) operational interface that are specific to point-to-point operations. Components that are shared by point-to-point and publish/subscribe are described in Chapter 15, "Oracle JMS Shared Interfaces".

This chapter contains these topics:

- Creating a Connection with Username/Password
- Creating a Connection with Default ConnectionFactory Parameters
- Creating a QueueConnection with Username/Password
- Creating a QueueConnection with an Open JDBC Connection
- Creating a QueueConnection with Default ConnectionFactory Parameters
- Creating a QueueConnection with an Open OracleOCIConnectionPool
- Creating a Session
- Creating a QueueSession
- Creating a QueueSender
- Sending Messages Using a QueueSender with Default Send Options
- Sending Messages Using a QueueSender by Specifying Send Options
- Creating a QueueBrowser for Standard JMS Type Messages
- Creating a QueueBrowser for Standard JMS Type Messages, Locking Messages
- Creating a QueueBrowser for Oracle Object Type Messages
- Creating a QueueBrowser for Oracle Object Type Messages, Locking Messages
- Creating a QueueReceiver for Standard JMS Type Messages
- Creating a QueueReceiver for Oracle Object Type Messages

Creating a Connection with Username/Password

```java
public javax.jms.Connection createConnection(
    java.lang.String username,
    java.lang.String password)
    throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with the specified username and password. This method is new and supports JMS version 1.1 specifications. It has the following parameters:
Creating a Connection with Default ConnectionFactory Parameters

```java
public javax.jms.Connection createConnection()
    throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with default `ConnectionFactory` parameters. This method is new and supports JMS version 1.1 specifications. If the `ConnectionFactory` properties do not contain a default username and password, then it throws a `JMSException`.

Creating a QueueConnection with Username/Password

```java
public javax.jms.QueueConnection createQueueConnection(
    java.lang.String username,
    java.lang.String password)
    throws JMSException
```

This method creates a queue connection with the specified username and password. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user connecting to the database for queuing</td>
</tr>
<tr>
<td>password</td>
<td>Password for creating the connection to the server</td>
</tr>
</tbody>
</table>

**Example 13–1 Creating a QueueConnection with Username/Password**

```java
QueueConnectionFactory qc_fact = AQjmsFactory.getQueueConnectionFactory("sun123", "oratest", 5521, "thin");
QueueConnection qc_conn = qc_fact.createQueueConnection("jmsuser", "jmsuser");
```

Creating a QueueConnection with an Open JDBC Connection

```java
public static javax.jms.QueueConnection createQueueConnection(
    java.sql.Connection jdbc_connection)
    throws JMSException
```

This method creates a queue connection with an open JDBC connection. It is static and has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc_connection</td>
<td>Valid open connection to the database</td>
</tr>
</tbody>
</table>

The method in **Example 13–2** can be used if the user wants to use an existing JDBC connection (say from a connection pool) for JMS operations. In this case JMS does not open a new connection, but instead uses the supplied JDBC connection to create the JMS `QueueConnection` object.
Creating a Session

**Example 13–2 Creating a QueueConnection with an Open JDBC Connection**

```java
Connection db_conn;     /* previously opened JDBC connection */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(db_conn);
```

The method in Example 13–3 is the only way to create a JMS QueueConnection when using JMS from a Java stored procedures inside the database (JDBC Server driver)

**Example 13–3 Creating a QueueConnection from a Java Procedure Inside Database**

```java
OracleDriver ora = new OracleDriver();
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(ora.defaultConnection());
```

Creating a QueueConnection with Default ConnectionFactory Parameters

```java
public javax.jms.QueueConnection createQueueConnection()
  throws JMSException
```

This method creates a queue connection with default ConnectionFactory parameters. If the queue connection factory properties do not contain a default username and password, then it throws a JMSException.

Creating a QueueConnection with an Open OracleOCIConnectionPool

```java
public static javax.jms.QueueConnection createQueueConnection(
  oracle.jdbc.pool.OracleOCIConnectionPool cpool)
  throws JMSException
```

This method creates a queue connection with an open OracleOCIConnectionPool. It is static and has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpool</td>
<td>Valid open OCI connection pool to the database</td>
</tr>
</tbody>
</table>

The method in Example 13–4 can be used if the user wants to use an existing OracleOCIConnectionPool instance for JMS operations. In this case JMS does not open a new OracleOCIConnectionPool instance, but instead uses the supplied OracleOCIConnectionPool instance to create the JMS QueueConnection object.

**Example 13–4 Creating a QueueConnection with an Open OracleOCIConnectionPool**

```java
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(cpool);
```

Creating a Session

```java
public javax.jms.Session createSession(boolean transacted,
  int ack_mode)
  throws JMSException
```

This method creates a Session, which supports both point-to-point and publish/subscribe operations. This method is new and supports JMS version 1.1
Creating a QueueSession

```
public javax.jms.QueueSession createQueueSession(
    boolean transacted, int ack_mode)
    throws JMSException
```

This method creates a QueueSession. Transactional and nontransactional sessions are supported. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transacted</td>
<td>If set to true, then the session is transactional</td>
</tr>
<tr>
<td>ack_mode</td>
<td>Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.</td>
</tr>
</tbody>
</table>

Example 13-5  Creating a Transactional QueueSession

```
QueueConnection qc_conn;
QueueSession q_sess = qc_conn.createQueueSession(true, 0);
```

Creating a QueueSender

```
public javax.jms.QueueSender createSender(javax.jms.Queue queue)
    throws JMSException
```

This method creates a QueueSender. If a sender is created without a default queue, then the destination queue must be specified on every send operation. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Name of destination queue</td>
</tr>
</tbody>
</table>

Sending Messages Using a QueueSender with Default Send Options

```
public void send(javax.jms.Queue queue,
    javax.jms.Message message)
    throws JMSException
```

This method sends a message using a QueueSender with default send options. This operation uses default values for message priority (1) and timeToLive (infinite). It has the following parameters:
Sending Messages Using a QueueSender by Specifying Send Options

If the QueueSender has been created with a default queue, then the queue parameter may not necessarily be supplied in the send() call. If a queue is specified in the send() operation, then this value overrides the default queue of the QueueSender.

If the QueueSender has been created without a default queue, then the queue parameter must be specified in every send() call.

**Example 13–6 Creating a Sender to Send Messages to Any Queue**

```java
/* Create a sender to send messages to any queue */
QueueSession jms_sess;
QueueSender sender1;
TextMessage message;
sender1 = jms_sess.createSender(null);
sender1.send(queue, message);
```

**Example 13–7 Creating a Sender to Send Messages to a Specific Queue**

```java
/* Create a sender to send messages to a specific queue */
QueueSession jms_sess;
QueueSender sender2;
Queue billed_orders_que;
TextMessage message;
sender2 = jms_sess.createSender(billed_orders_que);
sender2.send(queue, message);
```

Sending Messages Using a QueueSender by Specifying Send Options

```java
public void send(javax.jms.Queue queue,
                 javax.jms.Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive)
    throws JMSException
```

This method sends messages using a QueueSender by specifying send options. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to send this message to</td>
</tr>
<tr>
<td>message</td>
<td>Message to send</td>
</tr>
<tr>
<td>deliveryMode</td>
<td>Delivery mode to use</td>
</tr>
<tr>
<td>priority</td>
<td>Priority for this message</td>
</tr>
<tr>
<td>timeToLive</td>
<td>Message lifetime in milliseconds (zero is unlimited)</td>
</tr>
</tbody>
</table>

If the QueueSender has been created with a default queue, then the queue parameter may not necessarily be supplied in the send() call. If a queue is specified in the send() operation, then this value overrides the default queue of the QueueSender.
If the QueueSender has been created without a default queue, then the queue parameter must be specified in every send() call.

**Example 13–8 Sending Messages Using a QueueSender by Specifying Send Options 1**

```java
/* Create a sender to send messages to any queue */
/* Send a message to new_orders_que with priority 2 and timetoLive 100000 milliseconds */
QueueSession  jms_sess;
QueueSender  sender1;
TextMessage mesg;
Queue   new_orders_que
sender1 = jms_sess.createSender(null);
sender1.send(new_orders_que, mesg, DeliveryMode.PERSISTENT, 2, 100000);
```

**Example 13–9 Sending Messages Using a QueueSender by Specifying Send Options 2**

```java
/* Create a sender to send messages to a specific queue */
/* Send a message with priority 1 and timetoLive 400000 milliseconds */
QueueSession jms_sess;
QueueSender sender2;
Queue   billed_orders_que;
TextMessage mesg;
sender2 = jms_sess.createSender(billed_orders_que);
sender2.send(mesg, DeliveryMode.PERSISTENT, 1, 400000);
```

**Creating a QueueBrowser for Standard JMS Type Messages**

```java
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
   java.lang.String messageSelector)
   throws JMSException
```

This method creates a QueueBrowser for queues with text, stream, objects, bytes or MapMessage message bodies. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered</td>
</tr>
</tbody>
</table>

**See Also:** "MessageSelector" on page 11-17

Use methods in java.util.Enumeration to go through list of messages.

**Example 13–10 Creating a QueueBrowser Without a Selector**

```java
/* Create a browser without a selector */
QueueSession     jms_session;
QueueBrowser      browser;
Queue            queue;
browser = jms_session.createBrowser(queue);
```

**Example 13–11 Creating a QueueBrowser With a Specified Selector**

```java
/* Create a browser for queues with a specified selector */
QueueSession     jms_session;
QueueBrowser      browser;
```
Queue queue;
/* create a Browser to look at messages with correlationID = RUSH */
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");

Creating a QueueBrowser for Standard JMS Type Messages, Locking Messages

public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
java.lang.String messageSelector,
boolean locked)
throws JMSException

This method creates a QueueBrowser for queues with TextMessage, StreamMessage, ObjectMessage, BytesMessage, or MapMessage message bodies, locking messages while browsing. Locked messages cannot be removed by other consumers until the browsing session ends the transaction. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered</td>
</tr>
<tr>
<td>locked</td>
<td>If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)</td>
</tr>
</tbody>
</table>

Example 13–12  Creating a QueueBrowser Without a Selector, Locking Messages

/* Create a browser without a selector */
QueueSession jms_session;
QueueBrowser browser;
Queue queue;
browser = jms_session.createBrowser(queue, null, true);

Example 13–13  Creating a QueueBrowser With a Specified Selector, Locking Messages

/* Create a browser for queues with a specified selector */
QueueSession jms_session;
QueueBrowser browser;
Queue queue;
/* create a Browser to look at messages with correlationID = RUSH in lock mode */
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'", true);

Creating a QueueBrowser for Oracle Object Type Messages

public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
java.lang.String messageSelector,
java.lang.Object payload_factory)
throws JMSException

This method creates a QueueBrowser for queues of Oracle object type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
</tbody>
</table>
Creating a QueueBrowser for Oracle Object Type Messages, Locking Messages

**CustomDatumFactory** for a particular java class that maps to the SQL object payload can be obtained using the `getFactory` static method.

Assume the queue `test_queue` has payload of type `SCOTT.EMPLOYEE` and the java class that is generated by JPublisher for this Oracle object type is called `Employee`. The `Employee` class implements the `CustomDatum` interface. The `CustomDatumFactory` for this class can be obtained by using the `Employee.getFactory()` method.

**Example 13–14 Creating a QueueBrowser for ADTMessages**

```java
/* Create a browser for a Queue with AdtMessage messages of type EMPLOYEE*/
QueueSession jms_session
QueueBrowser browser;
Queue test_queue;
browser = ((AQjmsSession)jms_session).createBrowser(test_queue,
        "corrid='EXPRESS'",
        Employee.getFactory());
```

**Creating a QueueBrowser for Oracle Object Type Messages, Locking Messages**

```java
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
        java.lang.String messageSelector,
        java.lang.Object payload_factory,
        boolean locked)
        throws JMSException
```

This method creates a `QueueBrowser` for queues of Oracle object type messages, locking messages while browsing. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT</td>
</tr>
<tr>
<td>locked</td>
<td>If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)</td>
</tr>
</tbody>
</table>
Creating a QueueReceiver for Oracle Object Type Messages

public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
java.lang.String messageSelector,
java.lang.Object payload_factory)
throws JMSException

Note: CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Creating a QueueReceiver for Standard JMS Type Messages

public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
java.lang.String messageSelector)
throws JMSException

This method creates a QueueReceiver for queues of standard JMS type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered</td>
</tr>
</tbody>
</table>

See Also: "MessageSelector" on page 11-17

Creating a QueueReceiver for Oracle Object Type Messages

Example 13–15 Creating a QueueBrowser for AdtMessages, Locking Messages

Example 13–16 Creating a QueueReceiver Without a Selector

Example 13–17 Creating a QueueReceiver With a Specified Selector
This method creates a QueueReceiver for queues of Oracle object type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Queue to access</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**See Also:** "MessageSelector" on page 11-17

The CustomDatumFactory for a particular java class that maps to the SQL object type payload can be obtained using the getFactory static method.

**Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Assume the queue test_queue has payload of type SCOTT.EMPLOYEE and the java class that is generated by Jpublisher for this Oracle object type is called Employee. The Employee class implements the CustomDatum interface. The ORADataFactory for this class can be obtained by using the Employee.getFactory() method.

**Example 13–18  Creating a QueueReceiver for AdtMessage Messages**

```java
/* Create a receiver for a Queue with AdtMessage messages of type EMPLOYEE*/
QueueSession jms_session
QueueReceiver receiver;
Queue test_queue;
browser = ((AQjmsSession)jms_session).createReceiver(
    test_queue,
    'JMSCorrelationID = 'MANAGER',
    Employee.getFactory());
```

This chapter describes the components of the Oracle Streams Advanced Queuing (AQ) Java Message Service (JMS) operational interface that are specific to publish/subscribe operations. Components that are shared by point-to-point and publish/subscribe are described in Chapter 15, "Oracle JMS Shared Interfaces".

This chapter contains these topics:

- Creating a Connection with Username/Password
- Creating a Connection with Default ConnectionFactory Parameters
- Creating a TopicConnection with Username/Password
- Creating a TopicConnection with Open JDBC Connection
- Creating a TopicConnection with an Open OracleOCIConnectionPool
- Creating a Session
- Creating a TopicSession
- Creating a TopicPublisher
- Publishing Messages with Minimal Specification
- Publishing Messages Specifying Topic
- Publishing Messages Specifying Delivery Mode, Priority and TimeToLive
- Publishing Messages Specifying a Recipient List
- Creating a DurableSubscriber for a JMS Topic Without Selector
- Creating a DurableSubscriber for a JMS Topic With Selector
- Creating a DurableSubscriber for an Oracle Object Type Topic Without Selector
- Creating a DurableSubscriber for an Oracle Object Type Topic With Selector
- Specifying Transformations for Topic Subscribers
- Creating a Remote Subscriber for JMS Messages
- Creating a Remote Subscriber for Oracle Object Type Messages
- Specifying Transformations for Remote Subscribers
- Unsubscribing a Durable Subscription for a Local Subscriber
- Unsubscribing a Durable Subscription for a Remote Subscriber
- Creating a TopicReceiver for a Topic of Standard JMS Type Messages
- Creating a TopicReceiver for a Topic of Oracle Object Type Messages
Creating a Connection with Username/Password

```java
public javax.jms.Connection createConnection(
    java.lang.String username,
    java.lang.String password)
throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with the specified username and password. This method is new and supports JMS version 1.1 specifications. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user connecting to the database for queuing</td>
</tr>
<tr>
<td>password</td>
<td>Password for creating the connection to the server</td>
</tr>
</tbody>
</table>

Creating a Connection with Default ConnectionFactory Parameters

```java
public javax.jms.Connection createConnection()
throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with default ConnectionFactory parameters. This method is new and supports JMS version 1.1 specifications. If the ConnectionFactory properties do not contain a default username and password, then it throws a JMSException.

Creating a TopicConnection with Username/Password

```java
public javax.jms.TopicConnection createTopicConnection(
    java.lang.String username,
    java.lang.String password)
throws JMSException
```

This method creates a TopicConnection with the specified username/password. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Name of the user connecting to the database for queuing</td>
</tr>
<tr>
<td>password</td>
<td>Password for creating the connection to the server</td>
</tr>
</tbody>
</table>

Example 14-1  Creating a TopicConnection with Username/Password

```java
TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("sun123", "oratest", 5521, "thin"); /* Create a TopicConnection using a username/password */ TopicConnection tc_conn = tc_fact.createTopicConnection("jmsuser", "jmsuser");
```
Creating a TopicConnection with Open JDBC Connection

```java
public static javax.jms.TopicConnection createTopicConnection(
    java.sql.Connection jdbc_connection)
    throws JMSException
```

This method creates a TopicConnection with open JDBC connection. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc_connection</td>
<td>Valid open connection to database</td>
</tr>
</tbody>
</table>

**Example 14–2 Creating a TopicConnection with Open JDBC Connection**

```java
Connection db_conn; /*previously opened JDBC connection */
TopicConnection tc_conn =
AQjmsTopicConnectionFactory createTopicConnection(db_conn);
```

**Example 14–3 Creating a TopicConnection with New JDBC Connection**

```java
OracleDriver ora = new OracleDriver();
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(ora.defaultConnection());
```

Creating a TopicConnection with an Open OracleOCIConnectionPool

```java
public static javax.jms.TopicConnection createTopicConnection(
    oracle.jdbc.pool.OracleOCIConnectionPool cpool)
    throws JMSException
```

This method creates a TopicConnection with an open OracleOCIConnectionPool. It is static and has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpool</td>
<td>Valid open OCI connection pool to the database</td>
</tr>
</tbody>
</table>

**Example 14–4 Creating a TopicConnection with Open OracleOCIConnectionPool**

```java
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(cpool);
```

Creating a Session

```java
public javax.jms.Session createSession(boolean transacted,
    int ack_mode)
    throws JMSException
```

This method creates a Session supporting both point-to-point and publish/subscribe operations. It is new and supports JMS version 1.1 specifications. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transacted</td>
<td>If set to true, then the session is transactional</td>
</tr>
</tbody>
</table>
Creating a TopicSession

```java
public javax.jms.TopicSession createTopicSession(boolean transacted,
                                              int ack_mode)
    throws JMSException
```

This method creates a TopicSession. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transacted</td>
<td>If set to true, then the session is transactional.</td>
</tr>
<tr>
<td>ack_mode</td>
<td>Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.</td>
</tr>
</tbody>
</table>

Example 14–5  Creating a TopicSession

```java
TopicConnection tc_conn;
TopicSession t_sess = tc_conn.createTopicSession(true, 0);
```

Creating a TopicPublisher

```java
public javax.jms.TopicPublisher createPublisher(javax.jms.Topic topic)
    throws JMSException
```

This method creates a TopicPublisher. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to publish to, or null if this is an unidentified producer</td>
</tr>
</tbody>
</table>

Publishing Messages with Minimal Specification

```java
public void publish(javax.jms.Message message)
    throws JMSException
```

This method publishes a message with minimal specification. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to send</td>
</tr>
</tbody>
</table>

The TopicPublisher uses the default values for message priority (1) and timeToLive (infinite).

Example 14–6  Publishing Without Specifying Topic

```java
/* Publish without specifying topic */
```
Publishing Messages Specifying Topic

```java
public class Example14_7 {
    public void publish(javax.jms.Topic topic, javax.jms.Message message)
        throws JMSException {
        // Create connection and session
        TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("
            MYHOSTNAME",
            "MYSID",
            myport,
            "oci8");
        TopicConnection t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
        TopicSession jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
        Topic shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE",
            "Shipped_Orders_Topic");
        publisher1 = jms_sess.createPublisher(shipped_orders);
        TextMessage text_message = jms_sess.createTextMessage();
        publisher1.publish(text_message);
    }
}
```

**Example 14–7  Publishing Specifying Correlation and Delay**

```java
public class Example14_7 {
    public void publish(javax.jms.Topic topic, javax.jms.Message message)
        throws JMSException {
        // Create connection and session
        TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("
            MYHOSTNAME",
            "MYSID",
            myport,
            "oci8");
        TopicConnection t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
        TopicSession jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
        Topic shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE",
            "Shipped_Orders_Topic");
        publisher1 = jms_sess.createPublisher(shipped_orders);
        TextMessage text_message = jms_sess.createTextMessage();
        jms_sess.setJMSCorrelationID("FOO");
        jms_sess.setLongProperty("JMS_OracleDelay", 30);
        publisher1.publish(text_message);
    }
}
```

**Publishing Messages Specifying Topic**

```java
public void publish(javax.jms.Topic topic, javax.jms.Message message)
    throws JMSException {
    // Create connection and session
    TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("
        MYHOSTNAME",
        "MYSID",
        myport,
        "oci8");
    TopicConnection t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
    TopicSession jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
    Topic shipped_orders = ((AQjmsSession) jms_sess).getTopic("OE",
        "Shipped_Orders_Topic");
    publisher1 = jms_sess.createPublisher(shipped_orders);
    TextMessage text_message = jms_sess.createTextMessage();
    jms_sess.setJMSCorrelationID("FOO");
    jms_sess.setLongProperty("JMS_OracleDelay", 30);
    publisher1.publish(text_message);
}
```
This method publishes a message specifying the topic. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to publish to</td>
</tr>
<tr>
<td>message</td>
<td>Message to send</td>
</tr>
</tbody>
</table>

If the `TopicPublisher` has been created with a default topic, then the `topic` parameter may not be specified in the `publish()` call. If a topic is specified, then that value overrides the default in the `TopicPublisher`. If the `TopicPublisher` has been created without a default topic, then the topic must be specified with the `publish()` call.

**Example 14–8 Publishing Specifying Topic**

```java
/* Publish specifying topic */
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession jms_sess;
TopicPublisher publisher1;
Topic shipped_orders;
int myport = 5521;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory('MYHOSTNAME', 'MYSID', myport, 'oci8');
t_conn = tc_fact.createTopicConnection('jmstopic', 'jmstopic');
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* create TopicPublisher */
publisher1 = jms_sess.createPublisher(null);
/* get topic object */
shipped_orders = ((AQjmsSession )jms_sess).getTopic('WS', 'Shipped_Orders_Topic');
/* create text message */
TextMessage jms_sess.createTextMessage();
/* publish specifying the topic */
publisher1.publish(shipped_orders, text_message);
```

---

Publishing Messages Specifying Delivery Mode, Priority and TimeToLive

```java
public void publish(javax.jms.Topic topic, javax.jms.Message message, oracle.jms.AQjmsAgent[] recipient_list, int deliveryMode, int priority, long timeToLive)
throws JMSException
```

This method publishes a message specifying delivery mode, priority and `TimeToLive`. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to which to publish the message (overrides the default topic of the MessageProducer)</td>
</tr>
<tr>
<td>message</td>
<td>Message to publish</td>
</tr>
</tbody>
</table>
### Example 14–9 Publishing Specifying Priority and TimeToLive

```java
TopicConnectionFactory       tc_fact = null;
TopicConnection             t_conn = null;
TopicSession                jms_sess;
TopicPublisher              publisher1;
Topic                       shipped_orders;
int                         myport = 5521;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* Create TextMessage */
TextMessage                 jms_sess.createTextMessage();
/* Publish message with priority 1 and time to live 200 seconds */
publisher1.publish(text_message, DeliveryMode.PERSISTENT, 1, 200000);
```

### Publishing Messages Specifying a Recipient List

```java
public void publish(javax.jms.Message message,
                    oracle.jms.AQjmsAgent[] recipient_list)
    throws JMSException
```

This method publishes a message specifying a recipient list overriding topic subscribers. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to publish</td>
</tr>
<tr>
<td>recipient_list</td>
<td>List of recipients to which the message is published. Recipients are of type AQjmsAgent.</td>
</tr>
</tbody>
</table>

### Example 14–10 Publishing Specifying a Recipient List Overriding Topic Subscribers

```java
/* Publish specifying priority and timeToLive */
TopicConnectionFactory       tc_fact = null;
TopicConnection             t_conn = null;
TopicSession                jms_sess;
TopicPublisher              publisher1;
Topic                       shipped_orders;
int                         myport = 5521;
AQjmsAgent[]                recipList;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
```

---

**Parameter | Description**
---|---
recipient_list | List of recipients to which the message is published. Recipients are of type AQjmsAgent.
deliveryMode | PERSISTENT or NON_PERSISTENT (only PERSISTENT is supported in this release)
priority | Priority for this message
timeToLive | Message lifetime in milliseconds (zero is unlimited)
Creating a DurableSubscriber for a JMS Topic Without Selector

public javax.jms.TopicSubscriber createDurableSubscriber(
    javax.jms.Topic topic,
    java.lang.String subs_name)
    throws JMSException

This method creates a DurableSubscriber for a JMS topic without selector. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to subscribe to</td>
</tr>
<tr>
<td>subs_name</td>
<td>Name used to identify this subscription</td>
</tr>
</tbody>
</table>

Exclusive Access to Topics

CreateDurableSubscriber() and Unsubscribe() both require exclusive access to their target topics. If there are pending JMS send(), publish(), or receive() operations on the same topic when these calls are applied, then exception ORA - 4020 is raised. There are two solutions to the problem:

- Limit calls to createDurableSubscriber() and Unsubscribe() to the setup or cleanup phase when there are no other JMS operations pending on the topic. That makes sure that the required resources are not held by other JMS operational calls.
- Call TopicSession.commit before calling createDurableSubscriber() or Unsubscribe().

Example 14–11  Creating a Durable Subscriber for a JMS Topic Without Selector

```
/*MYHOSTNAME", "MYSID", myport, "oci8*");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = new AQjmsSession().getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* create TextMessage */
TextMessage jms_sess.createTextMessage();
/* create two receivers */
recipList = new AQjmsAgent[2];
recipList[0] = new AQjmsAgent("ES", "ES.shipped_orders_topic", AQAgent.DEFAULT_AGENT_PROTOCOL);
recipList[1] = new AQjmsAgent("WS", "WS.shipped_orders_topic", AQAgent.DEFAULT_AGENT_PROTOCOL);
/* publish message specifying a recipient list */
publisher1.publish(text_message, recipList);
```
Creating a DurableSubscriber for a JMS Topic With Selector

```java
public javax.jms.TopicSubscriber createDurableSubscriber(
    javax.jms.Topic topic,
    java.lang.String subs_name,
    java.lang.String messageSelector,
    boolean noLocal)
    throws JMSException
```

This method creates a durable subscriber for a JMS topic with selector. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to subscribe to</td>
</tr>
<tr>
<td>subs_name</td>
<td>Name used to identify this subscription</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>noLocal</td>
<td>If set to true, then it inhibits the delivery of messages published by its own connection</td>
</tr>
</tbody>
</table>

**See Also:** "Exclusive Access to Topics" on page 14-8

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a different messageSelector. An unsubscribe call is needed to end the subscription to the topic.

**See Also:** "MessageSelector" on page 11-17

**Example 14–12 Creating a Durable Subscriber for a JMS Topic With Selector**

```java
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession jms_sess;
TopicSubscriber subscriber1;
Topic shipped_orders;
int myport = 5521;
AQjmsAgent[] recipList;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE",
    "Shipped_Orders_Topic");
/* create a durable subscriber on the shipped_orders topic*/
subscriber1 = jms_sess.createDurableSubscriber(
    shipped_orders,
    "WesternShipping");
```
Creating a DurableSubscriber for an Oracle Object Type Topic Without Selector

```java
public javax.jms.TopicSubscriber createDurableSubscriber(
    javax.jms.Topic topic,
    java.lang.String subs_name,
    java.lang.Object payload_factory)
throws JMSException
```

This method creates a durable subscriber for an Oracle object type topic without selector. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to subscribe to</td>
</tr>
<tr>
<td>subs_name</td>
<td>Name used to identify this subscription</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

**See Also:** "Exclusive Access to Topics" on page 14-8

**Example 14–13  Creating a Durable Subscriber for an Oracle Object Type Topic Without Selector**

```java
/* Subscribe to an ADT queue */
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession t_sess = null;
TopicSession jms_sess;
TopicSubscriber subscriber1;
Topic shipped_orders;
int my[port = 5521;
AQjmsAgent[] recipList;
/* the java mapping of the oracle object type created by J Publisher */
ADTMessage message;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE", "Shipped_Orders_Topic");
/* create a subscriber */
/* with condition on JMSPriority and user property 'Region' */
subscriber1 = jms_sess.createDurableSubscriber(
    shipped_orders, 'WesternShipping',
    "JMSPriority > 2 and Region like 'Western%'", false);
```
Creating a DurableSubscriber for an Oracle Object Type Topic With Selector

```java
subscriber1 = jms_sess.createDurableSubscriber(
    shipped_orders, 'WesternShipping', AQjmsAgent.getFactory());
```

### Creating a DurableSubscriber for an Oracle Object Type Topic With Selector

```java
public javax.jms.TopicSubscriber createDurableSubscriber(
    javax.jms.Topic topic,
    java.lang.String subs_name,
    java.lang.String messageSelector,
    boolean noLocal,
    java.lang.Object payload_factory)
throws JMSException
```

This method creates a durable subscriber for an Oracle object type topic with selector. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to subscribe to</td>
</tr>
<tr>
<td>subs_name</td>
<td>Name used to identify this subscription</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>noLocal</td>
<td>If set to true, then it inhibits the delivery of messages published by its own connection</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADATAFactory for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**Note:** CustomDatum support will be deprecated in a future release. Use ORADATAFactory payload factories instead.

**See Also:** "Exclusive Access to Topics" on page 14-8

### Example 14–14 Creating a Durable Subscriber for an Oracle Object Type Topic With Selector

```java
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicSubscriber           subscriber1;
Topic                     shipped_orders;
int                       myport = 5521;
AQjmsAgent[]              recipList;
/* the java mapping of the oracle object type created by J Publisher */
ADTMessage                message;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE", "Shipped_Orders_Topic");
/* create a subscriber, specifying correct CustomDatumFactory and selector */
subscriber1 = jms_sess.createDurableSubscriber(
```
shipped_orders, "WesternShipping",
"priority > 1 and tab.user_data.region like 'WESTERN %'", false,
ADTMessage.getFactory());

Specifying Transformations for Topic Subscribers

A transformation can be supplied when sending/publishing a message to a queue/topic. The transformation is applied before putting the message into the queue/topic.

The application can specify a transformation using the setTransformation interface in the AQjmsQueueSender and AQjmsTopicPublisher interfaces.

Example 14–15  Sending Messages to a Destination Using a Transformation

Suppose that the orders that are processed by the order entry application should be published to WS_bookedorders_topic. The transformation OE2WS (defined in the previous section) is supplied so that the messages are inserted into the topic in the correct format.

```java
public void ship_bookedorders(
    TopicSession jms_session,
    AQjmsADTMessage adt_message)
{
    TopicPublisher publisher;
    Topic topic;

    try
    {
        /* get a handle to the WS_bookedorders_topic */
        topic = ((AQjmsSession)jms_session).getTopic("WS", "WS_bookedorders_topic");
        publisher = jms_session.createPublisher(topic);

        /* set the transformation in the publisher */
        ((AQjmsTopicPublisher)publisher).setTransformation("OE2WS");
        publisher.publish(topic, adt_message);
    }
    catch (JMSException ex)
    {
        System.out.println('Exception :' ex);
    }
}
```

A transformation can also be specified when creating topic subscribers using the CreateDurableSubscriber() call. The transformation is applied to the retrieved message before returning it to the subscriber. If the subscriber specified in the CreateDurableSubscriber() call already exists, then its transformation is set to the specified transformation.

Example 14–16  Specifying Transformations for Topic Subscribers

The Western Shipping application subscribes to the OE_bookedorders_topic with the transformation OE2WS. This transformation is applied to the messages and the returned message is of Oracle object type WS.WS_orders.

Suppose that the WSOrder java class has been generated by Jpublisher to map to the Oracle object WS.WS_order:

```java
public AQjmsAdtMessage retrieve_bookedorders(TopicSession jms_session)
{
```
TopicSubscriber subscriber;
Topic topic;
AQjmsAdtMessage msg = null;

try {
    /* get a handle to the OE_bookedorders_topic */
    topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");

    /* create a subscriber with the transformation OE2WS */
    subs = ((AQjmsSession)jms_session).createDurableSubscriber(
            topic, 'WShip', null, false, WSOrder.getFactory(), "OE2WS");
    msg = subscriber.receive(10);
    }
    catch (JMSException ex)
    {
        System.out.println("Exception :" + ex);
    }
    return (AQjmsAdtMessage)msg;
}

Creating a Remote Subscriber for JMS Messages

public void createRemoteSubscriber(javax.jms.Topic topic,
                                    oracle.jms.AQjmsAgent remote_subscriber,
                                    java.lang.String messageSelector)
                                    throws JMSException

This method creates a remote subscriber for topics of JMS messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to subscribe to</td>
</tr>
<tr>
<td>remote_subscriber</td>
<td>AQjmsAgent that refers to the remote subscriber</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
</tbody>
</table>

See Also:  "MessageSelector" on page 11-17

Oracle® Database allows topics to have remote subscribers, for example, subscribers at other topics in the same or different database. In order to use remote subscribers, you must set up propagation between the local and remote topic.

Remote subscribers can be a specific consumer at the remote topic or all subscribers at the remote topic. A remote subscriber is defined using the AQjmsAgent structure. An AQjmsAgent consists of a name and address. The name refers to the consumer_name at the remote topic. The address refers to the remote topic. Its syntax is schema.topic_name[@dblink].

To publish messages to a particular consumer at the remote topic, the subscription_name of the recipient at the remote topic must be specified in the name field of AQjmsAgent, and the remote topic must be specified in the address field. To publish messages to all subscribers of the remote topic, the name field of AQjmsAgent must be set to null.
Creating a Remote Subscriber for Oracle Object Type Messages

Example 14–17  Creating a Remote Subscriber for Topics of JMS Messages

```java
public void createRemoteSubscriber(javax.jms.Topic topic,
         oracle.jms.AQjmsAgent remote_subscriber,
         java.lang.String messageSelector,
         java.lang.Object payload_factory)
throws JMSException
```

This method creates a remote subscriber for topics of Oracle object type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to subscribe to</td>
</tr>
<tr>
<td>remote_subscriber</td>
<td>AQjmsAgent that refers to the remote subscriber</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADATAFactory for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**Note:** CustomDatum support will be deprecated in a future release. Use ORADATAFactory payload factories instead.

**See Also:** "MessageSelector" on page 11-17

Oracle® Database allows topics to have remote subscribers, for example, subscribers at other topics in the same or different database. In order to use remote subscribers, you must set up propagation between the local and remote topic.

Remote subscribers can be a specific consumer at the remote topic or all subscribers at the remote topic. A remote subscriber is defined using the AQjmsAgent structure. An AQjmsAgent consists of a name and address. The name refers to the consumer_name...
at the remote topic. The address refers to the remote topic. Its syntax is
schema.topic_name[@dblink].

To publish messages to a particular consumer at the remote topic, the
subscription_name of the recipient at the remote topic must be specified in the
name field of AQjmsAgent, and the remote topic must be specified in the address
field. To publish messages to all subscribers of the remote topic, the name field of
AQjmsAgent must be set to null.

Example 14–18 Creating a Remote Subscriber for Topics of Oracle Object Type
Messages

```
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession t_sess = null;
TopicSession jms_sess;
TopicSubscriber subscriber1;
Topic shipped_orders;
int myport = 5521;
AQjmsAgent remoteAgent;
ADTMessage message;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
/* create TopicSession */
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* get the Shipped order topic */
ishipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE", "Shipped_Orders_Topic");
/* create a remote agent */
remoteAgent = new AQjmsAgent("WesternRegion", "WS.shipped_orders_topic", null);
/* create a remote subscriber with null selector*/
subscriber1 = ((AQjmsSession)jms_sess).createRemoteSubscriber(
    shipped_orders, remoteAgent, null, message.getFactory);
```

Note: AQ does not support the use of multiple dblink to the same
destination. As a workaround, use a single database link for each
destination.

Specifying Transformations for Remote Subscribers

Oracle® Database allows a remote subscriber, that is a subscriber at another database,
to subscribe to a topic.

Transformations can be specified when creating remote subscribers using the
createRemoteSubscriber() call. This enables propagation of messages between
topics of different formats. When a message published at a topic meets the criterion of
a remote subscriber, Oracle® Database automatically propagates the message to the
queue/topic at the remote database specified for the remote subscriber. If a
transformation is also specified, then Oracle® Database applies the transformation to
the message before propagating it to the queue/topic at the remote database.

Example 14–19 Specifying Transformations for Remote Subscribers

A remote subscriber is created at the OE.OE_bookedorders_topic so that messages are
automatically propagated to the WS.WS_bookedorders_topic. The transformation
OE2WS is specified when creating the remote subscriber so that the messages reaching the WS_bookedorders_topic have the correct format.

Suppose that the WSOder java class has been generated by Jpublisher to map to the Oracle object WS.WS_order

```java
public void create_remote_sub(TopicSession jms_session) {
    AQjmsAgent subscriber;
    Topic topic;
    try {
        /* get a handle to the OE_bookedorders_topic */
        topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");
        subscriber = new AQjmsAgent("WShip", "WS.WS_bookedorders_topic");

        ((AQjmsSession )jms_session).createRemoteSubscriber(
            topic, subscriber, null, WSOder.getFactory(), "OE2WS");
    }
    catch (JMSException ex) {
        System.out.println('Exception :' ex);
    }
}
```

### Unsubscribing a Durable Subscription for a Local Subscriber

```java
public void unsubscribe(javax.jms.Topic topic,
                        java.lang.String subs_name)
    throws JMSException
```

This method unsubscribes a durable subscription for a local subscriber. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to unsubscribe</td>
</tr>
<tr>
<td>subs_name</td>
<td>Name used to identify this subscription</td>
</tr>
</tbody>
</table>

**See Also:** "Exclusive Access to Topics" on page 14-8

#### Example 14–20  Unsubscribing a Durable Subscription for a Local Subscriber

```java
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicSubscriber           subscriber1;
Topic                     shipped_orders;
int                       myport = 5521;
AQjmsAgent[]              recipList;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
            "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
            "OE", "Shipped_Orders_Topic");
/* unsubscribe 'WesternShipping' from shipped_orders */
```
Unsubscribing a Durable Subscription for a Remote Subscriber

```java
public void unsubscribe(javax.jms.Topic topic, oracle.jms.AQjmsAgent remote_subscriber) throws JMSException
```

This method unsubscribes a durable subscription for a remote subscriber. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Non-temporary topic to unsubscribe</td>
</tr>
<tr>
<td>remote_subscriber</td>
<td>AQjmsAgent that refers to the remote subscriber. The address field of the AQjmsAgent cannot be null.</td>
</tr>
</tbody>
</table>

**See Also:** "Exclusive Access to Topics" on page 14-8

**Example 14–21 Unsubscribing a Durable Subscription for a Remote Subscriber**

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
AQjmsAgent                remoteAgent;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(    
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "OE", "Shipped_Orders_Topic");
remoteAgent = new AQjmsAgent("WS", "WS.Shipped_Orders_Topic", null);
/* unsubscribe the remote agent from shipped_orders */
((AQjmsSession)jms_sess).unsubscribe(shipped_orders, remoteAgent);
```

Creating a TopicReceiver for a Topic of Standard JMS Type Messages

```java
public oracle.jms.AQjmsTopicReceiver createTopicReceiver(
    javax.jms.Topic topic,  
    java.lang.String receiver_name,  
    java.lang.String messageSelector)  
throws JMSException
```

This method creates a TopicReceiver for a topic of standard JMS type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>receiver_name</td>
<td>Name of message receiver</td>
</tr>
</tbody>
</table>
Creating a TopicReceiver for a Topic of Oracle Object Type Messages

Oracle® Database allows messages to be sent to specified recipients. These receivers may or may not be subscribers of the topic. If the receiver is not a subscriber to the topic, then it receives only those messages that are explicitly addressed to it. This method must be used to create a `TopicReceiver` object for consumers that are not durable subscribers.

**Example 14–22 Creating a TopicReceiver for Standard JMS Type Messages**

```java
TopicConnectionFactory tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
TopicReceiver             receiver;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(  
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(  
    "WS", "Shipped_Orders_Topic");
receiver = ((AQjmsSession)jms_sess).createTopicReceiver(  
           shipped_orders, "WesternRegion", null);
```

**Creating a TopicReceiver for a Topic of Oracle Object Type Messages**

```java
public oracle.jms.AQjmsTopicReceiver createTopicReceiver(  
    javax.jms.Topic topic,  
    java.lang.String receiver_name,  
    java.lang.String messageSelector,  
    java.lang.Object payload_factory)  
    throws JMSException
```

This method creates a `TopicReceiver` for a topic of Oracle object type messages with selector. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>receiver_name</td>
<td>Name of message receiver</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the <code>messageSelector</code> expression are delivered. A value of null or an empty string indicates that there is no <code>messageSelector</code> for the message consumer.</td>
</tr>
<tr>
<td>payload_factory</td>
<td><code>CustomDatumFactory</code> or <code>ORADataFactory</code> for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>
Oracle® Database allows messages to be sent to all subscribers of a topic or to specified recipients. These receivers may or may not be subscribers of the topic. If the receiver is not a subscriber to the topic, then it receives only those messages that are explicitly addressed to it. This method must be used to create a `TopicReceiver` object for consumers that are not durable subscribers.

### Example 14–23 Creating a TopicReceiver for Oracle Object Type Messages

```java
topicConnectionFactory    = null;
topicConnection           = null;
topicSession              = null;
topicSession              = jms_sess;
topic                     = shipped_orders;
int                       = myport = 5521;
TopicReceiver             = receiver;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "WS", "Shipped_Orders_Topic");
receiver = ((AQjmsSession)jms_sess).createTopicReceiver(
    shipped_orders, "WesternRegion", null);
```

### Creating a TopicBrowser for Standard JMS Messages

```java
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
    java.lang.String cons_name,
    java.lang.String messageSelector)
    throws JMSException
```

This method creates a `TopicBrowser` for topics with `TextMessage`, `StreamMessage`, `ObjectMessage`, `BytesMessage`, or `MapMessage` message bodies. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>cons_name</td>
<td>Name of the durable subscriber or consumer</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**See Also:** "MessageSelector" on page 11-17

---

**Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.
Example 14–24  Creating a TopicBrowser Without a Selector

/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(topic, "SUBS1");

Example 14–25  Creating a TopicBrowser With a Specified Selector

/* Create a browser for topics with a specified selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
/* create a Browser to look at messages with correlationID = RUSH */
browser = ((AQjmsSession) jms_session).createBrowser(
    topic, "SUBS1", "JMSCorrelationID = 'RUSH'");

Creating a TopicBrowser for Standard JMS Messages, Locking Messages

public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
    java.lang.String cons_name,
    java.lang.String messageSelector,
    boolean locked)
throws JMSException

This method creates a TopicBrowser for topics with text, stream, objects, bytes or map messages, locking messages while browsing. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>cons_name</td>
<td>Name of the durable subscriber or consumer</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>locked</td>
<td>If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)</td>
</tr>
</tbody>
</table>

Example 14–26  Creating a TopicBrowser Without a Selector, Locking Messages While Browsing

/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(
    topic, "SUBS1", true);

Example 14–27  Creating a TopicBrowser With a Specified Selector, Locking Messages

/* Create a browser for topics with a specified selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
/* create a Browser to look at messages with correlationID = RUSH in lock mode */
browser = ((AQjmsSession) jms_session).createBrowser(
Creating a TopicBrowser for Oracle Object Type Messages

```java
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
        java.lang.String cons_name,
        java.lang.String messageSelector,
        java.lang.Object payload_factory)
    throws JMSException
```

This method creates a TopicBrowser for topics of Oracle object type messages. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>cons_name</td>
<td>Name of the durable subscriber or consumer</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the Java class that maps to the Oracle ADT</td>
</tr>
</tbody>
</table>

**Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

**See Also:** "MessageSelector" on page 11-17

The CustomDatumFactory for a particular Java class that maps to the SQL object type payload can be obtained using the getFactory static method. Assume the topic test_topic has payload of type SCOTT.EMPLOYEE and the Java class that is generated by jpublisher for this Oracle object type is called Employee. The Employee class implements the CustomDatum interface. The CustomDatumFactory for this class can be obtained by using the Employee.getFactory() method.

**Example 14–28 Creating a TopicBrowser for AdtMessage Messages**

```java
/* Create a browser for a Topic with AdtMessage messages of type EMPLOYEE*/
TopicSession jms_session
TopicBrowser browser;
Topic        test_topic;
browser = ((AQjmsSession) jms_session).createBrowser(
    test_topic, "SUBS1", Employee.getFactory());
```

Creating a TopicBrowser for Oracle Object Type Messages, Locking Messages

```java
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
        java.lang.String cons_name,
        java.lang.String messageSelector,
        java.lang.Object payload_factory,
        boolean locked)
    throws JMSException
```
This method creates a TopicBrowser for topics of Oracle object type messages, locking messages while browsing. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic</td>
<td>Topic to access</td>
</tr>
<tr>
<td>cons_name</td>
<td>Name of the durable subscriber or consumer</td>
</tr>
<tr>
<td>messageSelector</td>
<td>Only messages with properties matching the messageSelector expression are delivered. A value of null or an empty string indicates that there is no messageSelector for the message consumer.</td>
</tr>
<tr>
<td>payload_factory</td>
<td>CustomDatumFactory or ORADataFactory for the Java class that maps to the Oracle ADT</td>
</tr>
<tr>
<td>locked</td>
<td>If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)</td>
</tr>
</tbody>
</table>

**Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

**See Also:** "MessageSelector" on page 11-17

**Example 14–29  Creating a TopicBrowser for AdtMessage Messages, Locking Messages**

```java
/* Create a browser for a Topic with AdtMessage messages of type EMPLOYEE* in lock mode*/
TopicSession jms_session
TopicBrowser browser;
Topic test_topic;
browser = ((AQjmsSession) jms_session).createBrowser(
    test_topic, "SUBS1", Employee.getFactory(), true);
```

**Browsing Messages Using a TopicBrowser**

```java
public void purgeSeen()
    throws JMSException
```

This method browses messages using a TopicBrowser. Use methods in java.util.Enumeration to go through the list of messages. Use the method purgeSeen in TopicBrowser to purge messages that have been seen during the current browse.

**Example 14–30  Creating a TopicBrowser with a Specified Selector**

```java
/* Create a browser for topics with a specified selector */
public void browse_rush_orders(TopicSession jms_session)
TopicBrowser browser;
Topic topic;
ObjectMessage obj_message
BoOrder new_order;
Enumeration messages;
/* get a handle to the new_orders topic */
topic = ((AQjmsSession) jms_session).getTopic("OE", "OE_bookedorders_topic");
/* create a Browser to look at RUSH orders */
browser = ((AQjmsSession) jms_session).createBrowser(
    topic, "SUBS1", JMSCorrelationID = 'RUSH');
```
/* Browse through the messages */
for (messages = browser.elements() ; message.hasMoreElements();)
    {obj_message = (ObjectMessage)message.nextElement();}
/* Purge messages seen during this browse */
browser.purgeSeen()
This chapter describes the Java Message Service (JMS) operational interface (shared interfaces) to Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- Oracle® Database JMS Operational Interface: Shared Interfaces
- Specifying JMS Message Properties
- Setting Default TimeToLive for All Messages Sent by a MessageProducer
- Setting Default Priority for All Messages Sent by a MessageProducer
- Creating an AQjms Agent
- Receiving a Message Synchronously
- Specifying the Navigation Mode for Receiving Messages
- Receiving a Message Asynchronously
- Getting Message ID
- Getting JMS Message Properties
- Closing and Shutting Down
- Troubleshooting

Oracle® Database JMS Operational Interface: Shared Interfaces

This section discusses Oracle® Database shared interfaces for JMS operations.

This section contains these topics:

- Starting a JMS Connection
- Getting a JMS Connection
- Committing All Operations in a Session
- Rolling Back All Operations in a Session
- Getting the JDBC Connection from a Session
- Getting the OracleOCIConnectionPool from a JMS Connection
- Creating a BytesMessage
- Creating a MapMessage
- Creating a StreamMessage
- Creating an ObjectMessage
- Creating a TextMessage
- Creating a JMS Message
- Creating an AdtMessage
- Setting JMS Correlation Identifier

### Starting a JMS Connection

```java
class SampleClass {
    public void start() throws JMSException {
        AQjmsConnection.start() starts a JMS connection for receiving messages.
    }
}
```

### Getting a JMS Connection

```java
class SampleClass {
    public oracle.jms.AQjmsConnection getJmsConnection() throws JMSException {
        AQjmsSession.getJmsConnection() gets a JMS connection from a session.
    }
}
```

### Committing All Operations in a Session

```java
class SampleClass {
    public void commit() throws JMSException {
        AQjmsSession.commit() commits all JMS and SQL operations performed in a session.
    }
}
```

### Rolling Back All Operations in a Session

```java
class SampleClass {
    public void rollback() throws JMSException {
        AQjmsSession.rollback() terminates all JMS and SQL operations performed in a session.
    }
}
```

### Getting the JDBC Connection from a Session

```java
class SampleClass {
    public java.sql.Connection getDBConnection() throws JMSException {
        AQjmsSession.getDBConnection() gets the underlying JDBC connection from a JMS session. The JDBC connection can be used to perform SQL operations as part of the same transaction in which the JMS operations are accomplished.
    }
}
```

#### Example 15-1 Getting Underlying JDBC Connection from JMS Session

```java
java.sql.Connection db_conn;
QueueSession jms_sess;
db_conn = ((AQjmsSession)jms_sess).getDBConnection();
```

### Getting the OracleOCIConnectionPool from a JMS Connection

```java
class SampleClass {
    public oracle.jdbc.pool.OracleOCIConnectionPool getOCICConnectionPool() {
    }
}
```
AQjmsConnection.getOCIConnectionPool() gets the underlying OracleOCIConnectionPool from a JMS connection. The settings of the OracleOCIConnectionPool instance can be tuned by the user depending on the connection usage, for example, the number of sessions the user wants to create using the given connection. The user should not, however, close the OracleOCIConnectionPool instance being used by the JMS connection.

**Example 15–2 Getting Underlying OracleOCIConnectionPool from JMS Connection**

```java
oracle.jdbc.pool.OracleOCIConnectionPool cpool;
QueueConnection jms_conn;
cpool = ((AQjmsConnection)jms_conn).getOCIConnectionPool();
```

**Creating a BytesMessage**

```java
public javax.jms.BytesMessage createBytesMessage()
    throws JMSException

AQjmsSession.createBytesMessage() creates a bytes message. It can be used only if the queue table that contains the destination queue/topic was created with the SYS.AQ$_JMS_BYTES_MESSAGE or AQ$_JMS_MESSAGE payload types.
```

**Creating a MapMessage**

```java
public javax.jms.MapMessage createMapMessage()
    throws JMSException

AQjmsSession.createMapMessage() creates a map message. It can be used only if the queue table that contains the destination queue/topic was created with the SYS.AQ$_JMS_MAP_MESSAGE or AQ$_JMS_MESSAGE payload types.
```

**Creating a StreamMessage**

```java
public javax.jms.StreamMessage createStreamMessage()
    throws JMSException

AQjmsSession.createStreamMessage() creates a stream message. It can be used only if the queue table that contains the destination queue/topic was created with the SYS.AQ$_JMS_STREAM_MESSAGE or AQ$_JMS_MESSAGE payload types.
```

**Creating an ObjectMessage**

```java
public javax.jms.ObjectMessage createObjectMessage(java.io.Serializable object)
    throws JMSException

AQjmsSession.createObjectMessage() creates an object message. It can be used only if the queue table that contains the destination queue/topic was created with the SYS.AQ$_JMS_OBJECT_MESSAGE or AQ$_JMS_MESSAGE payload types.
```

**Creating a TextMessage**

```java
public javax.jms.TextMessage createTextMessage()
    throws JMSException

AQjmsSession.createTextMessage() creates a text message. It can be used only if the queue table that contains the destination queue/topic was created with the SYS.AQ$_JMS_TEXT_MESSAGE or AQ$_JMS_MESSAGE payload types.
```
Creating a JMS Message

```java
public javax.jms.Message createMessage()
    throws JMSException

AQjmsSession.createMessage() creates a JMS message. You can use the AQ$_JMS_MESSAGE construct message to construct messages of different types. The message type must be one of the following:
- DBMS_AQ.JMS_TEXT_MESSAGE
- DBMS_AQ.JMS_OBJECT_MESSAGE
- DBMS_AQ.JMS_MAP_MESSAGE
- DBMS_AQ.JMS_BYTES_MESSAGE
- DBMS_AQ.JMS_STREAM_MESSAGE

You can also use this ADT to create a header-only JMS message.
```

Creating an AdtMessage

```java
public oracle.jms.AdtMessage createAdtMessage()
    throws JMSException

AQjmsSession.createAdtMessage() creates an AdtMessage. It can be used only if the queue table that contains the queue/topic was created with an Oracle ADT payload type. An AdtMessage must be populated with an object that implements the CustomDatum interface. This object must be the Java mapping of the SQL ADT defined as the payload for the queue/topic. Java classes corresponding to SQL ADT types can be generated using the Jpublisher tool.
```

Setting JMS Correlation Identifier

```java
public void setJMSCorrelationID(java.lang.String correlationID)
    throws JMSException

AQjmsMessage.setJMSCorrelationID() specifies the message correlation identifier.
```

Specifying JMS Message Properties

Property names starting with JMS are provider-specific. User-defined properties cannot start with JMS.

The following provider properties can be set by clients using text, stream, object, bytes or map messages:
- JMSXAppID (string)
- JMSXGroupID (string)
- JMSXGroupSeq (int)
- JMS_OracleExcpQ (string)
  This message property specifies the exception queue.
- JMS_OracleDelay (int)
  This message property specifies the message delay in seconds.

The following properties can be set on AdtMessage
Specifying JMS Message Properties

- **JMS_OracleExcpQ (String)**
  This message property specifies the exception queue as "schema.queue_name"

- **JMS_OracleDelay (int)**
  This message property specifies the message delay in seconds.

This section contains these topics:

- Setting a Boolean Message Property
- Setting a String Message Property
- Setting an Integer Message Property
- Setting a Double Message Property
- Setting a Float Message Property
- Setting a Byte Message Property
- Setting a Long Message Property
- Setting a Short Message Property
- Setting an Object Message Property

### Setting a Boolean Message Property

```
public void setBooleanProperty(java.lang.String name,
                                boolean value)
    throws JMSException
```

`AQjmsMessage.setBooleanProperty()` specifies a message property as Boolean. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the Boolean property</td>
</tr>
<tr>
<td>value</td>
<td>Boolean property value to set in the message</td>
</tr>
</tbody>
</table>

### Setting a String Message Property

```
public void setStringProperty(java.lang.String name,
                               java.lang.String value)
    throws JMSException
```

`AQjmsMessage.setStringProperty()` specifies a message property as string. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the string property</td>
</tr>
<tr>
<td>value</td>
<td>String property value to set in the message</td>
</tr>
</tbody>
</table>

### Setting an Integer Message Property

```
public void setIntProperty(java.lang.String name,
                            int value)
    throws JMSException
```

Specifying JMS Message Properties

**AQjmsMessage.setIntProperty()** specifies a message property as integer. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the integer property</td>
</tr>
<tr>
<td>value</td>
<td>Integer property value to set in the message</td>
</tr>
</tbody>
</table>

**Setting a Double Message Property**

```java
public void setDoubleProperty(java.lang.String name, double value)
    throws JMSException
```

**AQjmsMessage.setDoubleProperty()** specifies a message property as double. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the double property</td>
</tr>
<tr>
<td>value</td>
<td>Double property value to set in the message</td>
</tr>
</tbody>
</table>

**Setting a Float Message Property**

```java
public void setFloatProperty(java.lang.String name, float value)
    throws JMSException
```

**AQjmsMessage.setFloatProperty()** specifies a message property as float. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the float property</td>
</tr>
<tr>
<td>value</td>
<td>Float property value to set in the message</td>
</tr>
</tbody>
</table>

**Setting a Byte Message Property**

```java
public void setByteProperty(java.lang.String name, byte value)
    throws JMSException
```

**AQjmsMessage.setByteProperty()** specifies a message property as byte. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the byte property</td>
</tr>
<tr>
<td>value</td>
<td>Byte property value to set in the message</td>
</tr>
</tbody>
</table>

**Setting a Long Message Property**

```java
public void setLongProperty(java.lang.String name, long value)
    throws JMSException
```

**AQjmsMessage.setLongProperty()** specifies a message property as long. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the long property</td>
</tr>
<tr>
<td>value</td>
<td>Long property value to set in the message</td>
</tr>
</tbody>
</table>
AQ.jmsMessage.setLongProperty() specifies a message property as long. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the long property</td>
</tr>
<tr>
<td>value</td>
<td>Long property value to set in the message</td>
</tr>
</tbody>
</table>

Setting a Short Message Property

public void setShortProperty(java.lang.String name, short value) throws JMSException

AQ.jmsMessage.setShortProperty() specifies a message property as short. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the short property</td>
</tr>
<tr>
<td>value</td>
<td>Short property value to set in the message</td>
</tr>
</tbody>
</table>

Setting an Object Message Property

public void setObjectProperty(java.lang.String name, java.lang.Object value) throws JMSException

AQ.jmsMessage.setObjectProperty() specifies a message property as object. Only objectified primitive values are supported: Boolean, byte, short, integer, long, float, double and string. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the Java object property</td>
</tr>
<tr>
<td>value</td>
<td>Java object property value to set in the message</td>
</tr>
</tbody>
</table>

Setting Default TimeToLive for All Messages Sent by a MessageProducer

public void setTimeToLive(long timeToLive) throws JMSException

This method sets the default TimeToLive for all messages sent by a MessageProducer. It is calculated after message delay has taken effect. This method has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeToLive</td>
<td>Message time to live in milliseconds (zero is unlimited)</td>
</tr>
</tbody>
</table>

Example 15–3 Setting Default TimeToLive for All Messages Sent by a MessageProducer

/* Set default timeToLive value to 100000 milliseconds for all messages sent by the QueueSender*/
QueueSender sender;
sender.setTimeToLive(100000);
Setting Default Priority for All Messages Sent by a MessageProducer

```java
public void setPriority(int priority)
    throws JMSException
```

This method sets the default `Priority` for all messages sent by a `MessageProducer`. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Message priority for this message producer. The default is 4.</td>
</tr>
</tbody>
</table>

Priority values can be any integer. A smaller number indicates higher priority. If a priority value is explicitly specified during a `send()` operation, then it overrides the default value set by this method.

**Example 15–4 Setting Default Priority Value for All Messages Sent by QueueSender**

/* Set default priority value to 2 for all messages sent by the QueueSender*/
QueueSender sender;
sender.setPriority(2);

**Example 15–5 Setting Default Priority Value for All Messages Sent by TopicPublisher**

/* Set default priority value to 2 for all messages sent by the TopicPublisher*/
TopicPublisher publisher;
publisher.setPriority(1);

Creating an AQjms Agent

```java
public void createAQAgent(java.lang.String agent_name,
    boolean enable_http,
    throws JMSException
```

This method creates an `AQjmsAgent`. It has the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_name</td>
<td>Name of the AQ agent</td>
</tr>
<tr>
<td>enable_http</td>
<td>If set to true, then this agent is allowed to access AQ through HTTP</td>
</tr>
</tbody>
</table>

Receiving a Message Synchronously

You can receive a message synchronously by specifying `Timeout` or without waiting. You can also receive a message using a transformation:

- Using a Message Consumer by Specifying Timeout
- Using a Message Consumer Without Waiting
- Receiving Messages from a Destination Using a Transformation

Using a Message Consumer by Specifying Timeout

```java
public javax.jms.Message receive(long timeout)
    throws JMSException
```
This method receives a message using a message **consumer** by specifying timeout.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Timeout value in milliseconds</td>
</tr>
</tbody>
</table>

**Example 15–6 Using a Message Consumer by Specifying Timeout**

```java
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession t_sess = null;
TopicSession jms_sess;
Topic shipped_orders;
int myport = 5521;

/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
    "WS", "Shipped_Orders_Topic");

/* create a subscriber, specifying the correct CustomDatumFactory and selector */
subscriber1 = jms_sess.createDurableSubscriber(
    shipped_orders, 'WesternShipping',
    ' priority > 1 and tab.user_data.region like 'WESTERN %''
    false, AQjmsAgent.getFactory());
/* receive, blocking for 30 seconds if there were no messages */
Message = subscriber.receive(30000);
```

**Example 15–7 JMS: Blocking Until a Message Arrives**

```java
public BolOrder get_new_order1(QueueSession jms_session)
{
    Queue queue;
    QueueReceiver qrec;
    ObjectMessage obj_message;
    BolCustomer customer;
    BolOrder new_order = null;
    String state;
    try
    {
        /* get a handle to the new_orders queue */
        queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
        qrec = jms_session.createReceiver(queue);
        /* wait for a message to show up in the queue */
        obj_message = (ObjectMessage)qrec.receive();
        new_order = (BolOrder)obj_message.getObject();
        customer = new_order.getCustomer();
        state = customer.getState();
        System.out.println("Order:  for customer " + customer.getName());
    }
    catch (JMSException ex)
    {
        System.out.println("Exception: " + ex);
    }
```
Using a Message Consumer Without Waiting

public javax.jms.Message receiveNoWait() throws JMSException

This method receives a message using a message consumer without waiting.

**Example 15–8  JMS: Nonblocking Messages**

```java
public BolOrder poll_new_order3(QueueSession jms_session)
{
    Queue            queue;
    QueueReceiver    qrec;
    ObjectMessage    obj_message;
    BolCustomer      customer;
    BolOrder         new_order = null;
    String           state;

    try
    {
        /* get a handle to the new_orders queue */
        queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
        qrec = jms_session.createReceiver(queue);

        /* check for a message to show in the queue */
        obj_message = (ObjectMessage)qrec.receiveNoWait();
        new_order = (BolOrder)obj_message.getObject();
        customer = new_order.getCustomer();
        state    = customer.getState();

        System.out.println("Order: for customer " + customer.getName());
    }
    catch (JMSException ex)
    {
        System.out.println("Exception: " + ex);
    }
    return new_order;
}
```

Receiving Messages from a Destination Using a Transformation

A transformation can be applied when receiving a message from a queue or topic. The transformation is applied to the message before returning it to JMS application.

The transformation can be specified using the `setTransformation()` interface of the `AQjmsQueueReceiver`, `AQjmsTopicSubscriber` or `AQjmsTopicReceiver`.

**Example 15–9  JMS: Receiving Messages from a Destination Using a Transformation**

Assume that the Western Shipping application retrieves messages from the OE_bookedorders_topic. It specifies the transformation `OE2WS` to retrieve the message as the Oracle object type `WS_order`. Assume that the `WSOrder` Java class has been generated by Jpublisher to map to the Oracle object `WS.WS_order`:

```java
public AQjmsAdtMessage retrieve_bookedorders(TopicSession jms_session)
AQjmsTopicReceiver  receiver;
Topic               topic;
```
Specifying the Navigation Mode for Receiving Messages

```java
public void setNavigationMode(int mode)
    throws JMSException

This method specifies the navigation mode for receiving messages. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>New value of the navigation mode</td>
</tr>
</tbody>
</table>

Example 15–10  Specifying Navigation Mode for Receiving Messages

```java
try {
    /* get a handle to the OE_bookedorders_topic */
    topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");

    /* Create a receiver for WShip */
    receiver = ((AQjmsSession)jms_session).createTopicReceiver(
        topic, "WShip, null, WSOrder.getFactory()");

    /* set the transformation in the publisher */
    receiver.setTransformation("OE2WS");
    msg = receiver.receive(10);
}
catch (JMSException ex) {
    System.out.println("Exception :", ex);
}
return (AQjmsAdtMessage)msg;
```
Receiving a Message Asynchronously

You can receive a message asynchronously two ways:

- Specifying a Message Listener at the Message Consumer
- Specifying a Message Listener at the Session

Specifying a Message Listener at the Message Consumer

public void setMessageListener(javax.jms.MessageListener myListener)
throws JMSException

This method specifies a message listener at the message consumer. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myListener</td>
<td>Sets the consumer message listener</td>
</tr>
</tbody>
</table>

Example 15–11 Specifying Message Listener at Message Consumer

```java
TopicConnectionFactory tc_fact = null;
TopicConnection t_conn = null;
TopicSession t_sess = null;
TopicSession jms_sess;
Topic shipped_orders;
int myport = 5521;
MessageListener mLis = null;

/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(  
    "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(  
    "WS", "Shipped_Orders_Topic");

/* create a subscriber, specifying the correct CustomDatumFactory and selector */
subscriber1 = jms_sess.createDurableSubscriber(  
    shipped_orders, 'WesternShipping',  
    "priority > 1 and tab.user_data.region like 'WESTERN %'",  
    false, AQjmsAgent.getFactory());
mLis = new myListener(jms_sess, 'foo');

/* get message for the subscriber, returning immediately if there was no message */
subscriber.setMessageListener(mLis);

The definition of the myListener class
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import java.util.*;
public class myListener implements MessageListener
{
    TopicSession mySess;
    String myName;
    /* constructor */
    myListener(TopicSession t_sess, String t_name)
    {
        mySess = t_sess;
    }
```
myName = t_name;
}
public void onMessage(Message m)
{
    System.out.println("Retrieved message with correlation: " ||
    m.getJMSCorrelationID());
    try{
        /* commit the dequeue */
        mySession.commit();
    } catch (java.sql.SQLException e)
    {System.out.println("SQL Exception on commit"); }
}

**Specifying a Message Listener at the Session**

public void setMessageListener(javax.jms.MessageListener listener)
    throws JMSException

This method specifies a message listener at the session.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listener</td>
<td>Message listener to associate with this session</td>
</tr>
</tbody>
</table>

**Getting Message ID**

This section contains these topics:

- Getting the Correlation Identifier
- Getting the Message Identifier

**Getting the Correlation Identifier**

public java.lang.String getJMSCorrelationID()
    throws JMSException

AQjmsMessage.getJMSCorrelationID() gets the correlation identifier of a message.

**Getting the Message Identifier**

public byte[] getJMSCorrelationIDAsBytes()
    throws JMSException

AQjmsMessage.getJMSCorrelationIDAsBytes() gets the message identifier of a message as bytes or a string.

**Getting JMS Message Properties**

This section contains these topics:

- Getting a Boolean Message Property
- Getting a String Message Property
- Getting an Integer Message Property
- Getting a Double Message Property
Getting JMS Message Properties

- Getting a Float Message Property
- Getting a Byte Message Property
- Getting a Long Message Property
- Getting a Short Message Property
- Getting an Object Message Property

Getting a Boolean Message Property

```java
public boolean getBooleanProperty(java.lang.String name) throws JMSException
AQjmsMessage.getBooleanProperty() gets a message property as Boolean. It has the following parameter:
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the Boolean property</td>
</tr>
</tbody>
</table>

Getting a String Message Property

```java
public java.lang.String getStringProperty(java.lang.String name) throws JMSException
AQjmsMessage.getStringProperty() gets a message property as string. It has the following parameter:
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the string property</td>
</tr>
</tbody>
</table>

Getting an Integer Message Property

```java
public int getIntProperty(java.lang.String name) throws JMSException
AQjmsMessage.getIntProperty() gets a message property as integer. It has the following parameter:
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the integer property</td>
</tr>
</tbody>
</table>

Getting a Double Message Property

```java
public double getDoubleProperty(java.lang.String name) throws JMSException
AQjmsMessage.getDoubleProperty() gets a message property as double. It has the following parameter:
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the double property</td>
</tr>
</tbody>
</table>
Getting a Float Message Property

public float getFloatProperty(java.lang.String name)
    throws JMSException

AQjmsMessage.getFloatProperty() gets a message property as float. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the float property</td>
</tr>
</tbody>
</table>

Getting a Byte Message Property

public byte getByteProperty(java.lang.String name)
    throws JMSException

AQjmsMessage.getByteProperty() gets a message property as byte. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the byte property</td>
</tr>
</tbody>
</table>

Getting a Long Message Property

public long getLongProperty(java.lang.String name)
    throws JMSException

AQjmsMessage.getLongProperty() gets a message property as long. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the long property</td>
</tr>
</tbody>
</table>

Getting a Short Message Property

public short getShortProperty(java.lang.String name)
    throws JMSException

AQjmsMessage.getShortProperty() gets a message property as short. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the short property</td>
</tr>
</tbody>
</table>

Getting an Object Message Property

public java.lang.Object getObjectProperty(java.lang.String name)
    throws JMSException

AQjmsMessage.getObjectProperty() gets a message property as object. It has the following parameter:
Closing and Shutting Down

This section contains these topics:

- Closing a MessageProducer
- Closing a Message Consumer
- Stopping a JMS Connection
- Closing a JMS Session
- Closing a JMS Connection

Closing a MessageProducer

```java
public void close()
    throws JMSException

AQjmsProducer.close() closes a MessageProducer.
```

Closing a Message Consumer

```java
public void close()
    throws JMSException

AQjmsConsumer.close() closes a message consumer.
```

Stopping a JMS Connection

```java
public void stop()
    throws JMSException

AQjmsConnection.stop() stops a JMS connection.
```

Closing a JMS Session

```java
public void close()
    throws JMSException

AQjmsSession.close() closes a JMS session.
```

Closing a JMS Connection

```java
public void close()
    throws JMSException

AQjmsConnection.close() closes a JMS connection and releases all resources allocated on behalf of the connection. Because the JMS provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close
```

### Example 15–12 Getting Message Property as an Object

```java
TextMessage message;
message.getObjectProperty("empid", new Integer(1000));
```
them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Troubleshooting

This section contains these topics:

- Getting a JMS Error Code
- Getting a JMS Error Number
- Getting an Exception Linked to a JMS Exception
- Printing the Stack Trace for a JMS Exception
- Setting an Exception Listener
- Getting an Exception Listener

Getting a JMS Error Code

```java
public java.lang.String getErrorCode()
```

`AQjmsException.getErrorCode()` gets the error code for a JMS exception.

Getting a JMS Error Number

```java
public int getErrorNumber()
```

`AQjmsException.getErrorNumber()` gets the error number for a JMS exception.

**Note:** This method will be deprecated in a future release. Use `getErrorCode()` instead.

Getting an Exception Linked to a JMS Exception

```java
public java.lang.String getLinkString()
```

`AQjmsException.getLinkString()` gets the exception linked to a JMS exception. In general, this contains the SQL exception raised by the database.

Printing the Stack Trace for a JMS Exception

```java
public void printStackTrace(java.io.PrintStream s)
```

`AQjmsException.printStackTrace()` prints the stack trace for a JMS exception.

Setting an Exception Listener

```java
public void setExceptionListener(javax.jms.ExceptionListener listener)
throws JMSException
```

`AQjmsConnection.setExceptionListener()` specifies an exception listener for a connection. It has the following parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listener</td>
<td>Exception listener</td>
</tr>
</tbody>
</table>
If an exception listener has been registered, then it is informed of any serious problem detected for a connection. This is accomplished by calling the listener onException() method, passing it a JMS exception describing the problem. This allows a JMS client to be notified of a problem asynchronously. Some connections only consume messages, so they have no other way to learn the connection has failed.

**Example 15–13 Specifying Exception Listener for Connection**

```java
//register an exception listener  
Connection jms_connection;  
jms_connection.setExceptionListener(
    new ExceptionListener() {
      public void onException (JMSException jmsException) {
        System.out.println("JMS-EXCEPTION: "+ jmsException.toString());
      }
    });
```

**Getting an Exception Listener**

```java
public javax.jms.ExceptionListener getExceptionListener()
  
  throws JMSException

AQ.jmsConnection.getExceptionListener() gets the exception listener for the connection.

**Example 15–14 Getting the Exception Listener for the Connection**

```java
//Get the exception listener  
Connection jms_connection;  
ExceptionListener el = jms_connection.getExceptionListener();
```
This chapter provides examples that illustrate how to use Oracle JMS Types to dequeue and enqueue Oracle Streams Advanced Queuing (AQ) messages.

The chapter contains the following topics:
- How to Run the Oracle® Database JMS Type Examples
- JMS BytesMessage Examples
- JMS StreamMessage Examples
- JMS MapMessage Examples
- More Oracle® Database JMS Examples

### How to Run the Oracle® Database JMS Type Examples

To run Example 16–2 through Example 16–7 follow these steps:

1. Copy and save Example 16–1 as setup.sql.
2. Run setup.sql as follows:
   ```sql
   sqlplus /NOLOG @setup.sql
   ```
3. Log in to SQL*Plus as jmsuser/jmsuser.
4. Run the corresponding pair of SQL scripts for each type of message.
   For JMS BytesMessage, for example, run Example 16–2 on page 16-5 and Example 16–3 on page 16-7.
5. Ensure that your database parameter java_pool-size is large enough. For example, you can use java_pool_size=20M.

### Setting Up the Examples

Example 16–1 performs the necessary setup for the JMS types examples. Copy and save it as setup.sql.

**Example 16–1  Setting Up Environment for Running JMS Types Examples**

```sql
connect sys;
enter password: password

Rem
Rem Create the JMS user: jmsuser
Rem
```
DROP USER jmsuser CASCADE;
CREATE USER jmsuser IDENTIFIED BY jmsuser;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_FLSQL TO jmsuser;

set echo off
set verify off

connect sys

DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password

Rem
Rem Creating five AQ queue tables and five queues for five payloads:
Rem SYS.AQ$_JMS_TEXT_MESSAGE
Rem SYS.AQ$_JMS_BYTES_MESSAGE
Rem SYS.AQ$_JMS_STREAM_MESSAGE
Rem SYS.AQ$_JMS_MAP_MESSAGE
Rem SYS.AQ$_JMS_MESSAGE

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_text',
Queue_payload_type => 'SYS.AQ$_JMS_TEXT_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_bytes',
Queue_payload_type => 'SYS.AQ$_JMS_BYTES_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_stream',
Queue_payload_type => 'SYS.AQ$_JMS_STREAM_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_map',
Queue_payload_type => 'SYS.AQ$_JMS_MAP_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_general',
Queue_payload_type => 'SYS.AQ$_JMS_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_text_que',
Queue_table => 'jmsuser.jms_qtt_text');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_bytes_que',
Queue_table => 'jmsuser.jms_qtt_bytes');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_stream_que',
Queue_table => 'jmsuser.jms_qtt_stream');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_map_que',
Queue_table => 'jmsuser.jms_qtt_map');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_general_que',
Queue_table => 'jmsuser.jms_qtt_general');

Rem
Rem Starting the queues and enable both enqueue and dequeue
Rem
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_text_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_bytes_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_stream_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_map_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_general_que');

Rem The supporting utility used in the example to help display results in SQLPLUS enviroment

Rem
Rem Display a RAW data in SQLPLUS
Rem
create or replace procedure display_raw(rdata raw)
IS
  pos     pls_integer;
  length  pls_integer;
BEGIN
  pos := 1;
  length := UTL_RAW.LENGTH(rdata);
  
  WHILE pos <= length LOOP
    IF pos+20 > length+1 THEN
      dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, length-pos+1));
    ELSE
      dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, 20));
    END IF;
    pos := pos+20;
END LOOP;
END display_raw;
/

show errors;

Rem
Rem Display a BLOB data in SQLPLUS
Rem
create or replace procedure display_blob(bdata blob)
IS
  pos     pls_integer;
  length  pls_integer;
BEGIN
  length := DBMS_LOB.GETLENGTH(bdata);
  pos := 1;
  WHILE pos <= length LOOP
    display_raw(DBMS_LOB.SUBSTR(bdata, 2000, pos));
    pos := pos+2000;
END LOOP;
END display_blob;
/

show errors;

Rem
Rem Display a VARCHAR data in SQLPLUS
Rem
create or replace procedure display_varchar(vdata varchar)
IS
  pos     pls_integer;
  text_len pls_integer;
BEGIN
  
  WHILE pos < length LOOP
    IF pos+20 > length+1 THEN
      dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, length-pos+1));
    ELSE
      dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, 20));
    END IF;
    pos := pos+20;
END LOOP;
END display_varchar;
/

show errors;
BEGIN
    text_len := length(vdata);
pos := 1;
    WHILE pos <= text_len LOOP
        IF pos+20 > text_len+1 THEN
            dbms_output.put_line(SUBSTR(vdata, pos, text_len-pos+1));
        ELSE
            dbms_output.put_line(SUBSTR(vdata, pos, 20));
        END IF;
pos := pos+20;
    END LOOP;
END display_varchar;
/

show errors;

Rem
Rem Display a CLOB data in SQLPLUS
Rem
create or replace procedure display_clob(cdata clob)
IS
    pos                 pls_integer;
    length              pls_integer;
BEGIN
    length := dbms_lob.getlength(cdata);
pos := 1;
    WHILE pos <= length LOOP
        display_varchar(DBMS_LOB.SUBSTR(cdata, 2000, pos));
pos := pos+2000;
    END LOOP;
END display_clob;
/

show errors;

Rem
Rem Display a SYS.AQ$_JMS_EXCEPTION data in SQLPLUS
Rem
Rem When application receives an ORA-24197 error, It means the JAVA stored
Rem procedures has thrown some exceptions that could not be categorized. The
Rem user can use GET_EXCEPTION procedure of SYS.AQ$_JMS_BYTES_MESSAGE,
Rem SYS.AQ$_JMS_STREAM_MESSAG or SYS.AQ$_JMS_MAP_MESSAGE
Rem to retrieve a SYS.AQ$_JMS_EXCEPTION object which contains more detailed
Rem information on this JAVA exception including the exception name, JAVA error
Rem message and stack trace.
Rem
Rem This utility function is to help display the SYS.AQ$_JMS_EXCEPTION object in
Rem SQLPLUS
Rem
create or replace procedure display_exp(exp SYS.AQ$_JMS_EXCEPTION)
IS
    pos1                pls_integer;
    pos2                pls_integer;
    text_data           varchar(2000);
BEGIN
    dbms_output.put_line('exception:'||exp.exp_name);
    dbms_output.put_line('err_msg:'||exp.err_msg);
    dbms_output.put_line('stack:'||length(exp.stack));
pos1 := 1;
LOOP
  pos2 := INSTR(exp.stack, chr(10), pos1);
  IF pos2 = 0 THEN
    pos2 := length(exp.stack)+1;
  END IF;
  dbms_output.put_line(SUBSTR(exp.stack, pos1, pos2-pos1));
  IF pos2 > length(exp.stack) THEN
    EXIT;
  END IF;
  pos1 := pos2+1;
END LOOP;

END display_exp;
/

show errors;
EXIT;

JMS BytesMessage Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS BytesMessage.

Example 16–2 shows how to use JMS type member functions with DBMS_AQ functions to populate and enqueue a JMS BytesMessage represented as sys.aq$_.jms_bytes_message type in the database. This message later can be dequeued by a JAVA Oracle Java Message Service (OJMS) client.

Example 16–2  Populating and Enqueuing a BytesMessage

set echo off
set verify off
connect sys

DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password

SET ECHO ON
set serveroutput on

DECLARE
  id                 pls_integer;
  agent              sys.aq$_.agent := sys.aq$_.agent(' ', null, 0);
  message            sys.aq$_.jms_bytes_message;
enqueue_options dbms_aq.enqueue_options_t;
message_properties dbms_aq.message_properties_t;
msgid raw(16);

java_exp exception;
pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN

-- Construct a empty BytesMessage object
message := sys.aq$_jms_bytes_message.construct;

-- Shows how to set the JMS header
message.set_replyto(agent);
message.set_type('tkaqpet1');
message.set_userid('jmsuser');
message.set_appid('plsql_enq');
message.set_groupid('st');
message.set_groupseq(1);

-- Shows how to set JMS user properties
message.set_string_property('color', 'RED');
message.set_int_property('year', 1999);
message.set_float_property('price', 16999.99);
message.set_long_property('mileage', 300000);
message.set_boolean_property('import', True);
message.set_byte_property('password', -127);

-- Shows how to populate the message payload of aq$_jms_bytes_message
-- Passing -1 reserve a new slot within the message store of sys.aq$_jms_bytes_message.
-- The maximum number of sys.aq$_jms_bytes_message type of messages to be operated at
-- the same time within a session is 20. Calling clean_body function with parameter -1
-- might result a ORA-24199 error if the messages currently operated is already 20.
-- The user is responsible to call clean or clean_all function to clean up message store.
id := message.clear_body(-1);

-- Write data into the BytesMessage payload. These functions are analogy of JMS JAVA api's.
-- See the document for detail.

-- Write a byte to the BytesMessage payload
message.write_byte(id, 10);

-- Write a RAW data as byte array to the BytesMessage payload
message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

-- Write a portion of the RAW data as byte array to BytesMessage payload
-- Note the offset follows JAVA convention, starting from 0
message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF'))[0, 16]);

-- Write a char to the BytesMessage payload
message.write_char(id, 'A');

-- Write a double to the BytesMessage payload
message.write_double(id, 9999.99);
-- Write a float to the BytesMessage payload
message.write_float(id, 99.99);
-- Write a int to the BytesMessage payload
message.write_int(id, 12345);
-- Write a long to the BytesMessage payload
message.write_long(id, 1234567);
-- Write a short to the BytesMessage payload
message.write_short(id, 123);
-- Write a String to the BytesMessage payload,
-- the String is encoded in UTF8 in the message payload
message.write_utf(id, 'Hello World!');
-- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
-- Without doing this, the PL/SQL message is still empty.
message.flush(id);
-- Use either clean_all or clean to clean up the message store when the user
-- do not plan to do payload population on this message anymore
sys.aq$_jms_bytes_message.clean_all();
--message.clean(id);
-- Enqueue this message into AQ queue using DBMS_AQ package
dbms_aq.enqueue(queue_name => 'jmsuser.jms_bytes_que',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => msgid);

EXCEPTION
WHEN java_exp THEN
   dbms_output.put_line('exception information:');
   display_exp(sys.aq$_jms_stream_message.get_exception());
END;
/
commit;

**Example 16–3** illustrates how to use JMS type member functions with DBMS_AQ functions to dequeue and retrieve data from a JMS BytesMessage represented as sys.aq$_jms_bytes_message type in the database. This message might be enqueued by a Java OJMS client.

**Example 16–3  Dequeuing and Retrieving JMS BytesMessage Data**

set echo off
set verify off
connect sys
DROP USER jmsuser CASCADE;
ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE
CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
set echo on
set serveroutput on size 20000

DECLARE
  id                 pls_integer;
  blob_data          blob;
  clob_data          clob;
  blob_len           pls_integer;
  message            sys.aq$.jms_bytes_message;
  agent              sys.aq$._agent;
  dequeue_options    dbms_aq.dequeue_options_t;
  message_properties dbms_aq.message_properties_t;
  msgid raw(16);
  gdata              sys.aq$._jms_value;

  java_exp            exception;
  pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
  DBMS_OUTPUT_ENABLE (20000);
  -- Dequeue this message from AQ queue using DBMS_AQ package
  dbms_aq.dequeue(queue_name => 'jmsuser.jms_bytes_que',
                   dequeue_options => dequeue_options,
                   message_properties => message_properties,
                   payload => message,
                   msgid => msgid);

  -- Retrieve the header
  agent := message.get_replyto;
  dbms_output.put_line('Type: ' || message.get_type ||
                        ' UserId: ' || message.get_userid ||
                        ' AppId: ' || message.get_appid ||
                        ' GroupId: ' || message.get_groupid ||
                        ' GroupSeq: ' || message.get_groupseq);

  -- Retrieve the user properties
  dbms_output.put_line('price: ' || message.get_float_property('price'));
  dbms_output.put_line('color: ' || message.get_string_property('color'));
  IF message.get_boolean_property('import') = TRUE THEN
    dbms_output.put_line('import: Yes');
  ELSIF message.get_boolean_property('import') = FALSE THEN
    dbms_output.put_line('import: No');
  END IF;
  dbms_output.put_line('year: ' || message.get_int_property('year'));
  dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));
  dbms_output.put_line('password: ' || message.get_byte_property('password'));

  -- Shows how to retrieve the message payload of aq$._jms_bytes_message
  -- Prepare call, send the content in the PL/SQL aq$._jms_bytes_message object to
-- Java stored procedure (Jserv) in the form of a byte array.
-- Passing -1 reserves a new slot in msg store of sys.aq$_jms_bytes_message.
-- Max number of sys.aq$_jms_bytes_message type of messages to be operated at
-- the same time in a session is 20. Call clean_body fn. with parameter -1
-- might result in ORA-24199 error if messages operated on are already 20.
-- You must call clean or clean_all function to clean up message store.
  id := message.prepare(-1);

  -- Read data from BytesMessage payload. These fn.s. are analogy of JMS Java
  -- API's. See the JMS Types chapter for detail.
  dbms_output.put_line('Payload:');

    -- read a byte from the BytesMessage payload
    dbms_output.put_line('read_byte:' || message.read_byte(id));

    -- read a byte array into a blob object from the BytesMessage payload
    dbms_output.put_line('read_bytes:');
    blob_len := message.read_bytes(id, blob_data, 272);
    display_blob(blob_data);

    -- read a char from the BytesMessage payload
    dbms_output.put_line('read_char:'|| message.read_char(id));

    -- read a double from the BytesMessage payload
    dbms_output.put_line('read_double:'|| message.read_double(id));

    -- read a float from the BytesMessage payload
    dbms_output.put_line('read_float:'|| message.read_float(id));

    -- read a int from the BytesMessage payload
    dbms_output.put_line('read_int:'|| message.read_int(id));

    -- read a long from the BytesMessage payload
    dbms_output.put_line('read_long:'|| message.read_long(id));

    -- read a short from the BytesMessage payload
    dbms_output.put_line('read_short:'|| message.read_short(id));

    -- read a String from the BytesMessage payload.
    -- the String is in UTF8 encoding in the message payload
    dbms_output.put_line('read_utf:');
    message.read_utf(id, clob_data);
    display_clob(clob_data);

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do payload retrieving on this message anymore
    message.clean(id);
    -- sys.aq$_jms_bytes_message.clean_all();

  EXCEPTION
    WHEN java_exp THEN
    dbms_output.put_line('exception information:');
    display_exp(sys.aq$_jms_bytes_message.get_exception());

END;
/

commit;
This section includes examples that illustrate enqueuing and dequeuing of a JMS StreamMessage.

Example 16–4 shows how to use JMS type member functions with DBMS_AQ functions to populate and enqueue a JMS StreamMessage represented as sys.aq$_jms_stream_message type in the database. This message later can be dequeued by a JAVA OJMS client.

Example 16–4  Populating and Enqueuing a JMS StreamMessage

set echo off
set verify off
connect sys

DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
SET ECHO ON
set serveroutput on

DECLARE
  id               pls_integer;
  agent            sys.aq$_agent := sys.aq$_agent(' ', null, 0);
  message          sys.aq$_jms_stream_message;
  enqueue_options  dbms_aq.enqueue_options_t;
  message_properties dbms_aq.message_properties_t;
 msgid raw(16);
  java_exp         exception;
  pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
  -- Construct a empty StreamMessage object
  message := sys.aq$_jms_stream_message.construct;

  -- Shows how to set the JMS header
  message.set_replyto(agent);
  message.set_type('tkaqpet1');
  message.set_userid('jmsuser');
  message.set_appid('plsql_enq');
  message.set_groupid('st');
  message.set_groupseq(1);

  -- Shows how to set JMS user properties
  message.set_string_property('color', 'RED');
  message.set_int_property('year', 1999);
  message.set_float_property('price', 16999.99);
  message.set_long_property('mileage', 300000);
message.set_boolean_property('import', True);
message.set_byte_property('password', -127);

-- Shows how to populate the message payload of sys.aq$_jms_stream_message
-- Passing -1 reserve a new slot within the message store of sys.aq$_jms_stream_message.
-- The maximum number of sys.aq$_jms_stream_message type of messages to be operated at
-- the same time within a session is 20. Calling clean_body function with parameter -1
-- might result a ORA-24199 error if the messages currently operated is already 20.
-- The user is responsible to call clean or clean_all function to clean up message store.
id := message.clear_body(-1);

-- Write data into the message payload. These functions are analogy of JMS JAVA api's.
-- See the document for detail.
-- Write a byte to the StreamMessage payload
message.write_byte(id, 10);

-- Write a RAW data as byte array to the StreamMessage payload
message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

-- Write a portion of the RAW data as byte array to the StreamMessage payload
-- Note the offset follows JAVA convention, starting from 0
message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')), 0, 16);

-- Write a char to the StreamMessage payload
message.write_char(id, 'A');

-- Write a double to the StreamMessage payload
message.write_double(id, 9999.99);

-- Write a float to the StreamMessage payload
message.write_float(id, 99.99);

-- Write a int to the StreamMessage payload
message.write_int(id, 12345);

-- Write a long to the StreamMessage payload
message.write_long(id, 1234567);

-- Write a short to the StreamMessage payload
message.write_short(id, 123);

-- Write a String to the StreamMessage payload
message.write_string(id, 'Hello World!');

-- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
-- Without doing this, the PL/SQL message is still empty.
message.flush(id);

-- Use either clean_all or clean to clean up the message store when the user
-- do not plan to do payload population on this message anymore
sys.aq$_jms_stream_message.clean_all();
-- Enqueue this message into AQ queue using DBMS_AQ package
dbms_aq.enqueue(queue_name => 'jmsuser.jms_stream_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => msgid);

EXCEPTION
WHEN java_exc THEN
  dbms_output.put_line('exception information:');
  display_exp(sys.aq$_jms_stream_message.get_exception());
END;
/
commit;

Example 16–5 shows how to use JMS type member functions with DBMS_AQ functions
to dequeue and retrieve data from a JMS StreamMessage represented as sys.aq$_
jms_stream_message type in the database. This message might be enqueued by a
JAVA OJMS client.

Example 16–5  Dequeuing and Retrieving Data From a JMS StreamMessage
set echo off
set verify off
connect sys

DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
set echo on
set serveroutput on

DECLARE

  id                 pls_integer;
  blob_data          blob;
  clob_data          clob;
  message            sys.aq$_jms_stream_message;
  agent              sys.aq$_agent;
  dequeue_options    dbms_aq.dequeue_options_t;
  message_properties dbms_aq.message_properties_t;
  msgid raw(16);
  gdata              sys.aq$_jms_value;
  java_exp           exception;
  pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
DBMS_OUTPUT.ENABLE (20000);

-- Dequeue this message from AQ queue using DBMS_AQ package
dbms_aq.dequeue(queue_name => 'jmsuser.jms_stream_que',
dequeue_options => dequeue_options,
message_properties => message_properties,
payload => message,
msgid => msgid);

-- Retrieve the header
agent := message.get_replyto;

dbms_output.put_line('Type: ' || message.get_type ||
' UserId: ' || message.get_userid ||
' AppId: ' || message.get_appid ||
' GroupId: ' || message.get_groupid ||
' GroupSeq: ' || message.get_groupseq);

-- Retrieve the user properties
dbms_output.put_line('price: ' || message.get_float_property('price'));
dbms_output.put_line('color: ' || message.get_string_property('color'));
IF message.get_boolean_property('import') = TRUE THEN
  dbms_output.put_line('import: Yes');
ELSIF message.get_boolean_property('import') = FALSE THEN
  dbms_output.put_line('import: No');
END IF;

dbms_output.put_line('year: ' || message.get_int_property('year'));
dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));
dbms_output.put_line('password: ' || message.get_byte_property('password'));

-- Shows how to retrieve the message payload of $jms_stream_message
-- The prepare call send the content in the PL/SQL $jms_stream_message
-- object to
-- JAVA stored procedure(Jserv) in the form of byte array.
-- Passing -1 reserve a new slot within the message store of sys.$jms_stream_message.
-- The maximum number of sys.$jms_stream_message type of messages to be
-- operated at
-- the same time within a session is 20. Calling clean_body function with
-- parameter -1
-- might result a ORA-24199 error if the messages currently operated is
-- already 20.
-- The user is responsible to call clean or clean_all function to clean up
-- message store.
id := message.prepare(-1);

-- Assume the users know the types of data in the StreamMessage payload.
-- The user can use the specific read function corresponding with the data
type.
-- These functions are analogy of JMS JAVA api's. See the document for detail.
dbms_output.put_line('Retrieve payload by Type:');

-- Read a byte from the StreamMessage payload
dbms_output.put_line('read_byte: ' || message.read_byte(id));

-- Read a byte array into a blob object from the StreamMessage payload
dbms_output.put_line('read_bytes:');
message.read_bytes(id, blob_data);
display_blob(blob_data);

-- Read another byte array into a blob object from the StreamMessage payload
dbms_output.put_line('read_bytes:');
message.read_bytes(id, blob_data);
display_blob(blob_data);

-- Read a char from the StreamMessage payload
dbms_output.put_line('read_char:'|| message.read_char(id));

-- Read a double from the StreamMessage payload
dbms_output.put_line('read_double:'|| message.read_double(id));

-- Read a float from the StreamMessage payload
dbms_output.put_line('read_float:'|| message.read_float(id));

-- Read a int from the StreamMessage payload
dbms_output.put_line('read_int:'|| message.read_int(id));

-- Read a long from the StreamMessage payload
dbms_output.put_line('read_long:'|| message.read_long(id));

-- Read a short from the StreamMessage payload
dbms_output.put_line('read_short:'|| message.read_short(id));

-- Read a String into a clob data from the StreamMessage payload
dbms_output.put_line('read_string:');
message.read_string(id, clob_data);
display_clob(clob_data);

-- Assume the users do not know the types of data in the StreamMessage payload.
-- The user can use read_object method to read the data into a sys.aq$_jms_value object
-- These functions are analogy of JMS JAVA api's. See the document for detail.
-- Reset the stream pointer to the begining of the message so that we can read throught
-- the message payload again.
message.reset(id);

LOOP
message.read_object(id, gdata);
IF gdata IS NULL THEN
   EXIT;
END IF;
END LOOP;

CASE gdata.type
    WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTE THEN
dbms_output.put_line('read_object/byte:'|| gdata.num_val);
    WHEN sys.dbms_jms_plsql.DATA_TYPE_SHORT THEN
dbms_output.put_line('read_object/short:'|| gdata.num_val);
    WHEN sys.dbms_jms_plsql.DATA_TYPE_INTEGER THEN
dbms_output.put_line('read_object/int:'|| gdata.num_val);
    WHEN sys.dbms_jms_plsql.DATA_TYPE_LONG THEN
dbms_output.put_line('read_object/long:'|| gdata.num_val);
    WHEN sys.dbms_jms_plsql.DATA_TYPE_FLOAT THEN
dbms_output.put_line('read_object/float:'|| gdata.num_val);
WHEN sys.dbms_jms_plsql.DATA_TYPE_DOUBLE THEN
    dbms_output.put_line('read_object/double:' || gdata.num_val);
WHEN sys.dbms_jms_plsql.DATA_TYPE_BOOLEAN THEN
    dbms_output.put_line('read_object/boolean:' || gdata.num_val);
WHEN sys.dbms_jms_plsql.DATA_TYPE_CHARACTER THEN
    dbms_output.put_line('read_object/char:' || gdata.char_val);
WHEN sys.dbms_jms_plsql.DATA_TYPE_STRING THEN
    dbms_output.put_line('read_object/string:');
display_clob(gdata.text_val);
WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTES THEN
    dbms_output.put_line('read_object/bytes:');
display_blob(gdata.bytes_val);
ELSE dbms_output.put_line('No such data type');
END CASE;
END LOOP;

-- Use either clean_all or clean to clean up the message store when the user
-- do not plan to do payload retrieving on this message anymore
message.clean(id);
-- sys.aq$_jms_stream_message.clean_all();

EXCEPTION
WHEN java_exp THEN
    dbms_output.put_line('exception information:');
display_exp(sys.aq$_jms_stream_message.get_exception());
END;
/

commit;

JMS MapMessage Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS
MapMessage.

Example 16–6 shows how to use JMS type member functions with DBMS_AQ functions
to populate and enqueue a JMS MapMessage represented as sys.aq$_jms_map_
message type in the database. This message later can be dequeued by a JAVA OJMS
client.

Example 16–6  Populating and Enqueuing a JMS MapMessage

set echo off
set verify off
connect sys
DROP USER jmsuser CASCADE;
ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE
CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
```sql
SET ECHO ON
set serveroutput on

DECLARE
    id           pls_integer;
    agent        sys.aq$_agent := sys.aq$_agent(' ', null, 0);
    message      sys.aq$_jms_map_message;
    enqueue_options  dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
   msgid raw(16);
    java_exp     exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
    -- Construct a empty map message object
    message := sys.aq$_jms_map_message.construct;

    -- Shows how to set the JMS header
    message.set_replyto(agent);
    message.set_type('tkaqpet1');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    -- Shows how to set JMS user properties
    message.set_string_property('color', 'RED');
    message.set_int_property('year', 1999);
    message.set_float_property('price', 16999.99);
    message.set_long_property('mileage', 300000);
    message.set_boolean_property('import', True);
    message.set_byte_property('password', -127);

    -- Shows how to populate the message payload of aq$_jms_map_message
    -- Passing -1 reserve a new slot within the message store of sys.aq$_jms_map_message.
    -- The maximum number of sys.aq$_jms_map_message type of messges to be operated at
    -- the same time within a session is 20. Calling clean_body function with
    -- parameter -1
    -- might result a ORA-24199 error if the messages currently operated is already 20.
    -- The user is responsible to call clean or clean_all function to clean up
    -- message store.
    id := message.clear_body(-1);

    -- Write data into the message payload. These functions are analogy of JMS
    -- JAVA api's.
    -- See the document for detail.
    message.set_byte(id, 'BYTE', 10);

    -- Set a byte array entry using RAW data in map message payload
    message.set_bytes(id, 'BYTES', UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));
```
-- Set a byte array entry using only a portion of the RAW data in map message payload
-- Note the offset follows JAVA convention, starting from 0
message.set_bytes(id, 'BYTES_PART', UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')), 0, 16);

-- Set a char entry in map message payload
message.set_char(id, 'CHAR', 'A');

-- Set a double entry in map message payload
message.set_double(id, 'DOUBLE', 9999.99);

-- Set a float entry in map message payload
message.set_float(id, 'FLOAT', 99.99);

-- Set a int entry in map message payload
message.set_int(id, 'INT', 12345);

-- Set a long entry in map message payload
message.set_long(id, 'LONG', 1234567);

-- Set a short entry in map message payload
message.set_short(id, 'SHORT', 123);

-- Set a String entry in map message payload
message.set_string(id, 'STRING', 'Hello World!');

-- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
-- Without doing this, the PL/SQL message is still empty.
message.flush(id);

-- Use either clean_all or clean to clean up the message store when the user
-- do not plan to do payload population on this message anymore
sys.aq$_jms_map_message.clean_all();
--message.clean(id);

-- Enqueue this message into AQ queue using DBMS_AQ package
dbms_aq.enqueue(queue_name => 'jmsuser.jms_map_que',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => message,
    msgid => msgid);

END;
/

commit;

Example 16–7 illustrates how to use JMS type member functions with DBMS_AQ
functions to dequeue and retrieve data from a JMS MapMessage represented as
sys.aq$_jms_map_message type in the database. This message can be enqueued by
a Java OJMS client.

Example 16–7  Dequeuing and Retrieving Data From a JMS MapMessage

set echo off
set verify off
connect sys
DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT ‘Enter the password for JMSUSER: ’ HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password

set echo on
set serveroutput on

DECLARE
    id                 pls_integer;
    blob_data          blob;
    clob_data          clob;
    message            sys.aq$_jms_map_message;
    agent              sys.aq$_agent;
    dequeue_options    dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid              raw(16);
    name_arr           sys.aq$_jms_namearray;
    gdata              sys.aq$_jms_value;

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
    DBMS_OUTPUT.ENABLE (20000);

    -- Dequeue this message from AQ queue using DBMS_AQ package
    dbms_aq.dequeue(queue_name => 'jmsuser.jms_map_que',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

    -- Retrieve the header
    agent := message.get_replyto;
    dbms_output.put_line('Type: ' || message.get_type ||
                          ' UserId: ' || message.get_userid ||
                          ' AppId: ' || message.get_appid ||
                          ' GroupId: ' || message.get_groupid ||
                          ' GroupSeq: ' || message.get_groupseq);

    -- Retrieve the user properties
    dbms_output.put_line('price: ' || message.get_float_property('price'));
    dbms_output.put_line('color: ' || message.get_string_property('color'));
    IF message.get_boolean_property('import') = TRUE THEN
        dbms_output.put_line('import: Yes');
    ELSIF message.get_boolean_property('import') = FALSE THEN
        dbms_output.put_line('import: No');
    END IF;
    dbms_output.put_line('year: ' || message.get_int_property('year'));
    dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));

END;
dbms_output.put_line('password: ' || message.get_byte_property('password'));

-- Shows how to retrieve the message payload of aq$_jms_map_message

-- 'Prepare' sends the content in the PL/SQL aq$_jms_map_message object to
-- Java stored procedure(Jserv) in the form of byte array.
-- Passing -1 reserve a new slot within the message store of
-- sys.aq$_jms_map_message. The maximum number of sys.aq$_jms_map_message
-- type of messages to be operated at the same time within a session is 20.
-- Calling clean_body function with parameter -1
-- might result a ORA-24199 error if the messages currently operated is
-- already 20. The user is responsible to call clean or clean_all function
-- to clean up message store.
id := message.prepare(-1);

-- Assume the users know the names and types in the map message payload.
-- The user can use names to get the corresponding values.
-- These functions are analogous to JMS Java API's. See JMS Types chapter
-- for detail.
dbms_output.put_line('Retrieve payload by Name:');

-- Get a byte entry from the map message payload
dbms_output.put_line('get_byte:' || message.get_byte(id, 'BYTE'));

-- Get a byte array entry from the map message payload
dbms_output.put_line('get_bytes:');
message.get_bytes(id, 'BYTES', blob_data);
display_blob(blob_data);

-- Get another byte array entry from the map message payload
dbms_output.put_line('get_bytes:');
message.get_bytes(id, 'BYTES_PART', blob_data);
display_blob(blob_data);

-- Get a char entry from the map message payload
dbms_output.put_line('get_char:' || message.get_char(id, 'CHAR'));

-- Get a double entry from the map message payload
dbms_output.put_line('get_double:' || message.get_double(id, 'DOUBLE'));

-- Get a float entry from the map message payload
dbms_output.put_line('get_float:' || message.get_float(id, 'FLOAT'));

-- Get a int entry from the map message payload
dbms_output.put_line('get_int:' || message.get_int(id, 'INT'));

-- Get a long entry from the map message payload
dbms_output.put_line('get_long:' || message.get_long(id, 'LONG'));

-- Get a short entry from the map message payload
dbms_output.put_line('get_short:' || message.get_short(id, 'SHORT'));

-- Get a String entry from the map message payload
dbms_output.put_line('get_string:');
message.get_string(id, 'STRING', clob_data);
display_clob(clob_data);

-- Assume users do not know names and types in map message payload.
-- User can first retrieve the name array containing all names in the
payload and iterate through the name list and get the corresponding value. These functions are analogous to JMS Java API's.

See JMS Type chapter for detail.

dbms_output.put_line('Retrieve payload by iteration:');

-- Get the name array from the map message payload
name_arr := message.get_names(id);

-- Iterate through the name array to retrieve the value for each of the name.
FOR i IN name_arr.FIRST..name_arr.LAST LOOP

-- Test if a name exist in the map message payload
-- (It is not necessary in this case, just a demonstration on how to use it)
IF message.item_exists(id, name_arr(i)) THEN
    dbms_output.put_line('item exists:'||name_arr(i));
-- Because we do not know the type of entry, we must use sys.aq$_jms_value type object for the data returned
    message.get_object(id, name_arr(i), gdata);
    IF gdata IS NOT NULL THEN
        CASE gdata.type
            WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTE
                THEN dbms_output.put_line('get_object/byte:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_SHORT
                THEN dbms_output.put_line('get_object/short:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_INTEGER
                THEN dbms_output.put_line('get_object/int:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_LONG
                THEN dbms_output.put_line('get_object/long:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_FLOAT
                THEN dbms_output.put_line('get_object/float:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_DOUBLE
                THEN dbms_output.put_line('get_object/double:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_BOOLEAN
                THEN dbms_output.put_line('get_object/boolean:' || gdata.num_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_CHARACTER
                THEN dbms_output.put_line('get_object/char:' || gdata.char_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_STRING
                THEN dbms_output.put_line('get_object/string:');
                    display_clob(gdata.text_val);
            WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTES
                THEN
                    dbms_output.put_line('get_object/bytes:');
                    display_blob(gdata.bytes_val);
                ELSE dbms_output.put_line('No such data type');
        END CASE;
    ELSE
        dbms_output.put_line('item not exists:'||name_arr(i));
    END IF;
END IF;
END LOOP;

-- Use either clean_all or clean to clean up the message store when the user -- do not plan to do payload population on this message anymore
message.clean(id);
-- sys.aq$_jms_map_message.clean_all();

EXCEPTION
MORE ORACLE® DATABASE JMS EXAMPLES

The sample program in Example 16–8 enqueues a large TextMessage (along with JMS user properties) in an Oracle® Database queue created through the OJMS administrative interfaces to hold JMS TEXT messages. Both the TextMessage and BytesMessage enqueued in this example can be dequeued using OJMS Java clients.

Example 16–8  Enqueuing a Large TextMessage

DECLARE

    text        varchar2(32767);
    agent       sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
    message     sys.aq$_jms_text_message;

    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid               raw(16);

BEGIN

    message := sys.aq$_jms_text_message.construct;
    message.set_replyto(agent);
    message.set_type('tkaqpet2');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);
    message.set_boolean_property('import', True);
    message.set_string_property('color', 'RED');
    message.set_short_property('year', 1999);
    message.set_long_property('mileage', 300000);
    message.set_double_property('price', 16999.99);
    message.set_byte_property('password', 127);

    FOR i IN 1..500 LOOP
        text := CONCAT (text, '1234567890');
    END LOOP;

    message.set_text(text);

    dbms_aq.enqueue(queue_name => 'jmsuser.jms_text_t1',
                    enqueue_options => enqueue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

END;
The sample program in Example 16–9 enqueues a large BytesMessage.

**Example 16–9  Enqueueing a Large BytesMessage**

```sql
DECLARE
  text   VARCHAR2(32767);
  bytes  RAW(32767);
  agent  sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
  message sys.aq$_jms_bytes_message;
  body    BLOB;
  position INT;
  enqueue_options    dbms_aq.enqueue_options_t;
  message_properties dbms_aq.message_properties_t;
 msgid  raw(16);
BEGIN
  message := sys.aq$_jms_bytes_message.construct;
  message.set_replyto(agent);
  message.set_type('tkaqper4');
  message.set_userid('jmsuser');
  message.set_appid('plsql_enq_raw');
  message.set_groupid('st');
  message.set_groupseq(1);
  message.set_boolean_property('import', True);
  message.set_string_property('color', 'RED');
  message.set_short_property('year', 1999);
  message.set_long_property('mileage', 300000);
  message.set_double_property('price', 16999.99);
  -- prepare a huge payload into a blob
  FOR i IN 1..1000 LOOP
    text := CONCAT (text, '0123456789ABCDEF');
  END LOOP;
  bytes := HEXTORAW(text);
  dbms_lob.createtemporary(lob_loc => body, cache => TRUE);
  dbms_lob.open (body, DBMS_LOB.LOB_READWRITE);
  position := 1;
  FOR i IN 1..10 LOOP
    dbms_lob.write ( lob_loc => body,
                    amount => FLOOR((LENGTH(bytes)+1)/2),
                    offset => position,
                    buffer => bytes);
    position := position + FLOOR((LENGTH(bytes)+1)/2) ;
  END LOOP;
  -- end of the preparation
  message.set_bytes(body);
  dbms_aq.enqueue(queue_name => 'jmsuser.jms_bytes_t1',
                   enqueue_options => enqueue_options,
                   message_properties => message_properties,
                   msgid => msgid);
END;
```

16-22 Oracle Streams Advanced Queuing User's Guide
payload => message,
msgid => msgid);

dbms_lob.freetemporary(lob_loc => body);
END;
Introducing Oracle Messaging Gateway

The Messaging Gateway administration package DBMS_MGWADM provides an interface for creating Messaging Gateway agents, managing agents, creating messaging system links, registering non-Oracle queues, and setting up propagation jobs.

This chapter contains these topics:

- Introducing Oracle Messaging Gateway
- Oracle Messaging Gateway Features
- Oracle Messaging Gateway Architecture
- Propagation Processing Overview
- Oracle Streams AQ Buffered Messages and Messaging Gateway

Introducing Oracle Messaging Gateway

Messaging Gateway enables communication between applications based on non-Oracle messaging systems and Oracle® Database.

Oracle® Database provides propagation between two Oracle® Database queues to enable e-business (HTTP through IDAP). Messaging Gateway extends this to applications based on non-Oracle messaging systems.

Because Messaging Gateway is integrated with Oracle® Database and Oracle Database, it offers reliable message delivery. Messaging Gateway guarantees that messages are delivered once and only once between Oracle® Database and non-Oracle messaging systems that support persistence. The PL/SQL interface provides an easy-to-learn administrative API, especially for developers already proficient in using Oracle® Database.

This release of Messaging Gateway supports the integration of Oracle® Database with applications based on WebSphere MQ 6.0 and TIB/Rendezvous 7.2.

Oracle Messaging Gateway Features

Messaging Gateway provides the following features:

- Extends Oracle® Database message propagation

  Messaging Gateway propagates messages between Oracle® Database and non-Oracle messaging systems. Messages sent by Oracle® Database applications can be received by non-Oracle messaging system applications. Conversely,
messages published by non-Oracle messaging system applications can be consumed by Oracle® Database applications.

See Also:
- "Propagation Processing Overview" on page 17-4
- Chapter 20, "Oracle Messaging Gateway Message Conversion"

- Support for Java Message Service (JMS) messaging systems
  Messaging Gateway propagates messages between Oracle Java Message Service (Oracle JMS) and WebSphere MQ Java Message Service (WebSphere MQ JMS).

- Native message format support
  Messaging Gateway supports the native message formats of messaging systems. Oracle® Database messages can have RAW or any Oracle object type payload. WebSphere MQ messages can be text or byte messages. TIB/Rendezvous messages can be any TIB/Rendezvous wire format datatype except the nested datatype MSG and those with unsigned integers.

- Message conversion
  Messaging Gateway facilitates message conversion between Oracle® Database messages and non-Oracle messaging system messages. Messages are converted through either automatic routines provided by Messaging Gateway or customized message transformation functions that you provide.

  Note: Messaging Gateway does not support message propagation between JMS and non-JMS messaging systems.

  See Also: "Converting Oracle Messaging Gateway Non-JMS Messages" on page 20-1

- Integration with Oracle Database
  Messaging Gateway is managed through a PL/SQL interface similar to that of Oracle® Database. Configuration information is stored in Oracle Database tables. Message propagation is carried out by an external process of the Oracle Database server.

- Guaranteed message delivery
  If the messaging systems at the propagation source and propagation destination both support transactions, then Messaging Gateway guarantees that persistent messages are propagated exactly once. If messages are not persistent or transactions are not supported by the messaging systems at the propagation source or propagation destination, then at-most-once propagation is guaranteed.

- Security support
  Messaging Gateway supports client authentication of Oracle Database and non-Oracle messaging systems.
  Messaging Gateway also allows Secure Socket Layer (SSL) support for IBM WebSphere MQ and WebSphere MQ JMS connections made by the Messaging Gateway agent.

- Multiple agent support
Messaging Gateway supports multiple agents for a given database. Users can partition propagation jobs based on functionality, organizations, or workload and assign them to different Messaging Gateway agents. This allows Messaging Gateway to scale in a RAC environment and enables propagation job grouping and isolation.

**Oracle Messaging Gateway Architecture**

Messaging Gateway has two main components:

- Administration Package DBMS_MGWADM
- Messaging Gateway Agent

Figure 17–1 shows how these components work together with Oracle Database and non-Oracle messaging systems.

![Figure 17–1 Messaging Gateway Architecture](image)

**Administration Package DBMS_MGWADM**

The Messaging Gateway administration package DBMS_MGWADM provides an interface for creating named Messaging Gateway agents, managing agents, creating messaging system links, registering non-Oracle queues, and setting up propagation jobs.

Users call the procedures in the package to make configuration changes regardless of whether the Messaging Gateway agent is running. If the Messaging Gateway agent is running, then the procedures in the package send notifications for configuration changes to the agent. The agent dynamically alters its configuration for most configuration changes, although some changes require that the agent be shut down and restarted before they take effect. All the procedures in the package are serialized to guarantee that the Messaging Gateway agent receives and processes notifications in the same order as they are made.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information on DBMS_MGWADM
Oracle Messaging Gateway Agent

The Messaging Gateway agent runs as an external process of the Oracle Database server and processes propagation jobs. It is started and shut down by calling the STARTUP and SHUTDOWN procedures in DBMS_MGWADM package.

The Messaging Gateway agent contains a multithreaded propagation engine and a set of drivers for messaging systems. The propagation engine fairly schedules propagation jobs and processes propagation jobs concurrently. The polling thread in the agent periodically polls the source queues of enabled propagation jobs and wakes up worker threads to process propagation jobs if messages are available. The drivers for non-Oracle messaging systems run as clients of the messaging systems for all messaging operations.

Oracle Database

As an Oracle Database feature, Messaging Gateway provides a mechanism of message propagation between Oracle® Database and non-Oracle messaging systems. Oracle® Database is involved in every propagation job as either propagation source or propagation destination.

Messaging Gateway is managed through the PL/SQL administration package DBMS_MGWADM. All configuration information and execution state information of Messaging Gateway are stored in Oracle Database and can be accessed through database views.

The Messaging Gateway agent runs as an external procedure of the Oracle Database server. Therefore, it runs only when its associated database server is running.

Non-Oracle Messaging Systems

The Messaging Gateway agent connects to non-Oracle messaging systems through messaging system links. Messaging system links are communication channels between the Messaging Gateway agent and non-Oracle messaging systems. Users can use the administration package DBMS_MGWADM to configure multiple links to the same or different non-Oracle messaging systems.

Queues in non-Oracle messaging systems, such as WebSphere MQ queues, TIB/Rendezvous subjects, and WebSphere MQ JMS destinations (queues and topics) can all serve as propagation sources and destinations for Messaging Gateway. They are referred to as foreign queues. All foreign queues involved in message propagation as source queues, destination queues, or exception queues must be registered through the administration package. The registration of a foreign queue does not create the physical queue in a non-Oracle messaging system, but merely records information about the queue, such as the messaging system link to access it, its native name, and its domain (queue or topic). The physical queue must be created through the administration interface of the non-Oracle messaging system.

See Also: "Registering a Non-Oracle Queue" on page 19-13

Propagation Processing Overview

Propagation jobs must be defined in order for messages to be propagated from one messaging system to another. A propagation job defines the source queue, destination queue, and various other attributes that affect the processing of the propagation job.

If the propagation source is a queue (point-to-point), then the Messaging Gateway agent moves all messages in the queue to the destination. If the propagation source is a topic (publish/subscribe), then the Messaging Gateway agent creates a subscription
on the propagation source topic. The agent moves all messages that are published to
the topic after the subscription is created.

A propagation job is processed when it is enabled. Disabling a propagation job stops
propagation processing but does not stop message subscription.

When the Messaging Gateway agent processes a propagation job, it dequeues
messages from the source queue and enqueues the messages to the destination queue.
As each message is propagated, it is converted from its native format in the source
messaging system to its native format in the destination messaging system. Messaging
Gateway provides automatic message conversions between simple and commonly
used message formats. You can customize message conversions by providing your
own message transformation functions.

When the Messaging Gateway agent fails to convert a message from the source format
to the destination format, the agent moves the message from the source queue to an
exception queue, if the exception queue exists, and continues to process the
propagation job.

If the Messaging Gateway agent runs into failures when processing a propagation job,
it retries up to sixteen times in an exponential backoff scheme (from two seconds up to
thirty minutes) before it stops retrying.

To guarantee reliable message delivery, Messaging Gateway requires logging queues
in messaging systems that support transactions and persistent messages. The
Messaging Gateway agent uses the logging queues to store the processing states of
propagation jobs so that it can restore propagation processing from failures.

See Also: "Configuring Oracle Messaging Gateway Propagation
Jobs" on page 19-14

Oracle Streams AQ Buffered Messages and Messaging Gateway

Messaging Gateway does not support propagation of buffered messages. In outbound
propagation, the Messaging Gateway agent dequeues only persistent messages from
AQ queues. In inbound propagation, the Messaging Gateway agent always enqueues
persistent messages into AQ queues.
Chapter 18: Getting Started with Oracle Messaging Gateway

This chapter describes Oracle Messaging Gateway (MGW) prerequisites and how to load, set up, and unload Messaging Gateway. It also describes how to set up and modify the mgw.ora initialization file.

This chapter contains these topics:

- Oracle Messaging Gateway Prerequisites
- Loading and Setting Up Oracle Messaging Gateway
- Setting Up Non-Oracle Messaging Systems
- Verifying the Oracle Messaging Gateway Setup
- Unloading Oracle Messaging Gateway
- Understanding the mgw.ora Initialization File

Oracle Messaging Gateway Prerequisites

Messaging Gateway uses one Oracle Scheduler job for each Messaging Gateway agent. If the value of the JOB_QUEUE_PROCESSES database initialization parameter is zero, then that parameter does not influence the number of Oracle Scheduler jobs that can run concurrently. However, if the value is non-zero, it effectively becomes the maximum number of Scheduler jobs and job queue jobs that can run concurrently. If a non-zero value is set, it should be large enough to accommodate a Scheduler job for each Messaging Gateway agent to be started.

Loading and Setting Up Oracle Messaging Gateway

Perform the following procedures before running Messaging Gateway:

- Loading Database Objects into the Database
- Modifying listener.ora for the External Procedure
- Modifying tnsnames.ora for the External Procedure
- Setting Up a mgw.ora Initialization File
- Creating an Oracle Messaging Gateway Administration User
- Creating an Oracle Messaging Gateway Agent User
- Configuring Oracle Messaging Gateway Connection Information
- Configuring Oracle Messaging Gateway in a RAC Environment
Loading Database Objects into the Database

Using SQL*Plus, run `ORACLE_HOME/mgw/admin/catmgw.sql` as user SYS as SYSDBA. This script loads the database objects necessary for Messaging Gateway, including roles, tables, views, object types, and PL/SQL packages. It creates public synonyms for Messaging Gateway PL/SQL packages. It creates two roles, `MGW_ADMINISTRATOR_ROLE` and `MGW_AGENT_ROLE`, with certain privileges granted. All objects are owned by SYS.

Modifying listener.ora for the External Procedure

This procedure is for Linux 32-bit operating systems only. Static service information for the listener is not necessary on the Windows operating system.

You must modify `listener.ora` so that the Messaging Gateway PL/SQL packages can call the external procedure.

1. Verify that the default **Inter-process Communication** (IPC) protocol address for the external procedures is set.

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

2. Add static service information for the listener in step 1. This involves setting a `SID_DESC` for the listener. Within the `SID_DESC`, the parameters described in Table 18-1 are important to Messaging Gateway and must be specified according to your own situation.

   **Table 18-1  SID_DESC Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID_NAME</td>
<td>The SID that is specified in the net service name in tnsnames.ora. In the following example, the SID_NAME is mgwextproc.</td>
</tr>
<tr>
<td>ENVS</td>
<td>Set up the LD_LIBRARY_PATH environment needed for the external procedure to run. The LD_LIBRARY_PATH must contain the following paths: JRE_HOME/lib/PLATFORM_TYPE JRE_HOME/lib/PLATFORM_TYPE/server ORACLE_HOME/lib It should also contain any additional libraries required by third-party messaging systems. See “Setting Up Non-Oracle Messaging Systems” on page 18-6.</td>
</tr>
<tr>
<td>ORACLE_HOME</td>
<td>Your Oracle home directory. Using $ORACLE_HOME does not work.</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>The name of the external procedure agent, which is extproc</td>
</tr>
</tbody>
</table>
Example 18–1 adds SID_NAME mgwextproc to a listener.ora file for Linux x86.

Example 18–1  Adding Static Service Information for a Listener

```bash
# Add a SID_DESC
SID_LIST_LISTENER= (SID_LIST=
    (SID_DESC= (SID_NAME= mgwextproc)
       (ENVS= "LD_LIBRARY_PATH=JRE_HOME/lib/i386:JRE_HOME/lib/i386/server:ORACLE_HOME/lib")
       (ORACLE_HOME=ORACLE_HOME)
       (PROGRAM = extproc))
```

Modifying tnsnames.ora for the External Procedure

This procedure is for Linux 32-bit operating systems only. For the external procedure, configure a net service name MGW_AGENT in tnsnames.ora whose connect descriptor matches the information configured in listener.ora, as shown in Example 18–2. The net service name must be MGW_AGENT (this value is fixed). The KEY value must match the KEY value specified for the IPC protocol in listener.ora. The SID value must match the value specified for SID_NAME of the SID_DESC entry in listener.ora.

Example 18–2  Configuring MGW_AGENT

```bash
MGW_AGENT =
    (DESCRIPTION= 
       (ADDRESS_LIST= (ADDRESS= (PROTOCOL=IPC)(KEY=EXTPROC)))
       (CONNECT_DATA= (SID=mgwextproc)))
```

Note: If the names.default_domain parameter for sqlnet.ora has been used to set a default domain, then that domain must be appended to the MGW_AGENT net service name in tnsnames.ora. For example, if sqlnet.ora contains the entry names.default_domain=acme.com, then the net service name in tnsnames.ora must be MGW_AGENT.acme.com.

Setting Up a mgw.ora Initialization File

The Messaging Gateway default initialization file ORACLE_HOME/mgw/admin/mgw.ora is a text file. The Messaging Gateway external procedure uses it to get initialization parameters to start the Messaging Gateway agent. Copy ORACLE_HOME/mgw/admin/sample_mgw.ora to mgw.ora and modify it according to your situation.

The following procedure sets environment variables and other parameters required for all applications of Messaging Gateway:

1. **Windows Operating System Only**: Set the MGW_PRE_PATH variable. Its value is the path to the jvm.dll library:
Loading and Setting Up Oracle Messaging Gateway

set MGW_PRE_PATH = JRE_HOME\bin\client

This variable is prepended to the path inherited by the Messaging Gateway agent process.

2. Set CLASSPATH to include at least the following:
   - JRE runtime classes:
     JRE_HOME/lib/rt.jar
   - Oracle JDBC classes:
     ORACLE_HOME/jdbc/lib/ojdbc5.jar
   - Oracle internationalization classes:
     ORACLE_HOME/jlib/orai18n.jar
   - SQLJ runtime:
     ORACLE_HOME/sqlj/lib/runtime12.jar
   - Java Message Service (JMS) interface
     ORACLE_HOME/rdbms/jlib/jmscommon.jar
   - Oracle JMS implementation classes
     ORACLE_HOME/rdbms/jlib/aqapi.jar
   - Java transaction API
     ORACLE_HOME/jlib/jta.jar
   - Any additional classes needed for Messaging Gateway to access non-Oracle messaging systems

See Also: "Setting Up Non-Oracle Messaging Systems" on page 18-6

Note: Replace ORACLE_HOME with the appropriate, spelled-out value. Using $ORACLE_HOME, for example, does not work.

Users of the Windows operating system must set CLASSPATH using the Windows operating system path syntax.

Creating an Oracle Messaging Gateway Administration User

To perform Messaging Gateway administration work, a database user must be created with MGW_ADMINISTRATOR_ROLE privileges, as shown in Example 18–3.

Example 18–3  Creating a Messaging Gateway Administrator User

CREATE USER admin_user IDENTIFIED BY admin_password;
GRANT CREATE SESSION to admin_user;
GRANT MGW_ADMINISTRATOR_ROLE to admin_user;

Creating an Oracle Messaging Gateway Agent User

To establish the Messaging Gateway agent connection back to the database, a database user with MGW_AGENT_ROLE privileges must be created, as shown in Example 18–4.
Getting Started with Oracle Messaging Gateway

Example 18–4  Creating a Messaging Gateway Agent User

CREATE USER agent_user IDENTIFIED BY agent_password;
GRANT CREATE SESSION to agent_user;
GRANT MGW_AGENT_ROLE to agent_user;

Configuring Oracle Messaging Gateway Connection Information

After the Messaging Gateway agent user is created, the administration user uses
DBMS_MGWADM.ALTER_AGENT to configure Messaging Gateway with the username,
password, and database connect string used by the Messaging Gateway agent to
connect back to the database, as shown in Example 18–5. Use the Messaging Gateway
username and password that you created in "Creating an Oracle Messaging Gateway
Agent User" on page 18-4. The database connect string parameter can be set to either a
net service name in tnsnames.ora (with IPC protocol for better performance) or
NULL. If NULL, then the oracle_sid parameter must be set in mgw.ora.

For this release, always specify a not NULL value for the database connect string
parameter when calling DBMS_MGWADM.ALTER_AGENT.

Example 18–5  Configuring Messaging Gateway Connection Information

set echo off
set verify off
connect admin_user

ACCEPT password CHAR PROMPT 'Enter the password for AGENT_USER: ' HIDE

EXEC DBMS_MGWADM.ALTER_AGENT(
    agent_name  => 'default_agent',
    username    => 'agent_user',
    password    => '&password',
    database    => 'agent_database');

Configuring Oracle Messaging Gateway in a RAC Environment

This section contains these topics:

■ Configuring Connection Information for the MGW Agent Connections
■ Setting the RAC Instance for the Messaging Gateway Agent

Configuring Connection Information for the MGW Agent Connections

You must make all database connections made by the Messaging Gateway agent to the
instance on which the Messaging Gateway agent process is running. This ensures
correct failover behavior in a Real Application Clusters (RAC) environment. You can
configure connections this way by having the instances use slightly different
tnsnames.ora files. Each file contains an entry with the same net service name, but
the connect data refers to only the instance associated with that tnsnames.ora file.
The common net service name would then be used for the database parameter when
DBMS_MGWADM.ALTER_AGENT is used to configure the Messaging Gateway agent
database connection information.

For example, in a two-instance RAC environment with instances OraDB1 and OraDB2,
where the net service name AGENT_DB is to be used, the tnsnames.ora for instance
OraDB1 would look like this:

AGENT_DB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = myhost1.mycorp.com)(PORT = 1521))
The `tnsnames.ora` for OraDB2 would look like this:

```sql
AGENT_DB =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = myhost2.mycorp.com)(PORT = 1521))
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = OraDB10.mycorp.com)
            (INSTANCE_NAME = OraDB2)
        )
    )
```

You would then configure Messaging Gateway agent user connection information by running the following command:

```sql
EXEC DBMS_MGWADM.ALTER_AGENT(
    agent_name   => 'default_agent',
    username     => 'agent_user',
    password     => 'agent_password',
    database     => 'agent_db');
```

### Setting the RAC Instance for the Messaging Gateway Agent

Messaging Gateway provides service affinity for the Messaging Gateway agent external process by leveraging the database service support of the Oracle Scheduler. By default, a Messaging Gateway agent will use the default database service that is mapped to all instances. If you want a Messaging Gateway agent to start on a select group of database instances, you must create a database service for those instances and then assign the database service to the Messaging Gateway agent using the `SERVICE` parameter of the `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT` procedures. The `DBMS_MGWADM.STARTUP` procedure submits an Oracle Scheduler job that starts the Messaging Gateway agent external process when the Scheduler job is executed. The Scheduler job will use the database service configured for the Messaging Gateway agent.

The database service specified by the `SERVICE` parameter is only used for the service affinity of the Oracle Scheduler job and thus the service affinity for the Messaging Gateway external process. It is not used for the database connections made by the Messaging Gateway agent user. Those JDBC client connections are based on the values specified for the `DATABASE` and `CONNTYPE` parameters.

**See Also:** "Running the Oracle Messaging Gateway Agent on RAC" on page 19-5

### Setting Up Non-Oracle Messaging Systems

This section contains these topics:

- Setting Up for TIB/Rendezvous
- Setting Up for WebSphere MQ Base Java or JMS
Setting Up for TIB/Rendezvous

Running as a TIB/Rendezvous Java client application, the Messaging Gateway agent requires TIB/Rendezvous software to be installed on the computer where the Messaging Gateway agent runs. In this section TIBRV_HOME refers to the installed TIB/Rendezvous software location.

Modifying listener.ora

On the Linux operating system, LD_LIBRARY_PATH in the entry for Messaging Gateway must include TIBRV_HOME/lib for the agent to access TIB/Rendezvous shared library files.

See Also: "Modifying listener.ora for the External Procedure" on page 18-2

On the Windows operating system, you are not required to modify listener.ora. But the system environment variable PATH must include TIBRV_HOME\bin.

Modifying mgw.ora

MGW_PRE_PATH must include the directory that contains the TIB/Rendezvous license ticket file (tibrv.tkt), which usually is located in TIBRV_HOME/bin.

CLASSPATH must include the TIB/Rendezvous jar file TIBRV_HOME/lib/tibrvj.jar. If you use your own customized TIB/Rendezvous advisory message callback, then the location of the callback class must also be included.

You can set the following Java properties to change the default setting:

- oracle.mgw.tibrv.encoding
- oracle.mgw.tibrv.intraProcAdvSubjects
- oracle.mgw.tibrv.advMsgCallback

See Also: "Understanding the mgw.ora Initialization File" on page 18-9

Example 18–6 Setting Java Properties

setJavaProp oracle.mgw.tibrv.encoding=ISO8859_1
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=_RV.>
setJavaProp oracle.mgw.tibrv.advMsgCallback=MyadvCallback

Setting Up for WebSphere MQ Base Java or JMS

The WebSphere MQ client and WebSphere MQ classes for Java and JMS must be installed on the computer where the Messaging Gateway agent runs. In this section MQ_HOME refers to the location of the installed client. On the Linux operating system, this location is always /opt/mqm. On the Windows operating system, the installed location can vary.

Modifying listener.ora

No extra modification of listener.ora is necessary for Messaging Gateway to access WebSphere MQ.
Modifying mgw.ora

When using WebSphere MQ Base Java (non-JMS) interface, set CLASSPATH to include at least the following (in addition to those in "Setting Up a mgw.ora Initialization File" on page 18-3):

- MQ_HOME/java/lib/com.ibm.mq.jar
- MQ_HOME/java/lib/connector.jar

When using WebSphere MQ JMS interface, set CLASSPATH to include at least the following (in addition to those in "Setting Up a mgw.ora Initialization File" on page 18-3):

- MQ_HOME/java/lib/com.ibm.mqjms.jar
- MQ_HOME/java/lib/com.ibm.mq.jar
- MQ_HOME/java/lib/connector.jar

Verifying the Oracle Messaging Gateway Setup

The following procedure verifies the setup and includes a simple startup and shutdown of the Messaging Gateway agent:

1. Start the database listeners.
   Start the listener for the external procedure and other listeners for the regular database connection.

2. Test the database connect string for the Messaging Gateway agent user.
   Run sqlplus agent_user/agent_password@agent_database.
   If it is successful, then the Messaging Gateway agent is able to connect to the database.

3. Linux Operating System Only: Test the net service entry used to call the external procedure.
   Run sqlplus agent_user/agent_password@MGW_AGENT.
   This should fail with "ORA-28547: connection to server failed, probable Oracle Net admin error". Any other error indicates that the tnsnames.ora, listener.ora, or both are not correct.

4. Connect as admin_user and call DBMS_MGWADM.STARTUP to start the Messaging Gateway agent.

5. Using the MGW_GATEWAY view, wait for AGENT_STATUS to change to RUNNING and AGENT_PING to change to REACHABLE.

6. Connect as admin_user and call DBMS_MGWADM.SHUTDOWN to shut down the Messaging Gateway agent.

7. Using the MGW_GATEWAY view, wait for AGENT_STATUS to change to NOT_STARTED.

Unloading Oracle Messaging Gateway

Use this procedure to unload Messaging Gateway:

1. Shut down Messaging Gateway.
2. Remove any user-created queues whose payload is a Messaging Gateway canonical type (for example, SYS.MGW_BASIC_MSG_T).

3. Using SQL*Plus, run ORACLE_HOME/mgw/admin/catnomgw.sql as user SYS as SYSDBA.

   This drops the database objects used by Messaging Gateway, including roles, tables, views, packages, object types, and synonyms.

4. Remove entries for Messaging Gateway created in listener.ora and tnsnames.ora.

Understanding the mgw.ora Initialization File

Messaging Gateway reads initialization information from a text file when the Messaging Gateway agent starts. The initialization file contains lines for setting initialization parameters, environment variables, and Java properties. Each entity must be specified on one line. Leading whitespace is trimmed in all cases.

A Messaging Gateway administrator can specify the initialization file to be used for a Messaging Gateway agent via DBMS_MGWADM.CREATE_AGENT and DBMS_MGWADM.ALTER_AGENT. If an initialization file is not specified then a default initialization file will be used.

The default initialization file for the default agent is ORACLE_HOME/mgw/admin/mgw.ora.

The default initialization file for a named agent is ORACLE_HOME/mgw/admin/mgw_AGENTNAME.ora where AGENTNAME is the name in uppercase of the Messaging Gateway agent. For example, if the agent name is my_agent then the name of the agent's default initialization file is ORACLE_HOME/mgw/admin/mgw_MY_AGENT.ora. If the default initialization file for a named agent is not found then ORACLE_HOME/mgw/admin/mgw.ora will be used.

mgw.ora Initialization Parameters

The initialization parameters are typically specified by lines having a "name=value<NL>" format where name represents the parameter name, value represents its value and <NL> represents a new line.

log_directory

   Usage: Specifies the directory where the Messaging Gateway log/trace file is created.

   Format: log_directory = value

   Default: ORACLE_HOME/mgw/log

   Example: log_directory = /private/mgwlog

log_level

   Usage: Specifies the level of logging detail recorded by the Messaging Gateway agent. The logging level can be dynamically changed by changing by calling DBMS_MGWADM.SET_LOG_LEVEL while the Messaging Gateway agent is running. Oracle recommends that log level 0 (the default value) be used at all times.
Format:  log_level = value

Values:
0 for basic logging; equivalent to DBMS_MGWADM.BASIC_LOGGING
1 for light tracing; equivalent to DBMS_MGWADM.TRACE_LITE_LOGGING
2 for high tracing; equivalent to DBMS_MGWADM.TRACE_HIGH_LOGGING
3 for debug tracing; equivalent to DBMS_MGWADM.TRACE_DEBUG_LOGGING

Example:  log_level = 0

mgw.ora Environment Variables

Because the Messaging Gateway process environment is not under the direct control of the user, certain environment variables should be set using the initialization file. The environment variables currently used by the Messaging Gateway agent are CLASSPATH, MGW_PRE_PATH, and ORACLE_SID.

Environment variables such as CLASSPATH and MGW_PRE_PATH are set so the Messaging Gateway agent can find the required shared objects, Java classes, and so on. Environment variables are specified by lines having a "set env_var=value<NL>" or "setenv env_var=value<NL>" format where env_var represents the name of the environment variable to set, value represents the value of the environment variable, and <NL> represents a new line.

CLASSPATH

Usage:  Used by the Java Virtual Machine to find Java classes needed by the Messaging Gateway agent for propagation between Oracle® Database and non-Oracle messaging systems.

Format:  set CLASSPATH=value


MGW_PRE_PATH

Usage:  Appended to the front of the path inherited by the Messaging Gateway process. For the Windows operating system, this variable must be set to indicate where the library jvm.dll is found.

Format:  set MGW_PRE_PATH=value

Example:  set MGW_PRE_PATH=JRE_HOME\bin\client

ORACLE_SID

Usage:  Can be used when a service name is not specified when configuring Messaging Gateway.
Format:  set ORACLE_SID=value

Example:  set ORACLE_SID=my_sid

**mgw.ora Java Properties**

You must specify Java system properties for the Messaging Gateway JVM when working with TIB/Rendezvous subjects. You can use the `setJavaProp` parameter of the Messaging Gateway initialization file for this. Java properties are specified by lines having a "`setJavaProp prop_name=value<NL>`" format, where `prop_name` represents the name of the Java property to set, `value` represents the value of the Java property, and `<NL>` represents a new line character.

**oracle.mgw.batch_size**

**Usage:** This Java property represents the maximum number of messages propagated in one transaction. It serves as a default value if the Messaging Gateway job option, `MsgBatchSize`, is not specified. If altered from the default, then consideration should be given to the expected **message** size and the Messaging Gateway agent memory (see `max_memory` parameter of `DBMS_MGWADM.ALTER_AGENT`). The minimum value of this Java property is 1, the maximum is 100, and the default is 30.

**Syntax:**  setJavaProp oracle.mgw.batch_size=value

**Example:**  setJavaProp oracle.mgw.batch_size=10

**oracle.mgw.polling_interval**

**Usage:** This parameter specifies the time (in milliseconds) that must elapse between polls for available messages of a propagation source **queue**. The default polling interval used by Messaging Gateway is 5000 milliseconds (5 seconds).

**Syntax:**  setJavaProp oracle.mgw.polling_interval=value

**Example:**  setJavaProp oracle.mgw.polling_interval=1000

**oracle.mgw.tibrv.encoding**

**Usage:** This parameter specifies the character encoding to be used by the TIB/Rendezvous messaging system links. Only one character set for all configured TIB/Rendezvous links is allowed due to TIB/Rendezvous restrictions. The default is ISO 8859-1 or the character set specified by the Java system property `file.encoding`.

**Syntax:**  setJavaProp oracle.mgw.tibrv.encoding=value

**Example:**  setJavaProp oracle.mgw.tibrv.encoding=ISO8859_1

**See Also:** "DBMS_MGWADM" in Oracle Database PL/SQL Packages and Types Reference
oracle.mgw.tibrv.intraProcAdvSubjects

**Usage**  
Used for all TIB/Rendezvous messaging system links, this parameter specifies the names of system advisory subjects that present on the intraprocess transport.

**Syntax**  
```
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=
advisorySubjectName[:advisorySubjectName]
```

**Example:**  
```
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=_RV.>
```

oracle.mgw.tibrv.advMsgCallback

**Usage:**  
Used for all TIB/Rendezvous messaging system links, this parameter specifies the name of the Java class that implements the `TibrvMsgCallback` interface to handle system advisory messages. If it is not specified, then the default system advisory message handler provided by Messaging Gateway is used, which writes system advisory messages into Messaging Gateway log files. If it is specified, then the directory where the class file is stored must be included in the `CLASSPATH` in `mgw.ora`.

**Syntax:**  
```
setJavaProp oracle.mgw.tibrv.advMsgCallback=className
```

**Example:**  
```
setJavaProp oracle.mgw.tibrv.advMsgCallback=MyAdvCallback
```

oracle.net.tns_admin

**Usage:**  
This parameter specifies the directory of the `tnsnames.ora` file. It must be set if the Messaging Gateway agent is configured to use the JDBC Thin driver and the database specifier of the agent connection information is a TNSNames alias. This does not need to be set if the JDBC OCI driver is used or the database specifier is something other than a TNSNames alias.

**Syntax:**  
```
setJavaProp oracle.net.tns_admin=value
```

**Example:**  
```
setJavaProp oracle.net.tns_admin=/myoraclehome/network/admin
```

**mgw.ora Comment Lines**

Comment lines are designated with a `#` character as the first character of the line.
After Oracle Messaging Gateway (MGW) is loaded and set up, it is ready to be configured and run. You can use DBMS_MGWADM.ALTER_AGENT to set the username, password, database specifier, and connection type the Messaging Gateway agent will use for creating database connections.

This chapter contains these topics:

- Configuring the Oracle Messaging Gateway Agent
- Starting and Shutting Down the Oracle Messaging Gateway Agent
- Configuring Messaging System Links
- Configuring Non-Oracle Messaging System Queues
- Configuring Oracle Messaging Gateway Propagation Jobs
- Propagation Jobs, Subscribers, and Schedules
- Configuration Properties

---

**Note:** All commands in the examples must be run as a user granted MGW_ADMINISTRATOR_ROLE.

**See Also:** "DBMS_MGWADM" and "DBMS_MGWMGS" in Oracle Database PL/SQL Packages and Types Reference

## Configuring the Oracle Messaging Gateway Agent

Messages are propagated between Oracle® Database and non-Oracle messaging systems by the Messaging Gateway agent. The Messaging Gateway agent runs as an external process of the Oracle Database server.

Messaging Gateway supports multiple agents for a given database. A default agent is automatically created that has the name of DEFAULT_AGENT. Additional named agents can be created to provide propagation job isolation and grouping, and scaling in a RAC environment. The default agent is usually sufficient for single instance, non-RAC, environments.

This section contains these topics:

- Creating a Messaging Gateway Agent
- Removing a Messaging Gateway Agent
- Database Connection
Configuring the Oracle Messaging Gateway Agent

- Resource Limits

Creating a Messaging Gateway Agent

You can use DBMS_MGWADM.CREATE_AGENT to create additional Messaging Gateway agents. The Messaging Gateway default agent, DEFAULT_AGENT, is automatically created when Messaging Gateway is installed and will always exist.

Agents can be configured with an agent user, connection information, database service, and resource limits when the agent is created, or at a later time using DBMS_MGWADM.ALTER_AGENT. A Messaging Gateway agent must be configured with a database user that has been granted the role MGW_AGENT_ROLE before the agent can be started.

Example 19–1 creates the agent named myagent and specifies the database connection information for the agent user. Default values are used for all other parameters.

**Example 19–1  Creating a Messaging Gateway Agent**

```sql
SQL> exec DBMS_MGWADM.CREATE_AGENT(
agent_name  => 'myagent',
username    => 'mgwagent',
password    => 'mgwagent_password',
database    => 'mydatabase');
```

Removing a Messaging Gateway Agent

A Messaging Gateway agent can be removed by calling DBMS_MGWADM.REMOVE_AGENT. Before an agent can be removed, all Messaging Gateway links associated with the agent must be removed and the agent shut down. The default agent, DEFAULT_AGENT, cannot be removed. Example 19–2 removes the agent named myagent.

**Example 19–2  Removing a Messaging Gateway Agent**

```sql
SQL> exec DBMS_MGWADM.REMOVE_AGENT(agent_name   => 'myagent');
```

Database Connection

The Messaging Gateway agent runs as a process external to the database. To access Oracle® Database and the Messaging Gateway packages, the Messaging Gateway agent needs to establish connections to the database. You can use DBMS_MGWADM.ALTER_AGENT to set the username, password and the database connect string that the Messaging Gateway agent will use for creating database connections. The user must be granted the role MGW_AGENT_ROLE before the Messaging Gateway agent can be started.

Example 19–3 shows the Messaging Gateway default agent being configured for user mgwagent with password mgwagent_password using net service name mydatabase.

**Example 19–3  Setting Database Connection Information**

```sql
SQL> exec DBMS_MGWADM.ALTER_AGENT (
agent_name => 'default_agent',
username   => 'mgwagent',
password   => 'mgwagent_password',
database   => 'mydatabase');
```
Resource Limits

You can use `DBMS_MGWADM.ALTER_AGENT` to set resource limits for the Messaging Gateway agent. For example, you can set the heap size of the Messaging Gateway agent process and the number of propagation threads used by the agent process. The default values are 64 MB of memory heap and one propagation thread. For named agents, these values can also be specified when the agent is created by `DBMS_MGWADM.CREATE_AGENT`.

Example 19–4 sets the heap size to 96 MB and two propagation threads for the agent myagent.

Example 19–4 Setting the Resource Limits

```sql
SQL> exec DBMS_MGWADM.ALTER_AGENT(
    2    agent_name  => 'myagent',
    3    max_memory  => 96,
    4    max_threads => 2);
```

The memory heap size and the number of propagation threads cannot be altered when the Messaging Gateway agent is running.

Starting and Shutting Down the Oracle Messaging Gateway Agent

This section contains these topics:

- Starting the Oracle Messaging Gateway Agent
- Shutting Down the Oracle Messaging Gateway Agent
- Oracle Messaging Gateway Agent Scheduler Job
- Running the Oracle Messaging Gateway Agent on RAC

Starting the Oracle Messaging Gateway Agent

After the Messaging Gateway agent is configured, you can start the agent with `DBMS_MGWADM.STARTUP`. Example 19–5 shows how to start the default agent and agent myagent.

Example 19–5 Starting the Messaging Gateway Agent

```sql
SQL> exec DBMS_MGWADM.STARTUP;
SQL> exec DBMS_MGWADM.STARTUP ('myagent');
```

You can use the `MGW_GATEWAY` view to check the status of the Messaging Gateway agent, as described in Chapter 21, "Monitoring Oracle Messaging Gateway".

Shutting Down the Oracle Messaging Gateway Agent

You can use `DBMS_MGWADM.SHUTDOWN` to shut down the Messaging Gateway agent. Example 19–6 shows how to shut down the Messaging Gateway default agent and agent myagent.

Example 19–6 Shutting Down the Messaging Gateway Agent

```sql
SQL> exec DBMS_MGWADM.SHUTDOWN;
SQL> exec DBMS_MGWADM.SHUTDOWN ('myagent');
```
You can use the `MGW_GATEWAY` view to check if the Messaging Gateway agent has shut down successfully, as described in Chapter 21, "Monitoring Oracle Messaging Gateway".

**Oracle Messaging Gateway Agent Scheduler Job**

Messaging Gateway uses a Scheduler job to start the Messaging Gateway agent. This job is created when procedure `DBMS_MGWADM.STARTUP` is called. When the job is run, it calls an external procedure that creates the Messaging Gateway agent in an external process. The job is removed after:

- The agent shuts down because `DBMS_MGWADM.SHUTDOWN` was called
- The agent terminates because a non-restartable error occurs

Messaging Gateway uses `DBMS_SCHEDULER` to create a repeatable Scheduler job with a repeat interval of one minute. The job is owned by `SYS`. A repeatable job enables the Messaging Gateway agent to restart automatically when a given job instance ends because of a database shutdown, database malfunction, or a restartable error. Only one instance of a Messaging Gateway agent job runs at a given time.

Each agent uses a Scheduler job class to specify the service affinity for the agent’s Scheduler job. The job class will be configured with the database service specified by `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT`. A database administrator is responsible for setting up the database service. If no database service is specified, the default database service that maps to every instance is used.

The name of the Scheduler job class used by the Messaging Gateway default agent is `SYS.MGW_JOBCLS_DEFAULT_AGENT`. The Scheduler job used by the default agent is `SYS.MGW_JOB_DEFAULT_AGENT`.

The name of the Scheduler job class used by a Messaging Gateway named agent is `SYS.MGW_JOBCLS_<agent_name>`. The Scheduler job used by a named agent is `SYS.MGW_JOB_<agent_name>`.

If the agent job encounters an error, then the error is classified as either a restartable error or non-restartable error. A restartable error indicates a problem that might go away if the agent job were to be restarted. A non-restartable error indicates a problem that is likely to persist and be encountered again if the agent job restarts. ORA-01089 (immediate shutdown in progress) and ORA-28576 (lost RPC connection to external procedure) are examples of restartable errors. ORA-06520 (error loading external library) is an example of a non-restartable error.

Messaging Gateway uses a database shutdown trigger. If the Messaging Gateway agent is running on the instance being shut down, then the trigger notifies the agent of the shutdown, and upon receipt of the notification, the agent will terminate the current run. The job scheduler will automatically schedule the job to run again at a future time.

If a Messaging Gateway agent job instance ends because of a database malfunction or a restartable error detected by the agent job, then the job will not be removed and the job scheduler will automatically schedule the job to run again at a future time.

The `MGW_GATEWAY` view shows the agent status, database service, and the database instance on which the Messaging Gateway agent is current running. The Oracle Scheduler views provide information about Scheduler jobs, job classes, and job run details.
Configuring Messaging System Links

Running the Oracle Messaging Gateway Agent on RAC

While the Messaging Gateway job startup and shutdown principles are the same for Oracle Real Application Clusters (RAC) and non-RAC environments, there are some things to keep in mind for a RAC environment.

A single process of each configured Messaging Gateway agent can be running, even in a RAC environment. For example, if the default agent and two named agents have been configured with an agent user, then one instance of all three agents could be running at the same time. The database service associated with each agent determines the service affinity of the agent's Scheduler job, and thus, the database instance on which the agent process can run.

When a database instance is shut down in a RAC environment, the Messaging Gateway shutdown trigger will notify the agent to shut down only if the Messaging Gateway agent is running on the instance being shut down. The job scheduler will automatically schedule the job to be run again at a future time, either on another instance, or if the job can only run on the instance being shut down, when that instance is restarted.

Oracle recommends that all database connections made by the Messaging Gateway agent be made to the instance on which the Messaging Gateway agent process is running. This ensures correct failover behavior in a RAC environment.

If a Messaging Gateway agent has been associated with a database service, the agent's Scheduler job will not run unless that service is currently enabled on a running instance. When you shut down a database Oracle stops all services to that database and you may need to manually restart the services when you start the database.

See Also:

- "Configuring Oracle Messaging Gateway in a RAC Environment" on page 18-5
- "DBMS_MGWADM" and "DBMS_SCHEDULER" in Oracle Database PL/SQL Packages and Types Reference

Configuring Messaging System Links

Running as a client of non-Oracle messaging systems, the Messaging Gateway agent communicates with non-Oracle messaging systems through messaging system links. A messaging system link is a set of connections between the Messaging Gateway agent and a non-Oracle messaging system.

To configure a messaging system link of a non-Oracle messaging system, users must provide information for the agent to make connections to the non-Oracle messaging system. Users can specify the maximum number of messaging connections.

An agent name will be associated with each messaging system link. This is done when the link is created and cannot be changed. The agent associated with the link is then responsible for processing all propagation jobs that use a registered queue associated with that link. The Messaging Gateway default agent will be used if an agent name is not specified when the messaging system link is created.
When configuring a messaging system link for a non-Oracle messaging system that supports transactions and persistent messages, the native name of log queues for inbound and outbound propagation must be specified in order to guarantee exactly-once message delivery. The log queues should be used only by the Messaging Gateway agent. No other programs should enqueue or dequeue messages of the log queues. The inbound log queue and outbound log queue can refer to the same physical queue, but better performance can be achieved if they refer to different physical queues.

One and only one Messaging Gateway agent should access a propagation log queue. This insures that a given log queue contains log records for only those propagation jobs processed by that agent and that the agent is free to discard any other log records it might encounter.

When configuring a messaging system link, users can also specify an options argument. An options argument is a set of {name, value} pairs of type SYS.MGW_PROPERTY.

This section contains these topics:
- Creating a WebSphere MQ Base Java Link
- Creating a WebSphere MQ JMS Link
- Creating a WebSphere MQ Link to Use SSL
- Creating a TIB/Rendezvous Link
- Altering a Messaging System Link
- Removing a Messaging System Link
- Views for Messaging System Links

Creating a WebSphere MQ Base Java Link

A WebSphere MQ Base Java link is created by calling DBMS_MGWADM.CREATE_MGSYSTEM_LINK with the following information provided:

- Interface type: DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE
- WebSphere MQ connection information:
  - Host name and port number of the WebSphere MQ server
  - Queue manager name
  - Channel name
  - User name and password
- Maximum number of messaging connections allowed
- Log queue names for inbound and outbound propagation
- Optional information such as:
  - Send, receive, and security exits
  - Character sets

Example 19–7 configures a WebSphere MQ Base Java link mqlink. The link is configured to use the WebSphere MQ queue manager my.queue.manager on host myhost.mydomain and port 1414, using WebSphere MQ channel mychannel.

This example also sets the option to register a WebSphere MQ SendExit class. The class mySendExit must be in the CLASSPATH set in mgw.ora. The Messaging
Gateway default agent (DEFAULT_AGENT) is responsible for the link and all propagation jobs using the link.

**Example 19–7 Configuring a WebSphere MQ Base Java Link**

```sql
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
BEGIN
  v_prop := sys.mgw_mqseries_properties.construct();

  v_prop.interface_type := dbms_mgwadm.MQSERIES_BASE_JAVA_INTERFACE;
  v_prop.max_connections := 1;
  v_prop.username := 'mqm';
  v_prop.password := 'mqm';
  v_prop.hostname := 'myhost.mydomain';
  v_prop.port := 1414;
  v_prop.channel := 'mychannel';
  v_prop.queue_manager := 'my.queue.manager';
  v_prop.outbound_log_queue := 'mylogq';

  -- Specify a WebSphere MQ send exit class 'mySendExit' to be associated with
  -- the queue.
  -- Note that this is used as an example of how to use the options parameter,
  -- but is not an option that is usually set.
  v_options := sys.mgw_properties(sys.mgw_property('MQ_SendExit',
    'mySendExit'));

  dbms_mgwadm.create_msgsystem_link(
    linkname => 'mqlink', agent_name=>'default_agent', properties => v_prop,
    options => v_options );
END;
```

**See Also:**
- "Understanding the mgw.ora Initialization File" on page 18-9 for information on setting the CLASSPATH of the Messaging Gateway agent
- "WebSphere MQ System Properties" on page 19-20

**Creating a WebSphere MQ JMS Link**

A WebSphere MQ JMS link is created by calling `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` with the following information provided:

- Interface type
  - **Java Message Service** (JMS) distinguishes between queue and topic connections. The Sun Microsystems JMS 1.1 standard supports domain unification that allows both JMS queues and topics to be accessed by a single JMS connection:
  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_CONNECTION` can be used to access both JMS queues and topics. This is the recommended interface for a WebSphere MQ JMS link.
  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_QUEUE_CONNECTION` can be used to access only JMS queues.
  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_TOPIC_CONNECTION` can be used to access only JMS topics.
- WebSphere MQ connection information:
Configuring Messaging System Links

- Host name and port number of the WebSphere MQ server
- Queue manager name
- Channel name
- User name and password
- Maximum number of messaging connections allowed
  A messaging connection is mapped to a JMS session.
- Log destination (JMS queue or JMS topic) for inbound and outbound propagation
  The log destination type must be valid for the link type. JMS unified links and JMS queue links must use JMS queues for log destinations, and JMS topic links must use topics:
  - For a WebSphere MQ JMS unified or queue link, the log queue name must be the name of a physical WebSphere MQ JMS queue created using WebSphere MQ administration tools.
  - For a WebSphere MQ JMS topic link, the log topic name must be the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE. A link option can be used to specify a different physical queue.
- Optional information such as:
  - Send, receive, and security exits
  - Character sets
  - WebSphere MQ publish/subscribe configuration used for JMS topics

Example 19–8 configures a Messaging Gateway link to a WebSphere MQ queue manager using a JMS topic interface. The link is named mqjmslink and is configured to use the WebSphere MQ queue manager my.queue.manager on host myhost.mydomain and port 1414, using WebSphere MQ channel mychannel. This example also uses the options parameter to specify a nondefault durable subscriber queue to be used with the log topic. The Messaging Gateway agent myagent is responsible for the link and all propagation jobs using the link.

Example 19–8 Configuring a WebSphere MQ JMS Link

DECLARE
    v_options sys.mgw_properties;
    v_prop sys.mgw_mqseries_properties;
BEGIN
    v_prop := sys.mgw_mqseries_properties.construct();
    v_prop.max_connections := 1;
    v_prop.interface_type := DBMS_MGWADM.JMS_TOPIC_CONNECTION;
    v_prop.username := 'mqm';
    v_prop.password := 'mqm';
    v_prop.hostname := 'myhost.mydomain';
    v_prop.port := 1414;
    v_prop.channel := 'mychannel';
    v_prop.queue_manager := 'my.queue.manager';
    v_prop.outbound_log_queue := 'mylogtopic'
-- Specify a WebSphere MQ durable subscriber queue to be used with the
-- log topic.
v_options := sys.mgw_properties(
    sys.mgw_property('MQ_JMSDurSubQueue', 'myDSQueue'));

DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
    linkname        => 'mqjmslink',
    agent_name      => 'myagent',
    properties      => v_prop,
    options         => v_options );
END;

See Also:

- "Registering a WebSphere MQ JMS Queue or Topic" on page 19-13 for more information on JMS queues and topics
- "WebSphere MQ System Properties" on page 19-20

Creating a WebSphere MQ Link to Use SSL

Messaging Gateway allows SSL support for IBM WebSphere MQ and WebSphere MQ JMS connections. This section describes how to configure Messaging Gateway to use SSL for a WebSphere MQ Base Java link and the same information applies to a WebSphere MQ JMS link. There are no differences in terms of the Messaging Gateway configuration.

The following are needed in order to use SSL for WebSphere MQ connections:

- A WebSphere MQ channel configured to use SSL.
- A truststore and optionally a keystore file that are in a location accessible to the Messaging Gateway agent process. In a RAC environment, these files must be accessible to all instances on which the Messaging Gateway agent process might run, using the same path specification.
- Use DBMS_MGWADM.CREATE_MSGSYSTEM_LINK to create a WebSphere MQ link with the desired SSL related link options. At minimum, the MQ_SSLCIPHERSUITE property should be set to specify the SSL ciphersuite used by the channel.
- Use DBMS_MGWADM.SET_OPTION to set certain JSSE Java properties for the Messaging Gateway agent assigned to the link.

JSSE related properties:

- java.net.ssl.keyStore
  This property is used to specify the location of the keystore. A keystore is a database of key material used for various purposes, including authentication and data integrity.
- java.net.ssl.keyStorePassword
  This property is used to specify the password of the keystore. This password is used to check the integrity of the data in the keystore before accessing it.
- java.net.ssl.trustStore
  This property is used to specify the location of the truststore. A truststore is a keystore that is used when making decisions about which clients and servers are trusted.
- java.net.ssl.trustStorePassword
This property is used to specify the password of the truststore. This password is used to check the integrity of the data in the truststore before accessing it.

The `java.net.ssl.keyStore` and `java.net.ssl.keyStorePassword` properties are only needed if the WebSphere MQ channel is configured to use SSL client authentication.

Example 19–9 configures a WebSphere MQ Base Java link `mqssllink` to use SSL connections using the `SSL_RSA_WITH_RC4_128_MD5` ciphersuite. It assumes the channel has been configured for SSL client authentication so the Messaging Gateway agent associated with the link, `DEFAULT_AGENT`, is configured with Java properties for both a keystore and a truststore.

This configuration should be done when the Messaging Gateway agent is shut down since the Java properties set by `DBMS_MGWADM.SET_OPTION` are set only when the agent first starts. If the agent is running when the configuration is done it will need to be shutdown and restarted before the SSL connections will be used.

```
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
  v_agent varchar2(30) := 'default_agent';
BEGIN
  v_prop := sys.mgw_mqseries_properties.construct();
  v_prop.interface_type := DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE;
  v_prop.max_connections := 1;
  v_prop.username := 'mqm';
  v_prop.password := 'mqm';
  v_prop.hostname := 'myhost.mydomain';
  v_prop.port := 1414;
  v_prop.channel := 'mysslchannel';
  v_prop.queue_manager := 'my.queue.manager';
  v_prop.outbound_log_queue := 'mylogq';

  -- specify the SSL ciphersuite
  v_options := sys.mgw_properties('MQ_SSLCIPHERSUITE','SSL_RSA_WITH_RC4_128_MD5');

  -- create the MQSeries link
  DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(linkname => 'mqssllink',
    agent_name => v_agent,
    properties => v_prop,
    options => v_options);

  -- set Java properties for the agent that specify the JSSE security
  -- properties for the keystore and truststore; the paths will be
  -- saved as cleartext and the passwords encrypted

  DBMS_MGWADM.SET_OPTION(target_type => DBMS_MGWADM.AGENT_JAVA_PROP,
    target_name => v_agent,
    option_name => 'javax.net.ssl.keyStore',
    option_value => '/tmp/mq_ssl/key.jks',
    encrypted => false);

  DBMS_MGWADM.SET_OPTION(target_type => DBMS_MGWADM.AGENT_JAVA_PROP,
    target_name => v_agent,
    option_name => 'javax.net.ssl.keyStorePassword',
    option_value => 'welcome',
    encrypted => true);
```
DBMS_MGWADM.SET_OPTION(target_type => DBMS_MGWADM.AGENT_JAVA_PROP,
target_name => v_agent,
option_name => 'javax.net.ssl.trustStore',
option_value => '/tmp/mq_ssl/trust.jks',
encrypted => false);

DBMS_MGWADM.SET_OPTION(target_type => DBMS_MGWADM.AGENT_JAVA_PROP,
target_name => v_agent,
option_name => 'javax.net.ssl.trustStorePassword',
option_value => 'welcome',
encrypted => true);

END;

Creating a TIB/Rendezvous Link

A TIB/Rendezvous link is created by calling DBMS_MGWADM.CREATE_MSGSYSTEM_LINK with three parameters (service, network and daemon) for the agent to create a corresponding transport of TibrvRvdTransport type.

A TIB/Rendezvous message system link does not need propagation log queues. Logging information is stored in memory. Therefore, Messaging Gateway can only guarantee at-most-once message delivery.

Example 19–10 configures a TIB/Rendezvous link named rvlink that connects to the rvd daemon on the local computer. An agent name is not specified for the link so the Messaging Gateway default agent (DEFAULT_AGENT) is responsible for the link and all propagation jobs using the link.

Example 19–10 Configuring a TIB/Rendezvous Link

DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_tibrv_properties;
BEGIN
  v_prop := sys.mgw_tibrv_properties.construct();

  DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(linkname => 'rvlink', properties => v_prop);
END;

Altering a Messaging System Link

Using DBMS_MGWADM.ALTER_MSGSYSTEM_LINK, you can alter some link information after the link is created. You can alter link information with the Messaging Gateway agent running or shut down. Example 19–11 alters the link mqlink to change the max_connections property.

Example 19–11 Altering a WebSphere MQ Link

DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
BEGIN
  -- use alter_construct() for initialization
  v_prop := sys.mgw_mqseries_properties.alter_construct();
Configuring Non-Oracle Messaging System Queues

```sql
v_prop.max_connections := 2;

DBMS_MGWADM.ALTER_MSGSYSTEM_LINK(
    linkname => 'mqlink', properties => v_prop);
END;
```

See Also: "Configuration Properties" on page 19-20 for restrictions on changes when the Messaging Gateway agent is running

Removing a Messaging System Link

You can remove a Messaging Gateway link to a non-Oracle messaging system with DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK, but only if all registered queues associated with this link have already been unregistered. The link can be removed with the Messaging Gateway agent running or shut down. Example 19–12 removes the link mqlink.

**Example 19–12 Removing a Messaging Gateway Link**

```sql
BEGIN
    dbms_mgwadm.remove_msgsystem_link(linkname =>'mqlink');
END;
```

Views for Messaging System Links

You can use the MGW_LINKS view to check links that have been created. It lists the name and link type, as shown in Example 19–13.

**Example 19–13 Listing All Messaging Gateway Links**

```sql
SQL> select link_name, link_type from MGW_LINKS;

LINK_NAME     LINK_TYPE
------------------------
MQLINK        MQSERIES
RVLINK        TIBRV
```

You can use the MGW_MQSERIES_LINKS and MGW_TIBRV_LINKS views to check messaging system type-specific configuration information, as shown in Example 19–14.

**Example 19–14 Checking Messaging System Link Configuration Information**

```sql
SQL> select link_name, queue_manager, channel, hostname from mgw_mqseries_links;

LINK_NAME   QUEUE_MANAGER      CHANNEL     HOSTNAME
----------------------------------------------------------
MQLINK      my.queue.manager  mychannel  myhost.mydomain
```

**Configuring Non-Oracle Messaging System Queues**

All non-Oracle messaging system queues involved in propagation as a source queue, destination queue, or exception queue must be registered through the Messaging Gateway agent.
Gateway administration interface. You do not need to register Oracle® Database queues involved in propagation.

This section contains these topics:

- Registering a Non-Oracle Queue
- Unregistering a Non-Oracle Queue
- View for Registered Non-Oracle Queues

Registering a Non-Oracle Queue

You can register a non-Oracle queue using DBMS_MGWADM.REGISTER_FOREIGN_QUEUE. Registering a non-Oracle queue provides information for the Messaging Gateway agent to access the queue. However, it does not create the physical queue in the non-Oracle messaging system. The physical queue must be created using the non-Oracle messaging system administration interfaces before the Messaging Gateway agent accesses the queue.

The following information is used to register a non-Oracle queue:

- Name of the messaging system link used to access the queue
- Native name of the queue (its name in the non-Oracle messaging system)
- Domain of the queue
  - DBMS_MGWADM.DOMAIN_QUEUE for a point-to-point queue
  - DBMS_MGWADM.DOMAIN_TOPIC for a publish/subscribe queue
- Options specific to the non-Oracle messaging system
  These options are a set of {name, value} pairs, both of which are strings.

See Also: "Optional Foreign Queue Configuration Properties" on page 19-25

Example 19–15 shows how to register the WebSphere MQ Base Java queue my_mq_queue as a Messaging Gateway queue destq.

Example 19–15 Registering a WebSphere MQ Base Java Queue

BEGIN
    DBMS_MGWADM.REGISTER_FOREIGN_QUEUE(
        name => 'destq',
        linkname => 'mqlink',
        provider_queue => 'my_mq_queue',
        domain => DBMS_MGWADM.DOMAIN_QUEUE);
END;

Registering a WebSphere MQ Base Java Queue

The domain must be DBMS_MGWADM.DOMAIN_QUEUE or NULL, because only point-to-point queues are supported for WebSphere MQ.

Registering a WebSphere MQ JMS Queue or Topic

When registering a WebSphere MQ JMS queue, the domain must be DBMS_MGWADM.DOMAIN_QUEUE, and the linkname parameter must refer to a WebSphere MQ JMS unified link or queue link.
When registering a WebSphere MQ JMS topic, the domain must be `DBMS_MGWADM.DOMAIN_TOPIC`, and the `linkname` parameter must refer to a WebSphere MQ JMS unified link or topic link. The `provider_queue` for a WebSphere MQ JMS topic used as a propagation source may include wildcards. See WebSphere MQ documentation for wildcard syntax.

**Registering a TIB/Rendezvous Subject**

When registering a TIB/Rendezvous subject with Messaging Gateway, the `provider_queue` parameter specifies a TIB/Rendezvous subject name. The domain of a registered TIB/Rendezvous queue must be `DBMS_MGWADM.DOMAIN_TOPIC` or `NULL`.

A registered TIB/Rendezvous queue with `provider_queue` set to a wildcard subject name can be used as a propagation source queue for inbound propagation. It is not recommended to use queues with wildcard subject names as propagation destination queues or exception queues. As documented in TIB/Rendezvous, sending messages to wildcard subjects can trigger unexpected behavior. However, neither Messaging Gateway nor TIB/Rendezvous prevents you from doing so.

**Unregistering a Non-Oracle Queue**

A non-Oracle queue can be unregistered with `DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE`, but only if there are no propagation jobs referencing it.

Example 19–16 unregisters the queue `destq` of the link `mqlink`.

**Example 19–16  Unregistering a Non-Oracle Queue**

```sql
BEGIN
  DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE(name =>'destq', linkname=>'mqlink');
END;
```

**View for Registered Non-Oracle Queues**

You can use the `MGW_FOREIGN_QUEUES` view to check which non-Oracle queues are registered and what link each uses, as shown in Example 19–17.

**Example 19–17  Checking Which Queues Are Registered**

```sql
SELECT name, link_name, provider_queue FROM MGW_FOREIGN_QUEUES;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>LINK_NAME</th>
<th>PROVIDER_QUEUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESTQ</td>
<td>MQLINK</td>
<td>my_mq_queue</td>
</tr>
</tbody>
</table>

**Configuring Oracle Messaging Gateway Propagation Jobs**

Propagating messages between an Oracle Streams AQ queue and a non-Oracle messaging system queue requires a propagation job. Each propagation job will have a unique propagation type, source, and destination triplet.

You can create a propagation job to propagate messages between JMS destinations. You can also create a propagation job to propagate messages between non-JMS queues. Messaging Gateway does not support message propagation between a JMS destination and a non-JMS queue.

This section contains these topics:
Propagation Job Overview

A propagation job specifies what messages are propagated and how the messages are propagated.

Messaging Gateway allows bidirectional message propagation. An outbound propagation moves messages from Oracle® Database to non-Oracle messaging systems. An inbound propagation moves messages from non-Oracle messaging systems to Oracle® Database.

If the propagation source is a queue (point-to-point), then the Messaging Gateway agent moves all messages from the source queue to the destination queue. If the propagation source is a topic (publish/subscribe), then the Messaging Gateway agent creates a subscriber of the propagation source queue in the messaging system. The agent only moves messages that are published to the source queue after the subscriber is created.

When propagating a message, the Messaging Gateway agent converts the message from the format in the source messaging system to the format in the destination messaging system. Users can customize the message conversion by providing a message transformation. If message conversion fails, then the message will be moved to an exception queue, if one has been provided, so that the agent can continue to propagate messages for the subscriber.

A Messaging Gateway exception queue is different from an Oracle® Database exception queue. Messaging Gateway moves a message to a Messaging Gateway exception queue when message conversion fails. Oracle® Database moves a message to an Oracle® Database exception queue after MAX_RETRIES dequeue attempts on the message.

Messages moved to an Oracle® Database exception queue may result in irrecoverable failures on the associated Messaging Gateway propagation job. To avoid the problem, the MAX_RETRIES parameter of any Oracle® Database queue that is used as the propagation source of a Messaging Gateway propagation job should be set to a value much larger than 16.

If the messaging system of the propagation source queue supports message selection, then a message selection rule can be specified for a propagation subscriber. Only messages that satisfy the message selector will be propagated.

Users can also specify propagation job options to control how messages are propagated, such as options for JMS message delivery mode and TIB/Rendezvous queue policies.

The MGW_JOBS view can be used to check the configuration and status of Messaging Gateway propagation jobs.

See Also: Chapter 21, "Monitoring Oracle Messaging Gateway"
Creating an Oracle Messaging Gateway Propagation Job

Messaging Gateway propagation jobs are created by `DBMS_MGWADM.CREATE_JOB`.

If the propagation source for non-JMS propagation is an Oracle Streams AQ queue, then the queue can be either a single consumer queue or multiple consumer queue. If it is a multiple consumer queue, Messaging Gateway creates a corresponding Oracle Streams AQ subscriber `MGW_job_name` for the propagation job `job_name` when `DBMS_MGWADM.CREATE_JOB` is called.

If the propagation source is a JMS topic, such as an Oracle Java Message Service (OJMS) topic or a WebSphere MQ JMS topic, then a JMS subscriber `MGW_job_name` is created on the topic in the source messaging system by the Messaging Gateway agent. If the agent is not running, then the subscriber will not be created until the agent is restarted.

If the propagation source is a queue, then only one propagation job can be created using that queue as the propagation source. If the propagation source is a topic, then multiple propagation jobs can be set up using that topic as the propagation source with each propagation job having its own corresponding subscriber on the topic in the messaging system.

Example 19–18 creates Messaging Gateway propagation job `job_aq2mq`.

### Example 19–18 Creating a Messaging Gateway Propagation Job

```sql
BEGIN
    DBMS_MGWADM.CREATE_JOB(
        job_name            =>  'job_aq2mq',
        propagation_type    =>   DBMS_MGWADM.OUTBOUND_PROPAGATION,
        source              =>  'mquser.srcq',
        destination         =>  'deqq@mqlink');
END;
```

**Note:** If a WebSphere MQ JMS topic is involved in a propagation job and the interface type of the link is `DBMS_MGWADM.JMS_TOPIC_CONNECTION`, then a durable subscriber `MGL_subscriber_id` is created on the log topic. The durable subscriber is removed when the Messaging Gateway propagation job is successfully removed.

Enabling and Disabling a Propagation Job

A propagation job can be initially enabled or disabled when it is created by `DBMS_MGWADM.CREATE_JOB`. By default, a job is enabled when it is created. You can use `DBMS_MGWADM.ENABLE_JOB` to enable a propagation job and `DBMS_MGWADM.DISABLE_JOB` to disable a job. No propagation processing will occur when the job is disabled.

Example 19–19 enables the propagation for propagation job `job_aq2mq`.

### Example 19–19 Enabling a Messaging Gateway Propagation Job

```sql
BEGIN
    DBMS_MGWADM.ENABLE_JOB(job_name => 'job_aq2mq');
END;
```

Example 19–20 disables the propagation for propagation job `job_aq2mq`.

```sql
BEGIN
    DBMS_MGWADM.DISABLE_JOB(job_name => 'job_aq2mq');
END;
```
Example 19–20  Disabling a Messaging Gateway Propagation Job

BEGIN
  DBMS_MGWADM.DISABLE_JOB(job_name => 'job_aq2mq');
END;

Resetting a Propagation Job

When a problem occurs with a propagation job, the Messaging Gateway agent retries the failed operation up to 16 times in an exponential backoff scheme before the propagation job stops. You can use DBMS_MGWADM.RESET_JOB to reset the failure count to zero to allow the agent to retry the failed operation immediately.

Example 19–21 resets the failure count for propagation job job_aq2mq.

Example 19–21  Resetting a Propagation Job

BEGIN
  DBMS_MGWADM.RESET_JOB (job_name => 'job_aq2mq');
END;

Altering a Propagation Job

After a propagation job is created you can alter the selection rule, transformation, exception queue, job options, and poll interval of the job using DBMS_MGWADM.ALTER_JOB. The job can be altered with the Messaging Gateway running or shut down.

Example 19–22 adds an exception queue for a propagation job.

Example 19–22  Altering Propagation Job by Adding an Exception Queue

BEGIN
  DBMS_MGWADM.ALTER_JOB(
    job_name         =>  'job_aq2mq',
    exception_queue  =>  'mgwuser.my_ex_queue');
END;

Example 19–23 changes the polling interval for a propagation job. The polling interval determines how soon the agent can discover the available messages in the propagation source queue. The default polling interval is 5 seconds or the value set for oracle.mgw.polling_interval in the Messaging Gateway initialization file.

Example 19–23  Altering Propagation Job by Changing the Polling Interval

BEGIN
  DBMS_MGWADM.ALTER_JOB(
    job_name       =>  'job_aq2mq',
    poll_interval  =>  2);
END;

Removing a Propagation Job

You can remove a Messaging Gateway propagation job by calling DBMS_MGWADM.REMOVE_JOB.

Before removing the propagation job from the Messaging Gateway configuration, Messaging Gateway does the following cleanup:

- Removes from the messaging system the associated subscriber that may have been created by Messaging Gateway
Removes propagation log records from log queues for the job being removed. Messaging Gateway may fail to do the cleanup because:

- The Messaging Gateway agent is not running
- Non-Oracle messaging system is not running
- The Messaging Gateway agent is unable to interact with the source or destination messaging system

If the Messaging Gateway cleanup fails for any reason, then the propagation job being removed is placed in a `DELETE_PENDING` state. The Messaging Gateway agent tries to clean up propagation jobs in a `DELETE_PENDING` state when:

- `DBMS_MGWADM.REMOVE_JOB` is called and the Messaging Gateway agent is running.

- The Messaging Gateway agent is starting and finds a propagation job in a `DELETE_PENDING` state.

`DBMS_MGWADM.REMOVE_JOB` has a force parameter that allows you to force the propagation job to be removed from the Messaging Gateway configuration without placing it in `DELETE_PENDING` state. This is useful in case of cleanup failures or if you want to remove a propagation job when the Messaging Gateway agent is not running.

Forcing a propagation job to be removed may result in obsolete log records being left in the log queues, and subscriptions in the messaging systems that may cause unnecessary message accumulation. Oracle recommends that the force option not be used for `DBMS_MGWADM.REMOVE_JOB` if possible.

Example 19–24 removes a propagation job in a non-forced manner.

**Example 19–24 ** Removing a Propagation Job

```sql
BEGIN
    DBMS_MGWADM.REMOVE_JOB (job_name  =>  'job_aq2mq');
END;
```

### Propagation Jobs, Subscribers, and Schedules

Subprograms are provided as part of the `DBMS_MGWADM` package that simplify the creation and management of propagation jobs. Those subprograms allow a user to configure a propagation job rather than a disjoint subscriber and schedule as was done in prior releases. Oracle recommends that you use the propagation job procedures but still supports the subscriber and schedule procedures for backward compatibility.

**Table 19–1** lists the Messaging Gateway propagation job procedures and shows which subscriber and/or schedule procedures it replaces. All procedures are from the `DBMS_MGWADM` package.

**Table 19–1 Messaging Gateway Propagation Job Subprograms**

<table>
<thead>
<tr>
<th>Job Procedure</th>
<th>Replaces Subscriber, Schedule Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_JOB</td>
<td>ADD_SUBSCRIBER, SCHEDULE_PROPAGATION</td>
</tr>
<tr>
<td>ALTER_JOB</td>
<td>ALTER_SUBSCRIBER, ALTER_PROPAGATION_SCHEDULE</td>
</tr>
<tr>
<td>REMOVE_JOB</td>
<td>REMOVE_SUBSCRIBER, UNSCHEDULE_PROPAGATION</td>
</tr>
<tr>
<td>ENABLE_JOB</td>
<td>ENABLE_PROPAGATION_SCHEDULE</td>
</tr>
<tr>
<td>DISABLE_JOB</td>
<td>DISABLE_PROPAGATION_SCHEDULE</td>
</tr>
</tbody>
</table>
Propagation Jobs, Subscribers, and Schedules

This section contains the following topics:

- Propagation Job, Subscriber, Schedule Interface Interoperability
- Propagation Job, Subscriber, Schedule Views
- Single Consumer Queue As Propagation Source

Propagation Job, Subscriber, Schedule Interface Interoperability

The user can create two types of propagation jobs, a new style job or an old style job. A new style job is created by `DBMS_MGWADM.CREATE_JOB`. An old style job is created by calling `DBMS_MGWADM.ADD_SUBSCRIBER` and `DBMS_MGWADM.SCHEDULE_PROPAGATION` using the same `{propagation_type, source, destination}` triplet. A subscriber that does not have a matching schedule, or a schedule that does not have a matching subscriber, is not considered to be a propagation job.

For new style job, the job name will serve as both the subscriber ID and the schedule ID. For an old style job, the subscriber ID is used as the job name.

Both the propagation job subprograms and the subscriber/schedule subprograms can be used for old style propagation jobs. Oracle recommends that you use the job subprograms to create and manage propagation jobs. The job subprograms cannot be used for an unmatched subscriber or schedule since those do not constitute a propagation job.

Only the new job subprograms can be used for new style propagation jobs. An error will occur if a user tries to call a subscriber or scheduler procedure on a new style job.

Other than `DBMS_MGWADM.REMOVE_JOB`, calling the job subprograms for an old style job is straightforward and the results are effectively the same as calling the corresponding subscriber/schedule subprograms. There may be certain restrictions in the future but there are none at this time.

The `DBMS_MGWADM.REMOVE_JOB` procedure can be used to remove both new style and old style jobs. A forced and non-forced remove is supported. If the Messaging Gateway agent is not running when a non-forced remove is done, the job will be flagged as delete pending and neither the underlying subscriber nor schedule will be removed at that time. The job (subscriber/schedule pair) will be removed once the agent is restarted and performs its cleanup work or a forced `DBMS_MGWADM.REMOVE_JOB` is performed. In order to insure that the subscriber/schedule pair is removed at the same time, an error will occur if you first call `DBMS_MGWADM.REMOVE_JOB` and subsequently attempt to call `DBMS_MGWADM.REMOVE_SUBSCRIBER` or `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` for an old style job.

Once `DBMS_MGWADM.REMOVE_JOB` as been called for a job and it has been flagged as delete pending, all job procedures, other than `DBMS_MGWADM.REMOVE_JOB`, will fail for both new style and old style jobs. In addition, all subscriber and schedule subprograms will fail if the propagation job happens to be an old style job. `DBMS_MGWADM.REMOVE_SUBSCRIBER` and `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` can be used for an old style job as long as `DBMS_MGWADM.REMOVE_JOB` has not been called for that job. If `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` is called for an old style job, the schedule is immediately removed and it ceases to be a propagation job and `DBMS_MGWADM.REMOVE_SUBSCRIBER` must be used to remove the subscriber. If `DBMS_MGWADM.REMOVE_REFERER` is called for a new style job, the propagation job is removed but the subscriber will remain in existence until the next time the agent is restarted.

---

<table>
<thead>
<tr>
<th>Table 19–1 (Cont.) Messaging Gateway Propagation Job Subprograms</th>
<th>Replaces Subscriber, Schedule Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET_JOB</td>
<td>RESET_SUBSCRIBER</td>
</tr>
</tbody>
</table>
MGWADM.REMOVE_SUBSCRIBER is called for an old style job, the user can subsequently call DBMS_MGWADM.REMOVE_JOB as long as the subscriber exists.

Propagation Job, Subscriber, Schedule Views

The MGW_JOBS view shows information for the current propagation jobs, both new style jobs and old style jobs, and includes all the pertinent information shown by the MGW_SUBSCRIBERS and MGW_SCHEDULES views. The MGW_SUBSCRIBERS and MGW_SCHEDULES views are still useful for finding an unmatched subscriber or schedule since they don't constitute a propagation job and will not show up in the MGW_JOBS view.

Single Consumer Queue As Propagation Source

Messaging Gateway allows an Oracle Streams AQ multiple consumer queue or a single consumer queue to be a propagation source for an outbound new style job created by DBMS_MGWADM.CREATE_JOB. A multiple consumer queue must be used for the propagation source for an outbound old style job. An error will occur if an administrator attempts to call DBMS_MGWADM.ADD_SUBSCRIBER and the source is a single consumer queue.

An Oracle Streams AQ dequeue condition is not supported for native (non-JMS) outbound propagation when the propagation source is a single consumer queue.

Configuration Properties

This section summarizes basic and optional properties related to Messaging Gateway links, foreign queues, and propagation jobs.

This section contains these topics:

- WebSphere MQ System Properties
- TIB/Rendezvous System Properties
- Optional Link Configuration Properties
- Optional Foreign Queue Configuration Properties
- Optional Job Configuration Properties

WebSphere MQ System Properties

Table 19–2 summarizes the basic configuration properties for a WebSphere MQ messaging link. The table indicates which properties of SYS.MGW_MQSERIES_PROPERTIES are optional (NULL allowed), which can be altered, and if alterable, which values can be dynamically changed.

See Also: "SYS.MGW_MQSERIES_PROPERTIES Type" in Oracle Database PL/SQL Packages and Types Reference

<table>
<thead>
<tr>
<th>Attribute</th>
<th>NULL Allowed?</th>
<th>Alter Value?</th>
<th>Dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_manager</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>hostname</td>
<td>yes (1)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>port</td>
<td>yes (1)</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Notes on Table 19–2

1. If hostname is NULL, then the port and channel must be NULL. If the hostname is not NULL, then the port must be not NULL. If the hostname is NULL, then a WebSphere MQ bindings connection is used; otherwise a client connection is used.

2. If interface_type is NULL, then a default value of DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE is used.

3. If max_connections is NULL, then a default value of 1 is used.

4. Attribute inbound_log_queue can be NULL if the link is not used for inbound propagation. The log queue can be altered only when no inbound propagation job references the link.

5. Attribute outbound_log_queue can be NULL if the link is not used for outbound propagation. The log queue can be altered only when no outbound propagation job references the link.

6. The channel attribute must be NULL if a client channel definition table (CCDT) is used. The MQ_ccdtURL link option can be used to specify a CCDT.

Table 19–3 summarizes the optional configuration properties supported when a WebSphere MQ Base Java interface is used to access the WebSphere MQ messaging system. Table 19–4 summarizes the optional configuration properties supported when a WebSphere MQ JMS interface is used. Each table lists the property name, where that property applies, whether the property can be altered, and if alterable, whether the value can be dynamically changed. Only the properties listed in the tables are supported, and any extra properties are ignored.
Table 19–3 (Cont.) Optional Configuration Properties for WebSphere MQ Base Java

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Used For</th>
<th>Alter Value?</th>
<th>Dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ_openOptions</td>
<td>foreign queue</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MsgBatchSize</td>
<td>job</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PreserveMessageID</td>
<td>job</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 19–4 Optional Configuration Properties for WebSphere MQ JMS

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Used For</th>
<th>Alter Value?</th>
<th>Dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ_BrokerControlQueue</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_BrokerPubQueue</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_BrokerQueueManager</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_BrokerVersion</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_ccdtUrl</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_ccsid</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_JmsDurSubQueue</td>
<td>link</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MQ_PubAckInterval</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_ReceiveExit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_ReceiveExitInit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SecurityExit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SecurityExitInit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SendExit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SendExitInit</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SSLCipherSuite</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SSLCrl</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SSLFipsRequired</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SSLPeerName</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_SSLResetCount</td>
<td>link</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>MQ_CharacterSet</td>
<td>foreign queue</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MQ_JmsDurSubQueue</td>
<td>foreign queue</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MQ_JmsTargetClient</td>
<td>foreign queue</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>JMS_DeliveryMode</td>
<td>job</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>JMS_NoLocal</td>
<td>job</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MsgBatchSize</td>
<td>job</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PreserveMessageID</td>
<td>job</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

TIB/Rendezvous System Properties

Table 19–5 summarizes the basic configuration properties for a TIB/Rendezvous messaging link. It indicates which properties of SYS.MGW_TIBRV_PROPERTIES are optional (NULL allowed), which can be altered, and if alterable, which values can be dynamically changed.
See Also: "SYS.MGW_TIBRV_PROPERTIES Type" in Oracle Database PL/SQL Packages and Types Reference

Table 19–5  TIB/Rendezvous Link Properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>NULL allowed?</th>
<th>Alter value?</th>
<th>Dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>yes(1)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>daemon</td>
<td>yes(1)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>network</td>
<td>yes(1)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>cm_name</td>
<td>yes(2)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>cm_ledger</td>
<td>yes(2)</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Notes on Table 19–5:

1. System default values will be used if service, daemon, or network are NULL.
2. The cm_name and cm_ledger attributes are reserved for future use when TIB/Rendezvous certified messages are supported. At present, a NULL must be specified for these parameters when a TIB/Rendezvous link is configured.

Table 19–6 summarizes the optional configuration properties supported when a TIB/Rendezvous messaging system is used. The table lists the property name, where that property applies, whether the property can be altered, and if alterable, whether the value can be dynamically changed. Only the properties listed in the table are supported, and any extra properties will be ignored.

Optional Link Configuration Properties

This section describes optional link properties you can specify using the options parameter of DBMS_MGWADM.CREATE_MSGSYSTEM_LINK and DBMS_MGWADM.ALTER_MSGSYSTEM_LINK. Each listing also indicates which messaging system might use that property.

MQ_BrokerControlQueue
This property is used by WebSphere MQ JMS. It specifies the name of the broker control queue and corresponds to WebSphere MQ JMS administration tool property BROKERCONQ. The WebSphere MQ default is SYSTEM.BROKER.CONTROL.QUEUE.

MQ_BrokerPubQueue
This property is used by WebSphere MQ JMS. It specifies the name of the broker publish queue and corresponds to WebSphere MQ JMS administration tool property BROKERPUBQ. The WebSphere MQ default is SYSTEM.BROKER.DEFAULT.STREAM.
**MQ_BrokerQueueManager**
This property is used by WebSphere MQ JMS. It specifies the name of the broker queue manager and corresponds to WebSphere MQ administration tool property BROKERQMG. If it is not set, then no default is used.

**MQ_BrokerVersion**
This property is used by WebSphere MQ JMS. It specifies the broker version number and corresponds to WebSphere MQ JMS administration tool property BROKERVER. The WebSphere MQ default is 0.

**MQ_ccdtUrl**
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the URL string of a client channel definition table (CCDT) to be used. If not set, a CCDT is not used. If a CCDT is used, then the SYS.MGW_MQSERIES_PROPERTIES.channel link property must be NULL.

**MQ_ccsid**
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the character set identifier to be used to translate information in the WebSphere MQ message header. This should be the integer value of the character set (for example, 819) rather than a descriptive string. If it is not set, then the WebSphere MQ default character set 819 is used.

**MQ_JmsDurSubQueue**
This property is used by WebSphere MQ JMS. It applies to WebSphere MQ JMS topic links only. The SYS.MGW_MQSERIES_PROPERTIES attributes, inbound_log_queue and outbound_log_queue, specify the names of WebSphere MQ JMS topics used for propagation logging. This property specifies the name of the WebSphere MQ queue from which durable subscription messages are retrieved by the log topic subscribers. The WebSphere MQ default queue is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.

**MQ_PubAckInterval**
This property is used by WebSphere MQ JMS. It specifies the interval, in number of messages, between publish requests that require acknowledgment from the broker and corresponds to WebSphere MQ JMS administration tool property PUBACKINT. The WebSphere MQ default is 25.

**MQ_ReceiveExit**
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the MQReceiveExit interface. This class must be in the CLASSPATH of the Messaging Gateway agent. There is no default.

**MQ_ReceiveExitInit**
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by MQ_ReceiveExit and corresponds to WebSphere MQ JMS administration tool property RECEXITINIT. There is no default.

**MQ_SecurityExit**
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the MQSecurityExit interface. This class must be in the CLASSPATH of the Messaging Gateway agent. There is no default.
MQ_SecurityExitInit
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by MQ_SecurityExit and corresponds to WebSphere MQ JMS administration tool property SECEXITINIT. There is no default.

MQ_SendExit
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the MQSendExit interface. This class must be in the CLASSPATH of the Messaging Gateway agent. There is no default.

MQ_SendExitInit
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by MQ_SendExit. It corresponds to WebSphere MQ JMS administration tool property SENDEXITINIT. There is no default.

MQ_SSLCipherSuite
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the CipherSuite to be used; for example, SSL_RSA_WITH_RC4_128_MD5. This corresponds to the WebSphere MQ SSLCIPHERSUITE administration property.

MQ_SSLCrl
This property is used by WebSphere MQ JMS. It specifies a space-delimited list of LDAP servers that can be used for certificate revocation list (CRL) checking. If not set, no CRL checking is done. This corresponds to the WebSphere MQ SSLCRL administration property. This option is not supported for WebSphere MQ Base Java, and instead, a client channel definition table (CCDT) must be used if CRL checking is needed.

MQ_SSLFipsRequired
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It indicates whether the CipherSuite being used is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS). The value should be TRUE or FALSE. The default value is FALSE. This corresponds to the WebSphere MQ SSLFIPSREQUIRED administration property.

MQ_SSLPeerName
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies a distinguished name (DN) pattern that the queue manager certificate must match in order for a connection to be established. If not set, no DN check is performed. This corresponds to the WebSphere MQ SSLPEERNAME administration property.

MQ_SSLResetCount
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the total number of bytes sent and received before the secret key is renegotiated. If not set, the key is not renegotiated. This corresponds to the WebSphere MQ SSLRESETCOUNT administration property.

Optional Foreign Queue Configuration Properties
This section describes optional foreign queue properties that you can specify using the options parameter of DBMS_MGWADM.REGISTERFOREIGNQUEUE. Each listing also indicates which messaging system might use that property.
**MQ_CharacterSet**
This property is used by WebSphere MQ JMS. It is used only for outbound propagation to a JMS queue or topic. It specifies the character set to be used to encode text strings sent to the destination. It should be the integer value of the character set (for example, 1208) rather than a descriptive string. The default value used by Messaging Gateway is 1208 (UTF8).

**MQ_JmsDurSubQueue**
This property is used by WebSphere MQ JMS. It is a string representing the name of the WebSphere MQ queue from which durable subscription messages are retrieved by subscribers on this topic. It applies only to WebSphere MQ JMS topics. The WebSphere MQ default queue is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.

**MQ_JmsTargetClient**
This property is used by WebSphere MQ JMS. It is used only for outbound propagation to a JMS queue or topic. Supported values are TRUE and FALSE. TRUE indicates that WebSphere MQ should store the message as a JMS message. FALSE indicates that WebSphere MQ should store the message in non-JMS format so that non-JMS applications can access it. Default is TRUE.

**MQ_openOptions**
This property is used by WebSphere MQ Base Java. It specifies the value used for the openOptions argument of the WebSphere MQ Base Java MQQueueManager.accessQueue method. No value is required. But if one is given, then the Messaging Gateway agent adds MQOO_OUTPUT to the specified value for an enqueue (put) operation. MQOO_INPUT_SHARED is added for a dequeue (get) operation. The default is MQOO_OUTPUT for an enqueue/put operation; MQOO_INPUT_SHARED for a dequeue/get operation.

### Optional Job Configuration Properties

This section describes optional propagation job properties that you can specify using the options parameter of DBMS_MGWADM.CREATE_JOB and DBMS_MGWADM.ALTER_JOB.

**AQ_MsgProperties**
This property is used by TIB/Rendezvous. It specifies how Oracle Streams AQ message properties will be used during message propagation. Supported values are TRUE and FALSE. The default value is FALSE.

For an outbound propagation job, if the value is TRUE (case insensitive), then the Messaging Gateway agent will add a field for most Oracle Streams AQ message properties to the message propagated to the TIB/Rendezvous subject.

For an inbound propagation job, if the value is TRUE (case insensitive), then the Messaging Gateway agent will search the source message for a field with a reserved name, and if it exists, use its value to set the corresponding Oracle Streams AQ message property. A default value will be used if the field does not exist or does not have an expected datatype.

**JMS_DeliveryMode**
This property is used by WebSphere MQ JMS and Oracle JMS. You can use this property when the propagation destination is a JMS messaging system. It sets the delivery mode of messages enqueued to the propagation destination queue by a JMS MessageProducer. The default is PRESERVE_MSG. Supported values and their associated delivery modes are:

- PERSISTENT (DeliveryMode.getPERSISTENT)
- **NON_PERSISTENT** (DeliveryMode.NON_PERSISTENT)
- **PRESERVE_MSG** (delivery mode of the source JMS message is used)

**JMS_NoLocal**
This property is used by WebSphere MQ JMS and Oracle JMS. You can use it when the propagation source is a JMS messaging system. It sets the noLocal parameter of a JMS TopicSubscriber. TRUE indicates that messages that have been published to this topic through the same Messaging Gateway link will not be propagated. The default value FALSE indicates that such messages will be propagated from the topic.

**MsgBatchSize**
This property can be used by any supported messaging system. It specifies the maximum number of messages, if available, to be propagated in one transaction. The default is 30.

**PreserveMessageID**
This property is used by WebSphere MQ Base Java, WebSphere MQ JMS, TIB/Rendezvous, and Oracle JMS. It specifies whether Messaging Gateway should preserve the original message identifier when the message is propagated to the destination messaging system. The exact details depend on the capabilities of the messaging systems involved. Supported values are TRUE and FALSE. The default value is FALSE.

**RV_discardAmount**
This property is used by TIB/Rendezvous. It specifies the discard amount of a queue. It is meaningful only for an inbound propagation job. The default is 0.

**RV_limitPolicy**
This property is used by TIB/Rendezvous. It specifies the limit policy for resolving overflow of a queue limit. It is meaningful only for an inbound propagation job. The default is DISCARD_NONE. Supported values and their associated limit policies are: DISCARD_NONE, DISCARD_FIRST, DISCARD_LAST and DISCARD_NEW.
- **DISCARD_NONE** (TibrvQueue.DISCARD_NONE)
- **DISCARD_FIRST** (TibrvQueue.DISCARD_FIRST)
- **DISCARD_LAST** (TibrvQueue.DISCARD_LAST)
- **DISCARD_NEW** (TibrvQueue.DISCARD_NEW)

**RV_maxEvents**
This property is used by TIB/Rendezvous. It specifies the maximum event limit of a queue. It is meaningful only for an inbound propagation job. The default is 0.
This chapter discusses how Oracle Messaging Gateway (MGW) converts message formats from one messaging system to another. A conversion is generally necessary when moving messages between Oracle® Database and another system, because different messaging systems have different message formats. Java Message Service (JMS) messages are a special case. A JMS message can be propagated only to a JMS destination, making conversion a simple process.

This chapter contains these topics:

- Converting Oracle Messaging Gateway Non-JMS Messages
- Message Conversion for WebSphere MQ
- Message Conversion for TIB/Rendezvous
- JMS Messages

Converting Oracle Messaging Gateway Non-JMS Messages

MGW converts the native message format of the source messaging system to the native message format of the destination messaging system during propagation. MGW uses canonical types and a model centering on Oracle® Database for the conversion.

Overview of the Non-JMS Message Conversion Process

When a message is propagated by MGW, the message is converted from the native format of the source queue to the native format of the destination queue.

A native message usually contains a message header and a message body. The header contains the fixed header fields that all messages in that messaging system have, such as message properties in Oracle® Database and the fixed header in WebSphere MQ. The body contains message contents, such as the Oracle® Database payload, the WebSphere MQ message body, or the entire TIB/Rendezvous message. MGW converts both message header and message body components.

Figure 20–1 shows how non-JMS messages are converted in two stages. A message is first converted from the native format of the source queue to the MGW internal message format, and then it is converted from the internal message format to the native format of the destination queue.
The MGW agent uses an internal message format consisting of a header that is similar to the Oracle® Database message properties and a body that is a representation of an MGW canonical type.

### Oracle Messaging Gateway Canonical Types

MGW defines canonical types to support message conversion between Oracle® Database and non-Oracle messaging systems. A canonical type is a message type representation in the form of a PL/SQL Oracle type in Oracle Database. The canonical types are `RAW`, `SYS.MGW_BASIC_MSG_T`, and `SYS.MGW_TIBRV_MSG_T`.

WebSphere MQ propagation supports the canonical types `SYS.MGW_BASIC_MSG_T` and `RAW`. TIB/Rendezvous propagation supports the canonical types `SYS.MGW_TIBRV_MSG_T` and `RAW`.

See Also: "DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference* for Syntax and attribute information for `SYS.MGW_BASIC_MSG_T` and `SYS.MGW_TIBRV_MSG_T`.

### Message Header Conversion

MGW provides default mappings between Oracle® Database message properties and non-Oracle message header fields that have a counterpart in Oracle® Database message properties with the same semantics. Where MGW does not provide a mapping, the message header fields are set to a default value, usually the default value defined by the messaging system.

### Handling Arbitrary Payload Types Using Message Transformations

When converting to or from Oracle® Database messages, the MGW agent uses only its canonical types. Arbitrary payload types are supported, however, with the assistance of user-defined Oracle® Database message transformations to convert between an Oracle® Database queue payload and an MGW canonical type.

For MGW to propagate messages from an Oracle® Database queue with an arbitrary ADT payload (outbound propagation), you must provide a mapping to an MGW canonical ADT. The transformation is invoked when the MGW agent dequeues messages from the Oracle® Database queue. Similarly, for MGW to propagate...
messages to an Oracle® Database queue with an arbitrary ADT payload (inbound propagation), you must provide a mapping from an MGW canonical ADT. The transformation is invoked when the MGW agent enqueues messages to the Oracle® Database queue.

The transformation is always executed in the context of the MGW agent, which means that the MGW agent user (the user specified using DBMS_MGWADM.CREATE_AGENT or DBMS_MGWADM.ALTER_AGENT) must have EXECUTE privileges on the transformation function and the Oracle® Database payload type. This can be accomplished by granting the EXECUTE privilege to PUBLIC or by granting the EXECUTE privilege directly to the MGW agent user.

To configure a MGW propagation job with a transformation:

1. Create the transformation function.
2. Grant EXECUTE to the MGW agent user or to PUBLIC on the function and the object types it references.
3. Call DBMS_TRANSFORM.CREATE_TRANSFORMATION to register the transformation.
4. Call DBMS_MGWADM.CREATE_JOB to create a MGW propagation job using the transformation, or DBMS_MGWADM.ALTER_JOB to alter an existing job.

The value passed in the transformation parameter for these APIs must be the registered transformation name and not the function name. For example, trans_sampleadt_to_mgw_basic is a stored procedure representing a transformation function with the signature shown in Example 20–1.

**Example 20–1 Transformation Function Signature**

```sql
FUNCTION trans_sampleadt_to_mgw_basic(in_msg IN mgwuser.sampleADT)
RETURN SYS.MGW_BASIC_MSG_T;
```

---

**Note:** All commands in the examples must be run as a user granted MGW_ADMINISTRATOR_ROLE, except for the commands to create transformations.
You can create a transformation using `DBMS_TRANSFORM.CREATE_TRANSFORMATION`, as shown in Example 20–2.

**Example 20–2 Creating a Transformation**

```plsql
BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema         => 'mgwuser',
    name           => 'sample_adt_to_mgw_basic',
    from_schema    => 'mgwuser',
    from_type      => 'sampleadt',
    to_schema      => 'sys',
    to_type        => 'MGW_BASIC_MSG_T',
    transformation => 'mgwuser.trans_sampleadt_to_mgw_basic(user_data)');
END;
```

Once created, this transformation can be registered with MGW when creating a propagation job. Example 20–3 creates job `job_aq2mq`, for whom messages are propagated from Oracle® Database queue `mgwuser.srcq` to non-Oracle messaging system queue `destq@mqlink` using transformation `mgwuser.sample_adt_to_mgw_basic`.

**Example 20–3 Registering a Transformation**

```plsql
BEGIN
  DBMS_MGWADM.CREATE_JOB(
    job_name          => 'job_aq2mq',
    propagation_type  => DBMS_MGWADM.OUTBOUND_PROPAGATION,
    source            => 'mgwuser.srcq',
    destination       => 'destq.mqlink',
    transformation    => 'mgwuser.sample_adt_to_mgw_basic',
    exception_queue   => 'mgwuser.excq');
END;
```

An error that occurs while attempting a user-defined transformation is usually considered a message conversion exception, and the message is moved to the `exception queue` if it exists.

**Handling Logical Change Records**

MGW provides facilities to propagate Logical Change Records (LCRs). Routines are provided to help in creating transformations to handle the propagation of both row LCRs and DDL LCRs stored in queues with payload type `ANYDATA`. An LCR is propagated as an XML string stored in the appropriate message type.

Because Oracle Streams uses `ANYDATA` queues to store LCRs, an `ANYDATA` queue is the source for outbound propagation. The transformation must first convert the `ANYDATA` object containing an LCR into an `XMLType` object using the MGW routine `DBMS_MGWMSG.LCR_TO_XML`. If the `ANYDATA` object does not contain an LCR, then this routine raises an error. The XML document string of the LCR is then extracted from the

---

**See Also:** "DBMS_MGWADM", "DBMS_MGWMSG", and "DBMS_TRANSFORM" in *Oracle Database PL/SQL Packages and Types Reference*

**Note:** For LCR propagation, you must load the XDB package.
XMLType and placed in the appropriate MGW canonical type (SYS.MGW_BASIC_MSG_T or SYS.MGW_TIBRV_MSG_T).

Example 20–4 illustrates a simplified transformation used for LCR outbound propagation. The transformation converts an ANYDATA payload containing an LCR to a SYS.MGW_TIBRV_MSG_T object. The string representing the LCR as an XML document is put in a field named ORACLE_LCR.

Example 20–4  Outbound LCR Transformation

```
create or replace function any2tibrv(adata in anydata)
return SYS.MGW_TIBRV_MSG_T is
  v_xml   XMLType;
  v_text  varchar2(2000);
  v_tibrv sys.mgw_tibrv_msg_t;
BEGIN
  v_xml   := dbms_mgwmsg.lcr_to_xml(adata);
  -- assume the lcr is smaller than 2000 characters long.
  v_text  := v_xml.getStringVal();
  v_tibrv := SYS.MGW_TIBRV_MSG_T.CONSTRUCT;
  v_tibrv.add_string('ORACLE_LCR', 0, v_text);
  return v_tibrv;
END any2tibrv;
```

For LCR inbound propagation, an MGW canonical type (SYS.MGW_BASIC_MSG_T or SYS.MGW_TIBRV_MSG_T) is the transformation source type. A string in the format of an XML document representing an LCR must be contained in the canonical type. The transformation function must extract the string from the message, create an XMLType object from it, and convert it to an ANYDATA object containing an LCR with the MGW routine DBMS_MGWMSG.XML_TO_LCR. If the original XML document does not represent an LCR, then this routine raises an error.

Example 20–5 illustrates a simplified transformation used for LCR inbound propagation. The transformation converts a SYS.MGW_TIBRV_MSG_T object with a field containing an XML string representing an LCR to an ANYDATA object. The string representing the LCR as an XML document is taken from a field named ORACLE_LCR.

Example 20–5  Inbound LCR Transformation

```
create or replace function tibrv2any(tdata in sys.mgw_tibrv_msg_t)
return anydata is
  v_field   sys.mgw_tibrv_field_t;
  v_xml     XMLType;
  v_text    varchar2(2000);
  v_any     anydata;
BEGIN
  v_field := tdata.get_field_by_name('ORACLE_LCR');
  -- type checking
  v_text  := v_field.text_value;
  -- assume it is not null
  v_xml  := XMLType.createXML(v_text);
  v_any  := dbms_mgwmsg.xml_to_lcr(v_xml);
  return v_any;
END tibrv2any;
```
Message Conversion for WebSphere MQ

MGW converts between the MGW canonical types and the WebSphere MQ native message format. WebSphere MQ native messages consist of a fixed message header and a message body. The message body is treated as either a TEXT value or RAW (bytes) value. The canonical types supported for WebSphere MQ propagation are SYS.MGW_BASIC_MSG_T and RAW.

Figure 20–3 Message Conversion for WebSphere MQ Using MGW_BASIC_MSG_T

Figure 20–3 illustrates the message conversion performed by the MGW WebSphere MQ driver when using the canonical type SYS.MGW_BASIC_MSG_T. For outbound propagation, the driver maps the Oracle® Database message properties and canonical message to a WebSphere MQ message having a fixed header and a message body. For inbound propagation, the driver maps a native message to a set of Oracle® Database message properties and a canonical message. When the canonical type is RAW, the mappings are the same, except no canonical headers exist.

WebSphere MQ Message Header Mappings

When the MGW canonical type used in an outbound propagation job is RAW, no WebSphere MQ header information is set from the RAW message body. Similarly, for inbound propagation no WebSphere MQ header information is preserved in the RAW message body. MGW canonical type SYS.MGW_BASIC_MSG_T, however, has a header that can be used to specify WebSphere MQ header fields for outbound propagation, and preserve WebSphere MQ header fields for inbound propagation.

This section describes the message properties supported for the WebSphere MQ messaging system when using SYS.MGW_BASIC_MSG_T as the canonical type. Table 20–1 defines the MGW {name, value} pairs used to describe the WebSphere MQ header properties. The first column refers to valid string values for the SYS.MGW_NAME_VALUE_T.NAME field in the SYS.MGW_BASIC_MSG_T header. The second
column refers to the SYS.MGW_NAME_VALUE_T.TYPE value corresponding to the name. (Refer to “Notes on Table 20–1” on page 20-8 for explanations of the numbers in parentheses.)

See Also: "DBMS_MGWMSG" in Oracle Database PL/SQL Packages and Types Reference

For inbound propagation, the WebSphere MQ driver generates [name,value] pairs based on the source message header and stores them in the header part of the canonical message of the SYS.MGW_BASIC_MSG_T type. For outbound propagation, the WebSphere MQ driver sets the message header and enqueue options from {name,value} pairs for these properties stored in the header part of the SYS.MGW_BASIC_MSG_T canonical message.

### Table 20–1  MGW Names for WebSphere MQ Header Values

<table>
<thead>
<tr>
<th>MGW Name</th>
<th>MGW Type</th>
<th>WebSphere MQ Property Name</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGW_MQ_accountingToken</td>
<td>RAW_VALUE (size 32)</td>
<td>accountingToken</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_applicationIdData</td>
<td>TEXT_VALUE (size 32)</td>
<td>applicationIdData</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_applicationOriginData</td>
<td>TEXT_VALUE (size 4)</td>
<td>applicationOriginData</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_backoutCount</td>
<td>INTEGER_VALUE</td>
<td>backoutCount</td>
<td>Inbound</td>
</tr>
<tr>
<td>MGW_MQ_characterSet</td>
<td>INTEGER_VALUE</td>
<td>characterSet</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_correlationId</td>
<td>RAW_VALUE (size 24)</td>
<td>correlationId</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_encoding</td>
<td>INTEGER_VALUE</td>
<td>encoding</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_expiry</td>
<td>INTEGER_VALUE</td>
<td>expiry</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_feedback</td>
<td>INTEGER_VALUE</td>
<td>feedback</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_format</td>
<td>TEXT_VALUE (size 8)</td>
<td>format</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_groupId</td>
<td>RAW_VALUE (size 24)</td>
<td>groupId</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_messageFlags</td>
<td>INTEGER_VALUE</td>
<td>messageFlags</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_messageId</td>
<td>RAW_VALUE (size 24)</td>
<td>messageId</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_messageSequenceNumber</td>
<td>INTEGER_VALUE</td>
<td>messageSequenceNumber</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_messageType</td>
<td>INTEGER_VALUE</td>
<td>messageType</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_offset</td>
<td>INTEGER_VALUE</td>
<td>offset</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_originalLength</td>
<td>INTEGER_VALUE</td>
<td>originalLength</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_persistence</td>
<td>INTEGER_VALUE</td>
<td>persistence</td>
<td>Inbound</td>
</tr>
<tr>
<td>MGW_MQ_priority</td>
<td>INTEGER_VALUE</td>
<td>priority</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_putApplicationName</td>
<td>TEXT_VALUE (size 28)</td>
<td>putApplicationName</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_putApplicationType</td>
<td>INTEGER_VALUE</td>
<td>putApplicationType</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_putDateTime</td>
<td>DATE_VALUE</td>
<td>putDateTime</td>
<td>Inbound</td>
</tr>
<tr>
<td>MGW_MQ_putMessageOptions</td>
<td>INTEGER_VALUE</td>
<td>putMessageOptions</td>
<td>Outbound (1) (2)</td>
</tr>
<tr>
<td>MGW_MQ_replyToQueueManagerName</td>
<td>TEXT_VALUE (size 48)</td>
<td>replyToQueueManagerName</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_replyToQueueName</td>
<td>TEXT_VALUE (size 48)</td>
<td>replyToQueueName</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>MGW_MQ_report</td>
<td>INTEGER_VALUE</td>
<td>report</td>
<td>Outbound (1), Inbound</td>
</tr>
<tr>
<td>MGW_MQ_userId</td>
<td>TEXT_VALUE (size 12)</td>
<td>userId</td>
<td>Outbound, Inbound</td>
</tr>
</tbody>
</table>
Notes on Table 20–1

1. This use is subject to WebSphere MQ restrictions. For example, if `MGW_MQ_accountingToken` is set for an outgoing message, then WebSphere MQ overrides its value unless `MGW_MQ_putMessageOptions` is set to the WebSphere MQ constant `MQPMD_SET_ALL_CONTEXT`.

2. `MGW_MQ_putMessageOptions` is used as the `putMessageOptions` argument to the WebSphere MQ Base Java `Queue.put()` method. It is not part of the WebSphere MQ header information and is therefore not an actual message property.

   The value for the `openOptions` argument of the WebSphere MQ Base Java `MQQueueManager.accessQueue` method is specified when the WebSphere MQ queue is registered using the `DBMS_MGWADMREGISTER_FOREIGN_QUEUE` call. Dependencies can exist between the two. For instance, for `MGW_MQ_putMessageOptions` to include `MQPMD_SET_ALL_CONTEXT`, the `MQ_openMessageOptions` queue option must include `MQOO_SET_CONTEXT`.

   The MGW agent adds the value `MQPMO_SYNCPOINT` to any value that you can specify.

MGW sets default values for two WebSphere MQ message header fields: `messageType` defaults to `MQMT_DATAGRAM` and `putMessageOptions` defaults to `MQPMO_SYNCPOINT`.

MGW provides two default mappings between Oracle® Database message properties and WebSphere MQ header fields.

One maps the Oracle® Database message property `expiration`, representing the time-to-live of the message at the time the message becomes available in the queue, to the WebSphere MQ header field `expiry`, representing the time-to-live of the message. For outbound propagation, the value used for `expiry` is determined by subtracting the time the message was available in the queue from the `expiration`, converted to tenths of a second. Oracle® Database value `NEVER` is mapped to `MQEI_UNLIMITED`. For inbound propagation, the value of `expiration` is simply `expiry` converted to seconds. WebSphere MQ value `MQEI_UNLIMITED` is mapped to `NEVER`.

The other default maps Oracle® Database message property `priority` with the WebSphere MQ header field `priority`. It is described in Table 20–2.

### Table 20–2 Default Priority Mappings for Propagation

<table>
<thead>
<tr>
<th>Propagation Type</th>
<th>Message System</th>
<th>Priority Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outbound</td>
<td>Oracle® Database</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>Outbound</td>
<td>WebSphere MQ</td>
<td>9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Inbound</td>
<td>Oracle® Database</td>
<td>9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Inbound</td>
<td>WebSphere MQ</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

**Note:** For outbound propagation, Oracle® Database priority values less than 0 are mapped to WebSphere MQ priority 9, and Oracle® Database priority values greater than 9 are mapped to WebSphere MQ priority 0.
WebSphere MQ Outbound Propagation

If no message transformation is provided for outbound propagation, then the Oracle® Database source queue payload type must be either SYS.MGW_BASIC_MSG_T or RAW. If a message transformation is specified, then the target ADT of the transformation must be SYS.MGW_BASIC_MSG_T, but the source ADT can be any ADT supported by Oracle® Database.

If the Oracle® Database queue payload is RAW, then the resulting WebSphere MQ message has the message body set to the value of the RAW bytes and, by default, the format field set to the value "MGW_Byte".

If the Oracle® Database queue payload or transformation target ADT is SYS.MGW_BASIC_MSG_T, then the message is mapped to a WebSphere MQ native message as follows:

- The WebSphere MQ fixed header fields are based on the internal Oracle® Database message properties and the SYS.MGW_BASIC_MSG_T.header attribute of the canonical message, as described in "WebSphere MQ Message Header Mappings" on page 20-6.
- If the canonical message has a TEXT body, then the WebSphere MQ format header field is set to MQFMT_STRING unless overridden by the header property MGW_MQ_format. The message body is treated as text.
- If the canonical message has a RAW body, then the WebSphere MQ format header field is set to "MGW_Byte" unless overridden by the header property MGW_MQ_format. The message body is treated as raw bytes.
- If the canonical message has both a TEXT and RAW body, then message conversion fails.
- If the canonical message has neither a TEXT nor RAW body, then no message body is set, and the WebSphere MQ format header field is MQFMT_NONE.
- If the canonical message has a TEXT body with both small and large values set (SYS.MGW_BASIC_MSG_T.TEXT_BODY.small_value and SYS.MGW_BASIC_MSG_T.TEXT_BODY.large_value not empty), then message conversion fails.
- If the canonical message has a RAW body with both small and large values set (SYS.MGW_BASIC_MSG_T.RAW_BODY.small_value and SYS.MGW_BASIC_MSG_T.RAW_BODY.large_value not empty), then message conversion fails.

If the job option PreserveMessageID is specified with a value of TRUE, then the correlationId field of the WebSphere message header will be set to the AQ source message identifier. The correlationId value will be a 24-byte value of the form "AQMMSGID:"+AQ_msgid where AQ_msgid represents the 16-byte Streams AQ message identifier.

WebSphere MQ Inbound Propagation

If no message transformation is provided for inbound propagation, then the Oracle® Database destination queue payload type must be either SYS.MGW_BASIC_MSG_T or RAW. If a message transformation is specified, then the source ADT of the transformation must be SYS.MGW_BASIC_MSG_T, but the destination ADT can be any ADT supported by Oracle® Database.

If the Oracle® Database queue payload is RAW and the incoming WebSphere MQ message has a format of MQFMT_STRING, then message conversion fails. Otherwise the message body is considered as raw bytes and enqueued directly to the destination.
queue. If the number of bytes is greater than 32KB, then message conversion fails. The actual limit is 32512 bytes rather than 32767 bytes.

If the Oracle® Database queue payload or transformation source ADT is SYS.MGW_BASIC_MSG_T, then the WebSphere MQ message is mapped to a SYS.MGW_BASIC_MSG_T message as follows:

- Specific WebSphere MQ header fields are mapped to Oracle® Database message properties as previously described.
- The SYS.MGW_BASIC_MSG_T.header attribute of the canonical message is set to \{name, value\} pairs based on the WebSphere MQ header fields, as described in Table 20–1. These values preserve the original content of the WebSphere MQ message header.
- If the WebSphere MQ format header field is MQFMT_STRING, then the WebSphere MQ message body is treated as text, and its value is mapped to SYS.MGW_BASIC_MSG_T.text_body. For any other format value, the message body is treated as raw bytes, and its value is mapped to SYS.MGW_BASIC_MSG_T.raw_body.

See Also: “WebSphere MQ Message Header Mappings” on page 20-6

Message Conversion for TIB/Rendezvous

MGW regards a TIB/Rendezvous message as a set of fields and supplementary information. Figure 20–4 shows how messages are converted between MGW and TIB/Rendezvous.

When a message conversion failure occurs, messages are moved to an exception queue (if one has been provided), so that MGW can continue propagation of the remaining messages in the source queue. In inbound propagation from TIB/Rendezvous, an exception queue is a registered subject.

All TIB/Rendezvous wire format datatypes for TIB/Rendezvous fields are supported, except for the datatypes with unsigned integers and the nested message type. User-defined custom datatypes are not supported in this release. If a message contains data of the unsupported datatypes, then a message conversion failure occurs when the message is processed. A message conversion failure results in moving the failed message from the source queue to the exception queue, if an exception queue is provided.

Table 20–3 shows the datatype mapping used when MGW converts between a native TIB/Rendezvous message and the canonical ADT. For each supported
TIB/Rendezvous wire format type, it shows the Oracle type used to store the data and the `DBMS_MGWMSG` constant that represents that type.

**Table 20-3 TIB/Rendezvous Datatype Mapping**

<table>
<thead>
<tr>
<th>TIB/Rendezvous Wire Format</th>
<th>Oracle Type</th>
<th>ADT Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool</td>
<td>NUMBER</td>
<td>TIBRVMSG_BOOL</td>
</tr>
<tr>
<td>F32</td>
<td>NUMBER</td>
<td>TIBRVMSG_F32</td>
</tr>
<tr>
<td>F64</td>
<td>NUMBER</td>
<td>TIBRVMSG_F64</td>
</tr>
<tr>
<td>I8</td>
<td>NUMBER</td>
<td>TIBRVMSG_I8</td>
</tr>
<tr>
<td>I16</td>
<td>NUMBER</td>
<td>TIBRVMSG_I16</td>
</tr>
<tr>
<td>I32</td>
<td>NUMBER</td>
<td>TIBRVMSG_I32</td>
</tr>
<tr>
<td>I64</td>
<td>NUMBER</td>
<td>TIBRVMSG_I64</td>
</tr>
<tr>
<td>U8</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U16</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U32</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U64</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>IPADDR32</td>
<td>VARCHAR2</td>
<td>TIBRVMSG_IPADDR32</td>
</tr>
<tr>
<td>IPPORT16</td>
<td>NUMBER</td>
<td>TIBRVMSG_IPPORT16</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATE</td>
<td>TIBRVMSG_DATETIME</td>
</tr>
<tr>
<td>F32ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_F32ARRAY</td>
</tr>
<tr>
<td>F64ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_F64ARRAY</td>
</tr>
<tr>
<td>I8ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_I8ARRAY</td>
</tr>
<tr>
<td>I16ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_I16ARRAY</td>
</tr>
<tr>
<td>I32ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_I32ARRAY</td>
</tr>
<tr>
<td>I64ARRAY</td>
<td>SYS.MGW_NUMBER_ARRAY_T</td>
<td>TIBRVMSG_I64ARRAY</td>
</tr>
<tr>
<td>U8ARRAY</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U16ARRAY</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U32ARRAY</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>U64ARRAY</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>MSG</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td>OPAQUE</td>
<td>RAW or BLOB</td>
<td>TIBRVMSG_OPAQUE</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR2 or CLOB</td>
<td>TIBRVMSG_STRING</td>
</tr>
<tr>
<td>XML</td>
<td>RAW or BLOB</td>
<td>TIBRVMSG_XML</td>
</tr>
</tbody>
</table>

For propagation between Oracle® Database and TIB/Rendezvous, MGW provides direct support for the Oracle® Database payload types RAW and `SYS.MGW_TIBRV_MSG_T`. To support any other Oracle® Database payload type, you must supply a transformation.
AQ Message Property Mapping for TIB/Rendezvous

This section describes the mapping between Oracle Streams AQ message properties and TIB/Rendezvous fields. This mapping is used to preserve Streams AQ message properties during outbound propagation, and set Streams AQ message properties during inbound propagation.

Table 20–4 describes the Streams AQ message properties supported using TIB/Rendezvous fields. The first column indicates the DBMS_AQ.MESSAGE_PROPERTIES_T field for the Streams AQ message property. The second and third columns indicate the name and datatype used for the TIB/Rendezvous field. The last column indicates if the message property is supported for inbound and outbound propagation.

<table>
<thead>
<tr>
<th>Oracle® Database Message Property</th>
<th>MGW Name</th>
<th>TIB/Rendezvous Wire Format Datatype</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>MGW_AQ_priority</td>
<td>TibrvMsg.I32</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>expiration</td>
<td>MGW_AQ_expiration</td>
<td>TibrvMsg.I32</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>delay</td>
<td>MGW_AQ_delay</td>
<td>TibrvMsg.I32</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>correlation</td>
<td>MGW_AQ_correlation</td>
<td>TibrvMsg.STRING</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>exception_queue</td>
<td>MGW_AQ_exception_queue</td>
<td>TibrvMsg.STRING</td>
<td>Outbound, Inbound</td>
</tr>
<tr>
<td>enqueue_time</td>
<td>MGW_AQ_enqueue_time</td>
<td>TibrvMsg.DATETIME</td>
<td>Outbound</td>
</tr>
<tr>
<td>original_msgid</td>
<td>MGW_AQ_original_msgid</td>
<td>TibrvMsg.OPAQUE</td>
<td>Outbound</td>
</tr>
<tr>
<td>msgid (1)</td>
<td>MGW_AQ_messageID</td>
<td>TibrvMsg.OPAQUE</td>
<td>Outbound</td>
</tr>
</tbody>
</table>

Notes on Table 20–4:
1. The msgid Streams AQ property represents the Streams AQ message identifier, rather than a particular field of the DBMS_AQ.MESSAGE_PROPERTIES_T record.

TIB/Rendezvous Outbound Propagation

If no propagation transformation is provided for outbound propagation, then the Oracle® Database source queue payload type must be either SYS.MGW_TIBRV_MSG_T or RAW. If a propagation transformation is specified, then the target ADT of the transformation must be SYS.MGW_TIBRV_MSG_T, but the source ADT can be any ADT supported by Oracle® Database.

If the Oracle® Database queue payload or transformation target ADT is SYS.MGW_TIBRV_MSG_T, then:

- Every field in the source message is converted to a TIB/Rendezvous message field of the resulting TIB/Rendezvous message.
- If the reply_subject attribute is not NULL, then the reply subject supplementary information is set.
- The send_subject field is ignored.

If the Oracle® Database queue payload is RAW, then:

- The resulting message contains a field named MGW_RAW_MSG with value TibrvMsg.OPAQUE. The field ID is set to 0.
If the job option AQ_MsgProperties is specified with a value of TRUE, then the MGW agent generates fields to preserve the Streams AQ message properties in the TIB/Rendezvous message according to Table 20–4.

If the PreserveMessageID job option is specified with a value of TRUE, then the Streams AQ message identifier (msgid) is preserved in the TIB/Rendezvous message according to Table 20–4.

TIB/Rendezvous Inbound Propagation

If no propagation transformation is provided for inbound propagation, then the Oracle® Database destination queue payload type must be either RAW or SYS.MGW_TIBRV_MSG_T. If a propagation transformation is specified, then the target ADT of the transformation can be any ADT supported by Oracle® Database, but the source ADT of the transformation must be SYS.MGW_TIBRV_MSG_T.

If the Oracle® Database queue payload or transformation source ADT is SYS.MGW_TIBRV_MSG_T, then:

■ Every field in the source TIB/Rendezvous message is converted to a field of the resulting message of the SYS.MGW_TIBRV_MSG_T type.

■ The MGW agent extracts the send subject name from the source TIB/Rendezvous message and sets the send_subject attribute in SYS.MGW_TIBRV_MSG_T. The send subject name is usually the same as the subject name of the registered propagation source queue, but it might be different when wildcards are used.

■ The MGW agent extracts the reply subject name from the source TIB/Rendezvous message, if it exists, and sets the reply_subject attribute in SYS.MGW_TIBRV_MSG_T.

■ If the source TIB/Rendezvous message contains more than three large text fields (greater than 4000 bytes of text) or more than three large bytes fields (greater than 2000 bytes), then message conversion fails.

If the Oracle® Database queue payload is RAW, then:

■ The Oracle® Database message payload is the field data if the source TIB/Rendezvous message has a field named MGW_RAW_MSG of type TibrvMsg.OPAQUE or TibrvMsg.XML. The field name and ID are ignored. If no such field exists or has an unexpected type, then a message conversion failure occurs.

■ A message conversion failure occurs if the RAW data size is greater than 32KB. This is due to a restriction on the data size allowed for a bind variable. Also, the actual limit is 32512 rather than 32767.

If the job option AQ_MsgProperties is specified with a value of TRUE, then the MGW agent searches for fields in the original TIB/Rendezvous messages with reserved field names. Table 20–4 shows the field name strings and the corresponding values used in the TIB/Rendezvous message.

If such fields exist, then the MGW agent uses the field value to set the corresponding Oracle® Database message properties, instead of using the default values. If there is more than one such field with the same name, then only the first one is used. Such fields are removed from the resulting payload only if the Oracle® Database queue payload is RAW. If a field with the reserved name does not have the expected datatype, then it causes a message conversion failure.

See Also: "DBMS_MGWMSG" in Oracle Database PL/SQL Packages and Types Reference for the value datatypes
JMS Messages

MGW propagates only JMS messages between Oracle JMS and non-Oracle JMS systems, without changing the message content. Figure 20–5 shows JMS message propagation.

MGW supports only the standard JMS message types. It does not support:

- JMS provider extensions, because any such extensions would not be recognized by the destination JMS system. An attempt to propagate any such non-JMS message results in an error.
- User transformations for JMS propagation.
- Propagation of Logical Change Records (LCRs).

**Figure 20–5  JMS Message Propagation**

For the purposes of this discussion, a JMS message is a Java object of a class that implements one of the five JMS message interfaces. Table 20–5 shows the JMS message interfaces and the corresponding Oracle JMS ADTs. The table also shows the interface, `javax.jms.Message`, which can be any one of the five specific types, and the corresponding generic Oracle JMS type `SYS.AQ$_JMS_MESSAGE`.

<table>
<thead>
<tr>
<th>JMS Message</th>
<th>ADT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.jms.TextMessage</code></td>
<td><code>SYS.AQ$_JMS_TEXT_MESSAGE</code></td>
</tr>
<tr>
<td><code>javax.jms.BytesMessage</code></td>
<td><code>SYS.AQ$_JMS_BYTES_MESSAGE</code></td>
</tr>
<tr>
<td><code>javax.jms.MapMessage</code></td>
<td><code>SYS.AQ$_JMS_MAP_MESSAGE</code></td>
</tr>
<tr>
<td><code>javax.jms.StreamMessage</code></td>
<td><code>SYS.AQ$_JMS_STREAM_MESSAGE</code></td>
</tr>
<tr>
<td><code>javax.jms.ObjectMessage</code></td>
<td><code>SYS.AQ$_JMS_OBJECT_MESSAGE</code></td>
</tr>
<tr>
<td><code>javax.jms.Message</code></td>
<td><code>SYS.AQ$_JMS_MESSAGE</code></td>
</tr>
</tbody>
</table>

When a propagation job is activated, the MGW agent checks the Oracle® Database payload type for the propagation source or destination. If the type is one of those listed in Table 20–5 or ANYDATA, then message propagation is attempted. Otherwise an exception is logged and propagation is not attempted.

The MGW agent may add a JMS String property named `OracleMGW_OriginalMessageID` to the JMS message sent to the destination queue in order to
preserve the original message identifier of the source message. This property is added if the PreserveMessageID job option is specified with a value of TRUE. It will also be added for any message moved to an exception queue upon a message conversion failure.

**JMS Outbound Propagation**

When dequeuing a message from an Oracle® Database queue, Oracle JMS converts instances of the ADTs shown in Table 20–5 into JMS messages. In addition it can convert instances of ANYDATA into JMS messages, depending on the content.

A queue with payload type ANYDATA can hold messages that do not map to a JMS message. MGW fails to dequeue such a message. An error is logged and propagation of messages from that queue does not continue until the message is removed.

**JMS Inbound Propagation**

Every message successfully dequeued using WebSphere MQ JMS is a JMS message. No message conversion is necessary prior to enqueuing using Oracle JMS. However, if the payload ADT of the propagation destination does not accept the type of the inbound message, then an exception is logged and an attempt is made to place the message in an exception queue. An example of such type mismatches is a JMS TextMessage and a queue payload type SYS.AQ$_JMS_BYTES_MESSAGE.
This chapter discusses means of monitoring the Oracle Messaging Gateway (MGW) agent, abnormal situations you may experience, several sources of information about Messaging Gateway errors and exceptions, and suggested remedies.

This chapter contains these topics:

- Oracle Messaging Gateway Log Files
- Monitoring the Oracle Messaging Gateway Agent Status
- Monitoring Oracle Messaging Gateway Propagation
- Oracle Messaging Gateway Agent Error Messages

Oracle Messaging Gateway Log Files

Messaging Gateway agent status, history, and errors are recorded in Messaging Gateway log files. A different log file is created each time the Messaging Gateway agent is started. You should monitor the log file because any errors, configuration information read at startup time, or dynamic configuration information is written to the log.

The format of the log file name for the default agent is:

```
oramgw-hostname-timestamp-processid.log
```

The format of the log file name for a named agent is:

```
oramgw-AGENTNAME-hostname-timestamp-processid.log
```

By default the Messaging Gateway log file is in `ORACLE_HOME/mgw/log`. This location can overridden by the parameter `log_directory` in the Messaging Gateway initialization file used by the agent, usually `mgw.ora`.

This section contains these topics:

- Sample Oracle Messaging Gateway Log File
- Interpreting Exception Messages in an Oracle Messaging Gateway Log File

Sample Oracle Messaging Gateway Log File

The following sample log file shows the Messaging Gateway agent starting. The sample log file shows that a messaging link, a registered foreign queue, a propagation job, and a schedule associated with the job have been added. The log file shows that the propagation job has been activated. The last line indicates that the Messaging Gateway is up and running and ready to propagate messages.
Example 21–1  Sample Messaging Gateway Log File

>>2007-01-16 15:04:49 MGW C-Bootstrap 0 LOG process-id=11080
Bootstrap program starting
>>2007-01-16 15:04:50 MGW C-Bootstrap 0 LOG process-id=11080
JVM created -- heapsize = 64
>>2007-01-16 15:04:53 MGW Engine 0 200 main
MGW Agent version: 11.1.0.0
>>2007-01-16 15:04:53 MGW AdminMgr 0 LOG main
Connecting to database using connect string = jdbc:oracle:oci:@INST1
>>2007-01-16 15:05:00 MGW Engine 0 200 main
MGW Component version: 11.1.0.3.0
>>2007-01-16 15:05:01 MGW Engine 0 200 main
MGW agent name: DEFAULT_AGENT, MGW job instance id: 273006EC6ED255F1E040578C6D021A8C, MGW database instance: 1
>>2007-01-16 15:05:09 MGW Engine 0 1 main
Agent is initializing.
>>2007-01-16 15:05:09 MGW Engine 0 23 main
The number of worker threads is set to 1.
>>2007-01-16 15:05:09 MGW Engine 0 22 main
The default polling interval is set to 5000ms.
>>2007-01-16 15:05:09 MGW MQD 0 LOG main
Creating MQSeries messaging link:
  link : MQLINK
  link type : Base Java interface
  queue manager : my.queue.manager
  channel : channel1
  host : my.machine
  port : 1414
  user :
  ccdt url :
  ssl cipherSuite :
  connections : 1
  inbound logQ : logq1
  outbound logQ : logq2
>>2007-01-16 15:05:09 MGW Engine 0 4 main
Link MQLINK has been added.
>>2007-01-16 15:05:09 MGW Engine 0 7 main
Queue DESTQ@MQLINK has been registered; provider queue: MGWUSER.MYWQUEUE.
>>2007-01-16 15:05:09 MGW Engine 0 9 main
Propagation Schedule JOB_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been added.
>>2007-01-16 15:05:09 MGW AQN 0 LOG main
Creating AQ messaging link:
  link : oracleMgwAq
  link type : native
  database : INST1
  user : MGW/AGENT
  connection type : JDBC OCI
  connections : 1
  inbound logQ : SYS.MGW_RECV_LOG
  outbound logQ : SYS.MGW_SEND_LOG
>>2007-01-16 15:05:10 MGW Engine 0 19 main
MGW propagation job JOB_AQ2MQ has been activated.
>>2007-01-16 15:05:10 MGW Engine 0 14 main
MGW propagation job JOB_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been added.
>>2007-01-16 15:05:11 MGW Engine 0 2 main
Agent is up and running.
Interpreting Exception Messages in an Oracle Messaging Gateway Log File

Exception messages logged to the Messaging Gateway log file may include one or more linked exceptions, identified by [Linked-exception] in the log file. These are often the most useful means of determining the cause of a problem. For instance, a linked exception could be a java.sql.SQLException, possibly including an Oracle error message, a PL/SQL stack trace, or both.

The following example shows entries from a Messaging Gateway log file when an invalid value (bad_service_name) was specified for the database parameter of DBMS_MGWADM.CREATE_AGENT or DBMS_MGWADM.ALTER_AGENT. This resulted in the Messaging Gateway agent being unable to establish database connections.

Example 21–2 Sample Exception Message

```
>>2003-07-22 15:27:26  MGW AdminMgr  0  LOG  main
Connecting to database using connect string = jdbc:oracle:oci8:@BAD_SERVICE_NAME
>>2003-07-22 15:27:29  MGW Engine  0  EXCEPTION  main
[ ...Java stack trace here...]
[Linked-exception]
java.sql.SQLException: ORA-12154: TNS:could not resolve the connect identifier specified
[ ...Java stack trace here...]
>>2003-07-22 15:27:29  MGW Engine  0  25  main
Agent is shutting down.
```

Monitoring the Oracle Messaging Gateway Agent Status

This section contains these topics:

- **MGW_GATEWAY View**
- **Oracle Messaging Gateway Irrecoverable Error Messages**
- **Other Oracle Messaging Gateway Error Conditions**

**MGW_GATEWAY View**

The MGW_GATEWAY view monitors the progress of the Messaging Gateway agent. Among the fields that can be used to monitor the agent are:

- **AGENT_NAME**
- **AGENT_INSTANCE**
- **AGENT_PING**
- **AGENT_STATUS**
- **LAST_ERROR_MSG**
- **SERVICE**

The AGENT_STATUS field shows the status of the agent. This column has the following possible values:

- **NOT_STARTED**
  Indicates that the agent is neither running nor scheduled to be run.

- **START_SCHEDULED**
  Indicates that the agent job is waiting to be run by the job scheduler.
STARTING
Indicates that the agent is in the process of starting.

INITIALIZING
Indicates that the agent has started and is reading configuration data.

RUNNING
Indicates that the agent is ready to propagate any available messages or process dynamic configuration changes.

SHUTTING_DOWN
Indicates that the agent is in the process of shutting down.

BROKEN
Indicates that, while attempting to start an agent process, Messaging Gateway has detected another agent already running. This situation should never occur under normal usage.

Querying the AGENT_PING field pings the Messaging Gateway agent. Its value is either REACHABLE or UNREACHABLE. An agent with status of RUNNING should almost always be REACHABLE.

The columns LAST_ERROR_MSG, LAST_ERROR_DATE, and LAST_ERROR_TIME give valuable information if an error in starting or running the Messaging Gateway agent occurs. AGENT_INSTANCE indicates the Oracle Database instance on which the Messaging Gateway instance was started.

See Also: "DBMS_MGWADM" in Oracle Database PL/SQL Packages and Types Reference for more information on the MGW_GATEWAY view

Oracle Messaging Gateway Irrecoverable Error Messages
A status of NOT_STARTED in the AGENT_STATUS field of the MGW_GATEWAY view indicates that the Messaging Gateway agent is not running. If the AGENT_STATUS is NOT_STARTED and the LAST_ERROR_MSG field is not NULL, then the Messaging Gateway agent has encountered an irrecoverable error while starting or running. Check if a Messaging Gateway log file has been generated and whether it indicates any errors. If a log file is not present, then the Messaging Gateway agent process was probably not started.

This section describes the causes and solutions for some error messages that may appear in the LAST_ERROR_MSG field of the MGW_GATEWAY view. Unless indicated otherwise, the Messaging Gateway agent will not attempt to restart itself when one of these errors occurs.

ORA-01089: Immediate shutdown in progress - no operations are permitted
The Messaging Gateway agent has shut down because the SHUTDOWN_IMMEDIATE command was used to shut down a running Oracle Database instance on which the agent was running. The agent will restart itself on the next available database instance on which it is set up to run.

ORA-06520: PL/SQL: Error loading external library
The Messaging Gateway agent process was unable to start because the shared library was not loaded. This may be because the Java shared library was not in the library path. Verify that the library path in listener.ora has been set correctly.
**ORA-28575: Unable to open RPC connection to external procedure agent**

The Messaging Gateway agent was unable to start. It will attempt to start again automatically.

Possible causes include:

- The listener is not running. If you have modified listener.ora, then you must stop and restart the listener before the changes will take effect.

- Values in tnsnames.ora, listener.ora, or both are not correct.
  
  In particular, tnsnames.ora must have a net service name entry of MGW_AGENT. This entry is not needed for Messaging Gateway on Windows. The SID value specified for CONNECT_DATA of the MGW_AGENT net service name in tnsnames.ora must match the SID_NAME value of the SID_DESC entry in listener.ora. If the MGW_AGENT net service name is set up for an Inter-process Communication (IPC) connection, then the KEY values for ADDRESS in tnsnames.ora and listener.ora must match. If the names.default_domain parameter for sqlnet.ora has been used to set a default domain, then that domain must be appended to the MGW_AGENT net service name in tnsnames.ora.

**ORA-28576: Lost RPC connection to external procedure agent**

The Messaging Gateway agent process ended prematurely. This may be because the process was stopped by an outside entity or because an internal error caused a malfunction. The agent will attempt to start again automatically. Check the Messaging Gateway log file to determine if further information is available. If the problem persists, then contact Oracle Support Services for assistance.

**ORA-32830: Result code -2 returned by Messaging Gateway agent**

An error occurred when the Messaging Gateway agent tried to read its initialization file, usually mgw.ora. Verify that the file is readable.

**ORA-32830: Result code -3 returned by Messaging Gateway agent**

An error occurred creating the Messaging Gateway log file. Verify that the log directory can be written to. The default location is ORACLE_HOME/mgw/log.

**ORA-32830: Result code -8 returned by Messaging Gateway agent**

An error occurred starting the Java Virtual Machine (JVM). Verify that:

- You are using the correct Java version
- Your operating system version and patch level are sufficient for the JDK version
- You are using a reasonable value for the JVM heap size
  
  The heap size is specified by the max_memory parameter of DBMS_MGWADM.ALTER_AGENT

- On Windows platforms, verify the MGW_PRE_PATH set in mgw.ora contains the path to the correct JVM library (jvm.dll).

**ORA-32830: Result code -12 returned by Messaging Gateway agent**

An error occurred writing to the Messaging Gateway log file. Check the free disk space or any other issues that might result in file I/O problems.
ORA-32830: Result code -17 returned by Messaging Gateway agent
The JVM was successfully created but an error occurred trying to call the MGW Java agent program. Verify that the CLASSPATH set in mgw.ora is correct.

ORA-32830: Result code -19 returned by Messaging Gateway agent
The Messaging Gateway agent was configured to use a particular initialization file but that file does not exist. The INITFILE field of the MGW_GATEWAY view shows the full pathname of the file specified by the administrator. Either create that initialization file, or use DBMS_MGWADM.ALTER_AGENT to set INITFILE to another file or NULL to use the default initialization file.

ORA-32830: Result code -100 returned by Messaging Gateway agent
The Messaging Gateway agent JVM encountered a runtime exception or error on startup before it could write to the log file.

ORA-32830: Result code -101 returned by Messaging Gateway agent
An irrecoverable error caused the Messaging Gateway agent to shut down. Check the Messaging Gateway log file for further information. Verify that the values specified in mgw.ora are correct. Incorrect values can cause the Messaging Gateway agent to terminate due to unusual error conditions.

ORA-32830: Result code -102 returned by Messaging Gateway agent
The Messaging Gateway agent shut down because the version of file ORACLE_HOME/mgw/jlib/mgw.jar does not match the version of the Messaging Gateway PL/SQL packages. Verify that all Messaging Gateway components are from the same release.

ORA-32830: Result code -103 returned by Messaging Gateway agent
The Messaging Gateway agent shut down because the database instance on which it was running was shutting down. The agent should restart automatically, either on another instance if set up to do so, or when the instance that shut down is restarted.

ORA-32830: Result code -104 returned by Messaging Gateway agent
See previous error.

ORA-32830: Result code -105 returned by Messaging Gateway agent
The Messaging Gateway agent detected that it was running when it should not be. This should not happen. If it does, AGENT_STATUS will be BROKEN and the agent will shut down automatically. If you encounter this error:

- Terminate any Messaging Gateway agent process that may still be running. The process is usually named extprocmgwextproc.
- Run DBMS_MGWADM.CLEANUP_GATEWAY (DBMS_MGWADM.CLEAN_STARTUP_STATE).
- Start the Messaging Gateway agent using DBMS_MGWADM.STARTUP.

ORA-32830: Result code -106 returned by Messaging Gateway agent
See previous error.

See Also: "DBMS-MGWADM" in Oracle Database PL/SQL Packages and Types Reference
Other Oracle Messaging Gateway Error Conditions

This section discusses possible causes for $AGENT\_STATUS$ remaining $START\_SCHEDULED$ in $MGW\_GATEWAY$ view for an extended period.

Database Service Not Started
Messaging Gateway uses an Oracle Scheduler job to start the Messaging Gateway agent. Oracle Scheduler allows you to specify a database service under which a job should be run (service affinity). Messaging Gateway allows an administrator to configure the Messaging Gateway agent with a database service that will be used to configure the Scheduler job class associated with that agent.

When you shutdown a database Oracle stops all services to that database. You may need to manually restart the services when you start the database. If a Scheduler job is associated with a service then the job will not run until the service is started. If $AGENT\_STATUS$ for a Messaging Gateway agent remains $START\_SCHEDULED$ for an extended period that might indicate that the database service is disabled or no database instances associated with the service are running. Use the $MGW\_GATEWAY$ view, Oracle Scheduler views, and service views to determine how the agent was configured and the current state of the Scheduler job and database service.

Too Few Job Queue Processes
Messaging Gateway uses Oracle Scheduler to start the Messaging Gateway external process. When $AGENT\_STATUS$ is $START\_SCHEDULED$, the Messaging Gateway agent Scheduler job is waiting to be run by the Scheduler. The Messaging Gateway job will not run until there is an available job process. Messaging Gateway holds its Scheduler job process for the lifetime of the Messaging Gateway agent session. If multiple Messaging Gateway agents have been started, each agent uses its own Scheduler job and require its own job process.

If the value of the database initialization parameter $JOB\_QUEUE\_PROCESSES$ is zero, then that parameter does not influence the number of Oracle Scheduler jobs that can concurrently run. However, if the value is non-zero, it effectively becomes the maximum number of Scheduler jobs and job queue jobs than can concurrently run.

If Messaging Gateway status remains $START\_SCHEDULED$ for an extended period of time, then it may indicate that the database has been started with a non-zero value for $JOB\_QUEUE\_PROCESSES$ and that all jobs processes are busy. If the value is non-zero, verify that the database instance has been started with enough job queue processes so that one is available for each Messaging Gateway agent.

Scheduler Job Broken or Disabled
The Messaging Gateway agent status will remain $START\_SCHEDULED$ if the Oracle Scheduler job associated with a Messaging Gateway agent has become disabled or broken for some reason. To determine if this is the case, use the $DBA\_SCHEDULER\_JOBS$ view to look at $STATE$ field for the agent’s Scheduler job. Normally the Scheduler job state will be $SCHEDULED$ when the Messaging Gateway agent’s Scheduler job is waiting to be run, or $RUNNING$ when the Messaging Gateway agent is running. The agent’s Scheduler job should not exist if the Messaging Gateway agent status is $NOT\_STARTED$. 

See Also: Oracle Messaging Gateway Agent Scheduler Job on page 19-4 for information about Oracle Scheduler objects used by Messaging Gateway.
Check other Scheduler views, such as `DBA_SCHEDULER_JOB_RUN_DETAILS`, for additional information about the Messaging Gateway Scheduler jobs. Also check the `MGW_GATEWAY` view and the Messaging Gateway log file for any error messages that may indicate a problem.

---

**See Also:** Oracle Messaging Gateway Agent Scheduler Job on page 19-4 for information about Oracle Scheduler objects used by Messaging Gateway

---

**Real Application Clusters (RAC) Environment**

If Messaging Gateway is being used in a RAC environment and the agent has been configured with a database service but no database instances are running that have the service enabled, then the Messaging Gateway `AGENT_STATUS` will remain `START_SCHEDULED` until the service is started on a running database instance.

---

**Monitoring Oracle Messaging Gateway Propagation**

Messaging Gateway propagation can be monitored using the `MGW_JOBS` view and the Messaging Gateway log file. The view provides information on propagated messages and errors that may have occurred during propagation attempts. The log file can be used to determine the cause of the errors.

Besides showing configuration information, the `MGW_JOBS` view also has dynamic information that can be used to monitor message propagation. Applicable fields include `STATUS`, `ENABLED`, `PROPAGATED_MSGS`, `EXCEPTIONQ_MSGS`, `FAILURES`, `LAST_ERROR_MSG`, `LAST_ERROR_DATE`, and `LAST_ERROR_TIME`.

The `STATUS` field indicates current status of the job. `READY` means that the job is ready for propagation (but only if the `ENABLED` field is `TRUE`). `RETRY` means that a propagation failure occurred but that propagation will be retried. `FAILED` means that the agent has stopped propagation for the job due to an unrecoverable error or the maximum number of consecutive propagation failures has been reached. `DELETE_PENDING` means job removal is pending due to `DBMS_MGWADM.REMOVE_JOB` being called but certain cleanup tasks pertaining to the job are still outstanding. `SUBSCRIBER_DELETE_PENDING` means that `DBMS_MGWADM.REMOVE_SUBSCRIBER` has been called on an old style propagation job but certain cleanup tasks pertaining to the job are still outstanding.

The `ENABLED` field indicates whether the propagation job is currently enabled. `TRUE` indicates the job is enabled while `FALSE` indicates the job is disabled. No propagation will occur unless the job is enabled.

The `PROPAGATED_MSGS` field of the `MGW_JOBS` view indicates how many messages have been successfully propagated. This field is reset to zero when the Messaging Gateway agent is started.

If a Messaging Gateway propagation job has been configured with an exception queue, then the Messaging Gateway agent will move messages to that exception queue the first time the Messaging Gateway agent encounters a propagation failure caused by a message conversion failure. A message conversion failure is indicated by `oracle.mgw.common.MessageException` in the Messaging Gateway log file. The `EXCEPTIONQ_MSGS` field indicates how many messages have been moved to the exception queue. This field is reset to zero when the Messaging Gateway agent is started.
If an error occurs during message propagation for a propagation job, a count is incremented in the FAILURES field. This field indicates the number of failures encountered since the last successful propagation of messages. Each time a failure occurs, an error message and the time it occurred will be shown by LAST_ERROR_MSG, LAST_ERROR_DATE, and LAST_ERROR_TIME. When the number of failures reaches sixteen, Messaging Gateway halts propagation attempts for this propagation job. To resume propagation attempts you must call DBMS_MGWADM.RESET_JOB for the propagation job.

If an error occurs, then examine the Messaging Gateway log file for further information.

See Also: "DBMS_MGWADM" in Oracle Database PL/SQL Packages and Types Reference

Oracle Messaging Gateway Agent Error Messages

This section lists some of the most commonly occurring errors that are shown in the LAST_ERROR_MSG column of the MGW_JOBS view and logged to the Messaging Gateway agent log file. Also shown are some errors that require special action. When you notice that a failure has occurred, look at the linked exceptions in the log file to determine the root cause of the problem.

Two primary types of errors are logged to the Messaging Gateway agent log file:

- **oracle.mgw.common.MessageException**
  
  This error type is logged when a message conversion failure occurs. The Messaging Gateway agent probably cannot propagate the message causing the failure, and the propagation job will eventually be stopped.

- **oracle.mgw.common.GatewayException**
  
  This error type is logged when some failure other than message conversion occurs. Depending on the cause, the problem may fix itself or require user action.

**[221] Failed to access `<messaging_system>` queue: `<queue>`**

An error occurred while trying to access either an Oracle® Database queue or a non-Oracle queue. Check the linked exception error code and message in the log file.

**[241] Failed to connect to database. SQL error: `<error>`, connect string: `<connect_string>`**

This is probably caused by incorrect MGW agent connection information specified for DBMS_MGWADM.CREATE_AGENT or DBMS_MGWADM.ALTER_AGENT. Either the Messaging Gateway agent user or password is incorrect or the database specifier (database parameter) is incorrect. Verify that the connection information is correct for the connection type used by the agent, JDBC OCI or JDBC Thin.

If the database parameter is NULL, then check the Messaging Gateway log file for the following Oracle linked errors:

- ORA-01034: ORACLE not available
- ORA-27101: shared memory realm does not exist

These two errors together indicate that the Messaging Gateway agent is attempting to connect to the database using a local IPC connection, but the ORACLE_SID value is not correct.
A local connection is used when the database parameter is set to NULL. If a local connection is desired, the correct ORACLE_SID value must be set in the Messaging Gateway agent process. This can be done by adding the following line to the MGW initialization file, usually mgw.ora:

\[
\text{set ORACLE_SID} = \text{sid_value}
\]

ORACLE_SID need not be set in the MGW initialization file if the database parameter is not NULL.

If setting ORACLE_SID in the MGW initialization file does not work, then the database parameter must be set to a value that is not NULL.

If the JDBC Thin connection is used, then the database parameter must be not NULL. If the JDBC Thin connection is used and the database parameter is a TNSNames alias, make sure that the oracle.net.tns_names Java property is set in the MGW initialization file. The property can be set by adding the following line to the MGW initialization file:

\[
\text{setJavaProp oracle.net.tns_admin= &lt;directory containing tnsnames.ora&gt;}
\]

---

**See Also:** "oracle.net.tns_admin" on page 18-12 for more information

---

### [415] Missing messages from source queue of job \(<job_name>\)

Possible causes include:

- The agent partially processed persistent messages that were dequeued by someone other than the Messaging Gateway agent.
- The propagation source queue was purged or re-created.
- A message was moved to the Oracle® Database exception queue.

If this error occurs, then call procedure CLEANUP_GATEWAY in the DBMS_MGWADM package:

\[
\text{DBMS_MGWADM.CLEANUP_GATEWAY (}
\text{action => DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE,}
\text{sarg => &lt;job_name&gt;);}\]

The call takes effect only if the propagation job has encountered the missing message problem and the agent is running. The agent treats the missing messages as nonpersistent messages and continues processing the propagation job.

**See Also:** "Propagation Job Overview" on page 19-15 for more information on Messaging Gateway exception queues

### [416] Missing log records in receiving log queue for job \(<job_name>\)

Possible causes include:

- Log records were dequeued from the log queues by someone other than the Messaging Gateway agent.
- The log queues were purged or re-created.

If this error occurs, then call procedure CLEANUP_GATEWAY in the DBMS_MGWADM package:

\[
\text{DBMS_MGWADM.CLEANUP_GATEWAY (}
\text{action => DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC,}
\text{sarg => &lt;job_name&gt;);}\]
The call takes effect only if the propagation job has encountered the missing log records problem and the agent is running.

---

**Note:** Calling procedure `DBMS_MGWADM.CLEANUP_GATEWAY` may result in duplicated messages if the missing messages have already been propagated to the destination queue. Users should check the source and destination queues for any messages that exist in both places. If such messages exist, then they should be removed from either the source or destination queue before calling this procedure.

---

**[417] Missing log records in sending log queue for job <job_name>**

See previous error.

---

**[421] WARNING: Unable to get connections to recover job <job_name>**

This message is a warning message indicating that the Messaging Gateway agent failed to get a connection to recover the propagation job, because other propagation jobs are using them all. The agent will keep trying to get a connection until it succeeds.

If this message is repeated many times for a WebSphere MQ link, then increase the maximum number of connections used by the Messaging Gateway link associated with the propagation job.

**See Also:** "Altering a Messaging System Link" on page 19-11

---

**[434] Failed to access queue <queue>; provider queue <queue>**

This message indicates that a messaging system native queue cannot be accessed. The queue may have been registered by `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE`, or it may be an Oracle® Database queue. The linked exceptions should give more information.

Possible causes include:

- The foreign queue was registered incorrectly, or the Messaging Gateway link was configured incorrectly.
  
  Verify configuration information. If possible, use the same configuration information to run a sample application of the non-Oracle messaging system.

- The non-Oracle messaging system is not accessible.
  
  Check that the non-Oracle messaging system is running and can be accessed using the information supplied in the Messaging Gateway link.

- The Oracle® Database queue does not exist. Perhaps the queue was removed after the Messaging Gateway propagation job was created.
  
  Check that the Oracle® Database queue still exists.

---

**[436] LOW MEMORY WARNING: total memory = < >, free_mem = < >**

The Messaging Gateway agent JVM is running low on memory. Java garbage collection will be invoked, but this may represent a JVM heap size that is too small. Use the `max_memory` parameter of `DBMS_MGWADM.ALTER_AGENT` to increase the JVM heap size. If the Messaging Gateway agent is running, then it must be restarted for this change to take effect.
Failed to retrieve information for transformation <transformation_id>

The Messaging Gateway agent could not obtain all the information it needs about the transformation. The transformation parameter of `DBMS_MGWADM.CREATE_JOB` must specify the name of the registered transformation and not the name of the transformation function.

Possible causes include:

- The transformation does not exist. Verify that the transformation has been created. You can see this from the following query performed as user SYS:
  ```sql
  SELECT TRANSFORMATION_ID, OWNER FROM DBA_TRANSFORMATIONS;
  ```
- The wrong transformation is registered with Messaging Gateway. Verify that the transformation registered is the one intended.
- The Messaging Gateway agent user does not have `EXECUTE` privilege on the object type used for the `from_type` or the `to_type` of the transformation indicated in the exception.

It is not sufficient to grant `EXECUTE` to `MGW_AGENT_ROLE` and then grant `MGW_AGENT_ROLE` to the agent user. You must grant `EXECUTE` privilege on the object type directly to the agent user or to `PUBLIC`.

Example 21–3 shows such a case for the `from_type`. It also shows the use of linked exceptions for determining the precise cause of the error.

**Example 21–3  No EXECUTE Privilege on Object Type**

Errors occurred during processing of job JOB_AQ2MQ_2

```java
oracle.mgw.common.GatewayException: [703] Failed to retrieve information for transformation mgwuser.SAMPLEADT_TO_MGW_BASIC_MSG
[…Java stack trace here…]
[Linked-exception]
java.sql.SQLException: "from_type" is null
[…Java stack trace here…]
```

AQ payload type <type> not supported; queue: <queue>

The payload type of the Oracle® Database queue used by a Messaging Gateway propagation job is not directly supported by Messaging Gateway. For non-JMS propagation, Messaging Gateway directly supports the payload types `RAW`, `SYS.MGW_BASIC_MSG_T` and `SYS.MGW_TIBRV_MSG_T`.

Possible actions include:

- Configure the Messaging Gateway propagation job to use a transformation that converts the queue payload type to a supported type.
- Remove the Messaging Gateway propagation job and create a new job that uses an Oracle® Database queue with a supported payload type.

For **Java Message Service** (JMS) propagation, the Messaging Gateway propagation job must be removed and a new job created whose Oracle® Database payload type is supported by **Oracle Java Message Service** (OJMS). Transformations are not supported for JMS propagation.

Transformation type <type> not supported; queue: <queue_name>, transform: <transformation>

A Messaging Gateway propagation job was configured with a transformation that uses an object type that is not one of the Messaging Gateway canonical types.
For an outbound job, the transformation from_type must be the Oracle® Database payload type, and the to_type must be a Messaging Gateway canonical type. For an inbound job, the transformation from_type must be a Messaging Gateway canonical type and the to_type must be the Oracle® Database payload type.

[722] Message transformation failed; queue: <queue_name>, transform: <transformation>
An error occurred while attempting execution of the transformation. ORA-25229 is typically thrown by Oracle® Database when the transformation function raises a PL/SQL exception or some other Oracle error occurs when attempting to use the transformation.

Possible causes include:

- The Messaging Gateway agent user does not have EXECUTE privilege on the transformation function. This is illustrated in Example 21–4.

  It is not sufficient to grant EXECUTE to MGW_AGENT_ROLE and then grant MGW_AGENT_ROLE to the Messaging Gateway agent user. You must grant EXECUTE privilege on the transformation function directly to the Messaging Gateway agent user or to PUBLIC.

  Example 21–4  No EXECUTE Privilege on Transformation Function

  Errors occurred during processing of job JOB_MQ2AQ_2
  […]Java stack trace here…
  [Linked-exception]
  […]Java stack trace here…
  [Linked-exception]
  java.sql.SQLException: ORA-25229: error on transformation of message msgid:
  9749DB80C85B0BD4E03408002086745E
  ORA-00604: error occurred at recursive SQL level 1
  ORA-00904: invalid column name
  […]Java stack trace here…

- The transformation function does not exist, even though the registered transformation does. If the transformation function does not exist, it must be re-created.

- The Messaging Gateway agent user does not have EXECUTE privilege on the payload object type for the queue indicated in the exception.

  It is not sufficient to grant EXECUTE to MGW_AGENT_ROLE and then grant MGW_AGENT_ROLE to the Messaging Gateway agent user. You must grant EXECUTE privilege on the object type directly to the Messaging Gateway agent user or to PUBLIC.

- The transformation function raised the error. Verify that the transformation function can handle all messages it receives.

[724] Message conversion not supported; to AQ payload type: <type>, from type: <type>
A Messaging Gateway propagation job is configured for inbound propagation where the canonical message type generated by the non-Oracle messaging system link is not
compatible with the Oracle® Database queue payload type. For example, propagation from a TIB/Rendezvous messaging system to an Oracle® Database queue with a SYS.MGW_BASIC_MSG_T payload type, or propagation from WebSphere MQ to an Oracle® Database queue with a SYS.MGW_TIBRV_MSG_T payload type.

Possible actions include:

- Configure the Messaging Gateway propagation job with a transformation that maps the canonical message type generated by the non-Oracle messaging link to the Oracle® Database payload type.
- Remove the Messaging Gateway propagation job and create a new job whose Oracle® Database queue payload type matches the canonical message type generated by the non-Oracle link.

[725] **Text message not supported for RAW payload**

A Messaging Gateway propagation job is configured for inbound propagation to an Oracle® Database destination having a RAW payload type. A text message was received from the source (non-Oracle) queue resulting in a message conversion failure.

If support for text data is required, remove the Messaging Gateway propagation job and create a new job to an Oracle® Database destination whose payload type supports text data.

[726] **Message size <size> too large for RAW payload; maximum size is <size>**

A Messaging Gateway propagation job is configured for inbound propagation to an Oracle® Database destination having a RAW payload type. A message conversion failure occurred when a message containing a large RAW value was received from the source (non-Oracle) queue.

If large data support is required, remove the Messaging Gateway propagation job and create a new job to an Oracle® Database destination whose payload type supports large data, usually in the form of an object type with a BLOB attribute.

[728] **Message contains too many large (BLOB) fields**

The source message contains too many fields that must be stored in BLOB types. SYS.MGW_TIBRV_MSG_T is limited to three BLOB fields. Reduce the number of large fields in the message, perhaps by breaking them into smaller fields or combining them into fewer large fields.

[729] **Message contains too many large (CLOB) fields**

The source message contains too many fields that contain a large text value that must be stored in CLOB. SYS.MGW_TIBRV_MSG_T is limited to three CLOB fields. Reduce the number of large fields in the message, perhaps by breaking them into smaller fields or combining them into fewer large fields.

[805] **MQSeries Message error while enqueuing to queue: <queue>**

WebSphere MQ returned an error when an attempt was made to put a message in a WebSphere MQ queue. Check the linked exception error code and message in the log file. Consult WebSphere MQ documentation.
Using ANYDATA Queues for User Messages

This chapter describes how to use and manage Oracle® Database when enqueuing and propagating. It describes ANYDATA queues and user messages.

Oracle Streams uses queues of type ANYDATA to store three types of messages:

- **Captured logical change record (LCR)**
  
  This message type, produced by an Oracle Streams capture process, is not discussed in this guide.

  **See Also:** "Streams Capture Process" in Oracle Streams Concepts and Administration

- **User-enqueued LCR**
  
  This is a message containing an LCR that was enqueued by a user or application.

- **User message**
  
  This is a non-LCR message created and enqueued by a user or application.

All three types of messages can be used for information sharing within a single database or between databases.

This chapter contains these topics:

- ANYDATA Queues and User Messages
- Message Propagation and ANYDATA Queues
- Enqueuing User Messages in ANYDATA Queues
- Dequeuing User Messages from ANYDATA Queues
- Propagating User Messages from ANYDATA Queues to Typed Queues
- Propagating User-Enqueued LCRs from ANYDATA Queues to Typed Queues

**See Also:** Oracle Database PL/SQL Packages and Types Reference for more information about the ANYDATA type

### ANYDATA Queues and User Messages

This section contains these topics:

- ANYDATA Wrapper for User Messages Payloads
- Programmatic Interfaces for Enqueue and Dequeue of User Messages

**See Also:** Oracle Streams Concepts and Administration
**ANYDATA Wrapper for User Messages Payloads**

You can wrap almost any type of payload in an ANYDATA payload with the Convert\_data\_type static functions of the ANYDATA type, where data\_type is the type of object to wrap. These functions take the object as input and return an ANYDATA object.

The following datatypes cannot be wrapped in an ANYDATA wrapper:

- Nested table
- NCLOB
- ROWID and UROWID

The following datatypes can be directly wrapped in an ANYDATA wrapper, but they cannot be present in a user-defined type payload wrapped in an ANYDATA wrapper:

- CLOB
- BLOB
- BFILE
- VARRAY

**See Also:** Oracle Database PL/SQL Packages and Types Reference for more information about the ANYDATA type

**Programmatic Interfaces for Enqueue and Dequeue of User Messages**

Your applications can use the following programmatic interfaces to enqueue user messages into an ANYDATA queue and dequeue user messages from an ANYDATA queue:

- PL/SQL (DBMS\_AQ package)
- Java Message Service (JMS)
- OCI

The following sections provide information about using these interfaces to enqueue user messages into and dequeue user messages from an ANYDATA queue.

**See Also:** Chapter 3, "Oracle® Database: Programmatic Interfaces" for more information about these programmatic interfaces

**Enqueuing User Messages Using PL/SQL**

To enqueue a user message containing an LCR into an ANYDATA queue using PL/SQL, first create the LCR to be enqueued. You use the constructor for the SYS.LCR\$_ROW\_RECORD type to create a row LCR, and you use the constructor for the SYS.LCR\$_DDL\_RECORD type to create a DDL LCR. Then you use the ANYDATA.ConvertObject function to convert the LCR into an ANYDATA payload and enqueue it using the DBMS\_AQ.ENQUEUE procedure.

To enqueue a user message containing a non-LCR object into an ANYDATA queue using PL/SQL, you use one of the ANYDATA.Convert\* functions to convert the object into an ANYDATA payload and enqueue it using the DBMS\_AQ.ENQUEUE procedure.
Enqueuing User Messages Using OCI or JMS

To enqueue a user message containing an LCR into an ANYDATA queue using JMS or OCI, you must represent the LCR in XML format. To construct an LCR, use the oracle.xdb.XMLType class. LCRs are defined in the SYS schema. The LCR schema must be loaded into the SYS schema using the catxlc.sql script in ORACLE_HOME/rdbms/admin.

To enqueue a message using OCI, perform the same actions that you would to enqueue a message into a typed queue. To enqueue a message using JMS, a user must have EXECUTE privilege on the DBMS_AQ, DBMS_AQIN, and DBMS_AQJMS packages.

A non-LCR user message can be a message of any user-defined type or a JMS type. The JMS types include the following:

- javax.jms.TextMessage
- javax.jms.MapMessage
- javax.jms.StreamMessage
- javax.jms.ObjectMessage
- javax.jms.BytesMessage

When using user-defined types, you must generate the Java class for the message using Jpublisher, which implements the ORAData interface. To enqueue a message into an ANYDATA queue, you can use methods QueueSender.send or TopicPublisher.publish.

See Also:

- "Enqueuing Messages" on page 10-2
- Chapter 23, "Oracle Streams Messaging Example"
- Oracle Streams Concepts and Administration, "Managing a Streams Messaging Environment"

Note: Enqueue of JMS types and XML types does not work with ANYDATA queues unless you call DBMS_AQADM.ENABLE_JMS_TYPES(queue_table_name) after DBMS_STREAMS_ADM.SET_UP_QUEUE(queue_name). Enabling a queue for these types may affect import/export of the queue table.

See Also:

- "Enqueuing and Dequeuing Messages Using JMS" on page 23-18
- Oracle XML DB Developer’s Guide for more information about representing messages in XML format
- Oracle Streams Advanced Queuing Java API Reference for more information about the oracle.jms Java package
- The OCIAQenq function in the Oracle Call Interface Programmer’s Guide for more information about enqueuing messages using OCI
Dequeuing User Messages Using PL/SQL

To dequeue a user message from an ANYDATA queue using PL/SQL, you use the DBMS_AQ.DEQUEUE procedure and specify ANYDATA as the payload. The user message can contain an LCR or another type of object.

See Also: "Dequeuing Messages" on page 10-13

Dequeuing User Messages Using OCI or JMS

In an ANYDATA queue, user messages containing LCRs in XML format are represented as oracle.xdb.XMLType. Non-LCR messages can be any user-defined type or a JMS type.

To dequeue a message from an ANYDATA queue using JMS, you can use methods QueueReceiver, TopicSubscriber, or TopicReceiver. Because the queue can contain different types of objects wrapped in ANYDATA wrappers, you must register a list of SQL types and their corresponding Java classes in the type map of the JMS session. JMS types are already preregistered in the type map.

For example, suppose a queue contains user-enqueued LCR messages represented as oracle.xdb.XMLType and non-LCR messages of type person and address. The classes JPerson.java and JAddress.java are the ORAData mappings for person and address, respectively. Before dequeuing the message, the type map must be populated as follows:

```java
Map map = ((AQjmsSession)q_sess).getTypeMap();
map.put("SCOTT.PERSON", Class.forName("JPerson"));
map.put("SCOTT.ADDRESS", Class.forName("JAddress"));
map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLType"));  // For LCRs
```

When using a messageSelector with a QueueReceiver or TopicPublisher, the selector can contain any SQL expression that has a combination of one or more of the following:

- **JMS message** header fields or properties
  - These include JMSPriority, JMSCorrelationID, JMSType, JMSXUserID, JMSXAppID, JMSXGroupID, and JMSXGroupSeq. An example of a JMS message field messageSelector is:
    
    JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  
- **User-defined message properties**
  - An example of a user-defined message properties messageSelector is:
    
    color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
  
- **PL/SQL functions**
  - An example of a PL/SQL function messageSelector is:
    
    hr.GET_TYPE(tab.user_data) = 'HR.EMPLOYEES'

To dequeue a message from an ANYDATA queue using OCI, perform the same actions that you would to dequeue a message from a typed queue.
Message Propagation and ANYDATA Queues

ANYDATA queues can interoperate with typed queues. Table 22–1 shows the types of propagation possible between queues.

Table 22–1 Propagation Between Different Types of Queues

<table>
<thead>
<tr>
<th>Source Queue</th>
<th>Destination Queue</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANYDATA</td>
<td>ANYDATA</td>
<td>None</td>
</tr>
<tr>
<td>Typed</td>
<td>ANYDATA</td>
<td>Implicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Propagation is possible only if the messages in the typed queue meet the restrictions outlined in &quot;Object Type Support&quot; on page 1-3.</td>
</tr>
<tr>
<td>ANYDATA</td>
<td>Typed</td>
<td>Requires a rule to filter messages and a user-defined transformation. Only messages containing a payload of the same type as the typed queue can be propagated to the typed queue.</td>
</tr>
<tr>
<td>Typed</td>
<td>Typed</td>
<td>Follows Oracle® Database rules</td>
</tr>
</tbody>
</table>

Note: Propagations cannot propagate user-enqueued ANYDATA messages that encapsulate payloads of object types, varrays, or nested tables between databases with different character sets. Propagations can propagate such messages between databases with the same character set.

Although you cannot use Simple Object Access Protocol (SOAP) to interact directly with an ANYDATA queue, you can use SOAP by propagating messages between an ANYDATA queue and a typed queue. If you want to enqueue a message into an ANYDATA queue using SOAP, you must first configure propagation from a typed queue to the ANYDATA queue. Then you can use SOAP to enqueue a message into the typed queue. The message is propagated automatically from the typed queue to the ANYDATA queue.

If you want to use SOAP to dequeue a message that is in an ANYDATA queue, then you can configure propagation from the ANYDATA queue to a typed queue. The message is propagated automatically from the ANYDATA queue to the typed queue, where it is available for access using SOAP.

See Also: "Propagating Messages Between an ANYDATA Queue and a Typed Queue" in Oracle Streams Concepts and Administration
Enqueuing User Messages in ANYDATA Queues

This section provides examples of enqueuing messages into an ANYDATA queue. The examples assume you are in a SQL*Plus testing environment with access to two databases named db01 and db02. The first few examples prepare the testing environment for the other examples in this chapter.

In Example 22-1, you connect as a user with administrative privileges at databases db01 and db02 to create administrator user strmadmin and to grant EXECUTE privilege on the DBMS_AQ package to sample schema user oe.

**Example 22–1 Creating ANYDATA Users**

GRANT EXECUTE ON DBMS_AQ TO oe;
CREATE USER strmadmin IDENTIFIED BY strmadmin DEFAULT TABLESPACE example;
GRANT DBA TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;

In Example 22–2, you connect to db01 as strmadmin to create ANYDATA queue oe_queue_any. The oe user is configured automatically as a secure user of the oe_queue_any queue and is given ENQUEUE and DEQUEUE privileges on the queue.

**Example 22–2 Creating an ANYDATA Queue**

CONNECT strmadmin;
Enter password: password
BEGIN
   DBMS_STREAMS_ADM.SET_UP_QUEUE(
      queue_table  => 'oe_qtab_any',
      queue_name   => 'oe_queue_any',
      queue_user   => 'oe');
END;
/

In Example 22–3, you add a subscriber to the oe_queue_any queue. This subscriber performs explicit dequeues of messages. The ADD_SUBSCRIBER procedure will automatically create an AQ_AGENT.

**Example 22–3 Adding a Subscriber to the ANYDATA Queue**

DECLARE
   subscriber SYS.AQ$_AGENT;
BEGIN
   subscriber := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
   SYS.DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name => 'strmadmin.oe_queue_any',
      subscriber => subscriber);
END;
/

In Example 22–4, you associate the oe user with the local_agent agent.

**Example 22–4 Associating a User with an AQ_AGENT**

BEGIN
   DBMS_AQADM.ENABLE_DB_ACCESS(
      agent_name => 'local_agent',
      db_username => 'oe');
END;
In Example 22–5, you connect to database db01 as user oe to create an enqueue procedure. It takes an object of ANYDATA type as an input parameter and enqueues a message containing the payload into an existing ANYDATA queue.

**Example 22–5 Creating an Enqueue Procedure**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on
CREATE PROCEDURE oe.enq_proc (payload ANYDATA)  IS
  enqopt DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid RAW(16);
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  DBMS_AQ.ENQUEUE(
    queue_name          =>  'strmadmin.oe_queue_any',
    enqueue_options     =>  enqopt,
    message_properties  =>  mprop,
    payload             =>  payload,
    msgid               =>  enq_msgid);
END;
/
```

In Example 22–6, you use procedure oe.enq_proc to enqueue a message of type VARCHAR2 into an ANYDATA queue.

**Example 22–6 Enqueuing a VARCHAR2 Message into an ANYDATA Queue**

```sql
EXEC oe.enq_proc(ANYDATA.ConvertVarchar2('Chemicals - SW'));
COMMIT;
```

In Example 22–7, you use procedure oe.enq_proc to enqueue a message of type NUMBER into an ANYDATA queue.

**Example 22–7 Enqueuing a NUMBER Message into an ANYDATA Queue**

```sql
EXEC oe.enq_proc(ANYDATA.ConvertNumber('16'));
COMMIT;
```

In Example 22–8, you use procedure oe.enq_proc to enqueue a user-defined type message into an ANYDATA queue.

**Example 22–8 Enqueuing a User-Defined Type Message into an ANYDATA Queue**

```sql
BEGIN
  oe.enq_proc(ANYDATA.ConvertObject(oe.cust_address_typ('1646 Brazil Blvd','361168','Chennai','Tam', 'IN')));
END;
/
COMMIT;
```

**See Also:** "Viewing the Contents of User-Enqueued Events in a Queue" in Oracle Streams Concepts and Administration
Dequeuing User Messages from ANYDATA Queues

This section provides examples of dequeuing messages from an ANYDATA queue. The examples assume that you have completed the examples in “Enqueuing User Messages in ANYDATA Queues” on page 22-6.

To dequeue messages, you must know the consumer of the messages. To find the consumer for the messages in a queue, connect as the owner of the queue and query the AQ$queue_table_name view, where queue_table_name is the name of the queue table containing the queue.

In Example 22–9, you connect to database db01 as strmadmin, the owner of queue oe_queue_any, and perform a query on the AQ$OE_QTAB_ANY view. The query returns three rows, with LOCAL_AGENT as the CONSUMER_NAME in each row.

**Example 22–9  Determining the Consumer of Messages in a Queue**

```sql
CONNECT strmadmin;
Enter password: password
SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_QTAB_ANY;
```

In Example 22–10, you connect to database db01 as user oe to create a dequeue procedure that takes as an input the consumer of the messages you want to dequeue, dequeues messages of oe.cust_address_typ, and prints the contents of the messages.

**Example 22–10  Creating a Dequeue Procedure for an ANYDATA Queue**

```sql
CONNECT oe; -- @db01
Enter password: password

CREATE PROCEDURE oe.get_cust_address (consumer IN VARCHAR2) AS
    address         OE.CUST_ADDRESS_TYP;
    deq_address     ANYDATA;
    msgid           RAW(16);
    deqopt          DBMS_AQ.DEQUEUE_OPTIONS_T;
    mprop           DBMS_AQ.MESSAGE_PROPERTIES_T;
    new_addresses   BOOLEAN := TRUE;
    next_trans      EXCEPTION;
    no_messages     EXCEPTION;
    pragma exception_init (next_trans, -25235);
    pragma exception_init (no_messages, -25228);
    num_var         pls_integer;
BEGIN
    deqopt.consumer_name := consumer;
    deqopt.wait := 1;
    WHILE (new_addresses) LOOP
        BEGIN
            DBMS_AQ.DEQUEUE(
                queue_name          =>  'strmadmin.oe_queue_any',
                dequeue_options     =>  deqopt,
                message_properties  =>  mprop,
                payload             =>  deq_address,
                msgid               =>  msgid);
            deqopt.navigation := DBMS_AQ.NEXT;
            DBMS_OUTPUT.PUT_LINE('****');
            IF (deq_address.GetTypeName() = 'OE.CUST_ADDRESS_TYP') THEN
                DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' || deq_address.GetTypeName());
                num_var := deq_address.GetObject(address);
```
DBMS_OUTPUT.PUT_LINE(' **** CUSTOMER ADDRESS **** ');
DBMS_OUTPUT.PUT_LINE(address.street_address);
DBMS_OUTPUT.PUT_LINE(address.postal_code);
DBMS_OUTPUT.PUT_LINE(address.city);
DBMS_OUTPUT.PUT_LINE(address.state_province);
DBMS_OUTPUT.PUT_LINE(address.country_id);
ELSE
  DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' || deq_address.GetTypeName());
END IF;
COMMIT;
EXCEPTION
  WHEN next_trans THEN
    deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
  WHEN no_messages THEN
    new_addresses := FALSE;
    DBMS_OUTPUT.PUT_LINE('No more messages');
END;
END LOOP;
END;
/

In Example 22–11, you use procedure oe.get_cust_address, created in Example 22–10, specifying LOCAL_AGENT as the consumer.

Example 22–11  Dequeuing Messages from an ANYDATA Queue

SET SERVEROUTPUT ON SIZE 100000
EXEC oe.get_cust_address('LOCAL_AGENT');

The example returns:

****
Message TYPE is: SYS.VARCHAR2
****
Message TYPE is: SYS.NUMBER
****
Message TYPE is: OE.CUST_ADDRESS_TYP
**** CUSTOMER ADDRESS ****
1646 Brazil Blvd
361168
Chennai
Tam
IN
No more messages

Propagating User Messages from ANYDATA Queues to Typed Queues

This section provides examples showing how to propagate non-LCR user messages between an ANYDATA queue and a typed queue.

Note: The examples in this section assume that you have completed the examples in "Enqueuing User Messages in ANYDATA Queues" on page 22-6.

See Also: "Message Propagation and ANYDATA Queues" on page 22-5 for more information about propagation between ANYDATA and typed queues.
The first few examples set up propagation from the ANYDATA queue oe_queue_any, created in Example 22–2 on page 22-6, to a typed queue in database db02. In Example 22–12, you connect as sample schema user oe to grant EXECUTE privilege on oe.cust_address_typ at databases db01 and db02 to administrator user strmadmin.

**Example 22–12  Granting EXECUTE Privilege on a Type**

CONNECT oe; -- @db01
Enter password: password

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
CONNECT oe; -- @db02
Enter password: password

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;

In Example 22–13, you connect to database db02 as administrator user strmadmin and create a destination queue of type oe.cust_address_typ.

**Example 22–13  Creating a Typed Destination Queue**

CONNECT strmadmin;
Enter password: password

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table => 'strmadmin.oe_qtab_address',
    queue_payload_type => 'oe.cust_address_typ',
    multiple_consumers => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name => 'strmadmin.oe_queue_address',
    queue_table => 'strmadmin.oe_qtab_address');
  DBMS_AQADM.START_QUEUE(
    queue_name => 'strmadmin.oe_queue_address');
END;
/

In Example 22–14, you connect to database db01 as administrator user strmadmin to create a database link from db01 to db02.

**Example 22–14  Creating a Database Link**

CONNECT strmadmin;
Enter password: password

CREATE DATABASE LINK db02 CONNECT TO strmadmin IDENTIFIED BY password
  USING 'db02';

In Example 22–15, you create function any_to_cust_address_typ in the strmadmin schema at db01 that takes an ANYDATA payload containing an oe.cust_address_typ object and returns an oe.cust_address_typ object.

**Example 22–15  Creating a Function to Extract a Typed Object from an ANYDATA Object**

CONNECT strmadmin;
Enter password: password

CREATE FUNCTION strmadmin.any_to_cust_address_typ(in_any IN ANYDATA) RETURN OE.CUST_ADDRESS_TYP
Propagating User Messages from ANYDATA Queues to Typed Queues

Using ANYDATA Queues for User Messages

AS
address OE.CUST_ADDRESS_TYP;
num_var NUMBER;
type_name VARCHAR2(100);
BEGIN
    type_name := in_any.GetTypeName();
    IF (type_name = 'OE.CUST_ADDRESS_TYP') THEN
        num_var := in_any.GetObject(address);
        RETURN address;
    ELSE
        raise_application_error(-20101, 'Conversion failed - ' || type_name);
    END IF;
END;
/

In Example 22–16, you create a transformation at db01 using the DBMS_TRANSFORM package.

Example 22–16 Creating an ANYDATA to Typed Object Transformation

BEGIN
    DBMS_TRANSFORM.CREATE_TRANSFORMATION(
        schema => 'strmadmin',
        name => 'anytoaddress',
        from_schema => 'SYS',
        from_type => 'ANYDATA',
        to_schema => 'oe',
        to_type => 'cust_address_typ',
        transformation => 'strmadmin.any_to_cust_address_typ(source.user_data)');
END;
/

In Example 22–17, you create a subscriber for the typed queue. The subscriber must contain a rule that ensures that only messages of the appropriate type are propagated to the destination queue.

Example 22–17 Creating Subscriber ADDRESS_AGENT_REMOTE

DECLARE
    subscriber SYS.AQ$_AGENT;
BEGIN
    subscriber := SYS.AQ$_AGENT ('ADDRESS_AGENT_REMOTE',
        'STRMADMIN.OE_QUEUE_ADDRESS@DB02',
        0);
    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'strmadmin.oe_queue_any',
        subscriber => subscriber, 
        rule => 'TAB.USER_DATA.GetTypeName()=''OE.CUST_ADDRESS_TYP''',
        transformation => 'strmadmin.anytoaddress');
END;
/

In Example 22–18, you schedule propagation between the ANYDATA queue at db01 and the typed queue at db02.

Example 22–18 Scheduling Propagation from an ANYDATA Queue to a Typed Queue

BEGIN
    DBMS_AQADM.SCHEDULE_PROPAGATION(
        queue_name => 'strmadmin.oe_queue_any',
        ...
In Example 22–19, you connect to database db01 as sample schema user oe to enqueue a message of oe.cust_address_typ type wrapped in an ANYDATA wrapper. This example uses the enqueue procedure oe.enq_proc created in Example 22–5 on page 22-7.

**Example 22–19  Enqueuing a Typed Message in an ANYDATA Wrapper**

```sql
CONNECT oe;
Enter password:
BEGIN
    oe.enq_proc(ANYDATA.ConvertObject(oe.cust_address_typ('1668 Chong Tao','111181','Beijing',NULL,'CN')));
END;
/
COMMIT;
```

After allowing some time for propagation, in Example 22–20 you query queue table AQ$OE_QTAB_ADDRESS at db02 to view the propagated message.

**Example 22–20  Viewing the Propagated Message**

```sql
CONNECT strmadmin;
Enter password:
SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_QTAB_ADDRESS;
```

The example returns one message for ADDRESS_AGENT_REMOTE:

<table>
<thead>
<tr>
<th>MSG_ID</th>
<th>MSG_STATE</th>
<th>CONSUMER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBEF5CACC4665A6FE030578CE70A370D</td>
<td>READY</td>
<td>ADDRESS_AGENT_REMOTE</td>
</tr>
</tbody>
</table>

1 row selected.

**See Also:** Chapter 20, "Oracle Messaging Gateway Message Conversion" for more information about transformations during propagation

### Propagating User-Enqueued LCRs from ANYDATA Queues to Typed Queues

You can propagate user-enqueued LCRs to an appropriate typed queue, but propagation of captured LCRs to a typed queue is not supported.

**See Also:** "Streams Capture Process" in Oracle Streams Concepts and Administration for more information on capture processes

To propagate user-enqueued LCRs from an ANYDATA queue to a typed queue, you complete the same steps as you do for non-LCR messages, but Oracle supplies the transformation functions. You can use the following functions in the `DBMS_STREAMS` package to transform LCRs in ANYDATA queues to messages in typed queues:

- `CONVERT_ANYDATA_TO_LCR_ROW` transforms an ANYDATA payload containing a row LCR into a `SYS.LCR$_ROW_RECORD` payload.
CONVERT_ANYDATA_TO_LCR_DDL transforms an ANYDATA payload containing a DDL LCR into a SYS.LCR$_DDL_RECORD payload.

The examples in this section set up propagation of row LCRs from an ANYDATA queue named oe_queue_any to a typed queue of type SYS.LCR$_ROW_RECORD named oe_queue_lcr. The source queue oe_queue_any is at database db01, and the destination queue oe_queue_lcr is created at database db02 in Example 22–21.

---

**Example 22–21 Creating a Queue of Type LCR$_ROW_RECORD**

CONNECT strmadmin;
Enter password: password

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'strmadmin.oe_qtab_lcr',
    queue_payload_type  => 'SYS.LCR$_ROW_RECORD',
    multiple_consumers  => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name   => 'strmadmin.oe_queue_lcr',
    queue_table  => 'strmadmin.oe_qtab_lcr');
  DBMS_AQADM.START_QUEUE(
    queue_name => 'strmadmin.oe_queue_lcr');
END;
/

In Example 22–22, you connect to db01 as administrator user strmadmin to create an ANYDATA to LCR$_ROW_RECORD transformation at db01 using the DBMS_TRANSFORM package.

**Example 22–22 Creating an ANYDATA to LCR$_ROW_RECORD Transformation**

CONNECT strmadmin;
Enter password: password

BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema         => 'strmadmin',
    name           => 'anytolcr',
    from_schema    => 'SYS',
    from_type      => 'ANYDATA',
    to_schema      => 'SYS',
    to_type        => 'LCR$_ROW_RECORD',
    transformation =>
      'SYS.DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(source.user_data)');
END;
/

In Example 22–23, you create a subscriber at the typed queue. The subscriber specifies the anytolcr transformation created in Example 22–22 for the transformation parameter.

**Example 22–23 Creating Subscriber ROW_LCR_AGENT_REMOTE**

DECLARE
  subscriber  SYS.AQ$_AGENT;

---

Note: The examples in this section assume you have already run the examples in the preceding sections of this chapter.
BEGIN
subscriber := SYS.AQ$_AGENT('ROW_LCR_AGENT_REMOTE', 'STRMADMIN.OE_QUEUE_LCR@DB02', 0);
DBMS_AQADM.ADD_SUBSCRIBER{
queue_name => 'strmadmin.oe_queue_any',
subscriber => subscriber,
rule => 'TAB.USER_DATA.GetTypeName()=''SYS.LCR$_ROW_RECORD''',
transformation => 'strmadmin.anytolcr'};
END;
/

In Example 22–24, you connect to database db01 as sample schema user oe to create a procedure to construct and enqueue a row LCR into the strmadmin.oe_queue_any queue.

**Example 22–24 Creating a Procedure to Construct and Enqueue a Row LCR**

CONNECT oe;
Enter password: password

CREATE PROCEDURE oe.enq_row_lcr_proc{
source_dbname VARCHAR2,
cmd_type VARCHAR2,
obj_owner VARCHAR2,
obj_name VARCHAR2,
old_vals SYS.LCR$_ROW_LIST,
new_vals SYS.LCR$_ROW_LIST
AS
eopt DBMS_AQ.ENQUEUE_OPTIONS_T;
mprop DBMS_AQ.MESSAGE_PROPERTIES_T;
enq_msgid RAW(16);
row_lcr SYS.LCR$_ROW_RECORD;
BEGIN
mprop.SENDER_ID := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT{
source_database_name => source_dbname,
command_type => cmd_type,
object_owner => obj_owner,
object_name => obj_name,
old_values => old_vals,
new_values => new_vals};
DBMS_AQ.ENQUEUE{
queue_name => 'strmadmin.oe_queue_any',
enqueue_options => eopt,
message_properties => mprop,
payload => ANYDATA.ConvertObject(row_lcr),
msgid => enq_msgid};
END enq_row_lcr_proc;
/

In Example 22–25, you use the oe.enq_row_lcr_proc procedure first to create a row LCR that inserts a row into the oe.inventories table, and then to enqueue the row LCR into the strmadmin.oe_queue_any queue.
Example 22–25  Creating and Enqueuing a Row LCR

DECLARE
    newunit1  SYS.LCR$_ROW_UNIT;
    newunit2  SYS.LCR$_ROW_UNIT;
    newunit3  SYS.LCR$_ROW_UNIT;
    newvals   SYS.LCR$_ROW_LIST;
BEGIN
    newunit1 := SYS.LCR$_ROW_UNIT(
        'PRODUCT_ID',
        ANYDATA.ConvertNumber(3503),
        DBMS_LCR.NOT_A_LOB,
        NULL,
        NULL);
    newunit2 := SYS.LCR$_ROW_UNIT(
        'WAREHOUSE_ID',
        ANYDATA.ConvertNumber(1),
        DBMS_LCR.NOT_A_LOB,
        NULL,
        NULL);
    newunit3 := SYS.LCR$_ROW_UNIT(
        'QUANTITY_ON_HAND',
        ANYDATA.ConvertNumber(157),
        DBMS_LCR.NOT_A_LOB,
        NULL,
        NULL);
    newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3);
    oe.enq_row_lcr_proc(
        source_dbname  =>  'DB01',
        cmd_type       =>  'INSERT',
        obj_owner      =>  'OE',
        obj_name       =>  'INVENTORIES',
        old_vals       =>  NULL,
        new_vals       =>  newvals);
END;
/
COMMIT;

The LCR is propagated to database db02 by the schedule created in Example 22–18 on page 22-11. After allowing some time for propagation, in Example 22–26 you query queue table AQ$OE_QTAB_LCR at db02 to view the propagated message.

Example 22–26  Viewing the Propagated LCR

CONNECT strmadmin;
Enter password: password

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_QTAB_LCR;

The example returns one message for ROW_LCR_AGENT_REMOTE:

<table>
<thead>
<tr>
<th>MSG_ID</th>
<th>MSG_STATE</th>
<th>CONSUMER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECE2B0F912DDFF5EE030578CE70A04BB</td>
<td>READY</td>
<td>ROW_LCR_AGENT_REMOTE</td>
</tr>
</tbody>
</table>
See Also: "DBMS_STREAMS" in *Oracle Database PL/SQL Packages and Types Reference* for more information about the row LCR and DDL LCR conversion functions
Oracle Streams Messaging Example

The examples in this chapter illustrate a messaging environment that can be constructed using Oracle Streams. The examples assume you are in a SQL*Plus testing environment with access to a database named db01.

This chapter contains these topics:
- Overview of Messaging Example
- Setting Up Users and Creating an ANYDATA Queue
- Creating Enqueue Procedures
- Configuring an Apply Process
- Configuring Explicit Dequeue
- Enqueuing Messages
- Dequeuing Messages Explicitly and Querying for Applied Messages
- Enqueuing and Dequeuing Messages Using JMS

See Also: Oracle Streams Concepts and Administration for more information about messaging and ANYDATA queues

Overview of Messaging Example

This example illustrates using a single ANYDATA queue to create an Oracle Streams messaging environment in which message payloads of different types are stored in the same queue. Specifically, this example illustrates the following messaging features of Oracle Streams:

- Enqueuing messages containing order payload as ANYDATA payloads
- Enqueuing messages containing customer payload as ANYDATA payloads
- Enqueuing messages containing row LCRs as ANYDATA payloads
- Creating a rule set for applying the events
- Creating an evaluation context used by the rule set
- Creating an Oracle Streams apply process to dequeue and process the events based on rules
- Creating a message handler and associating it with the apply process
- Explicitly dequeuing and processing events based on rules without using the apply process

Figure 23–1 provides an overview of this environment.
### Setting Up Users and Creating an ANYDATA Queue

Because the examples in this chapter use the `oe` sample schema, the `oe` user must have privileges to run the subprograms in the `DBMS_AQ` package. This is accomplished in Example 23–1.

Note: The `oe` user is specified as the queue user when the ANYDATA queue is created in Example 23–2 on page 23-3. The `SET_UP_QUEUE` procedure grants the `oe` user `enqueue` and `dequeue` privileges on the queue, but the `oe` user also needs `EXECUTE` privilege on the `DBMS_AQ` package to enqueue and dequeue messages.

Most of the configuration and administration actions illustrated in these examples are performed by the Oracle Streams administrator `strmadmin`. Example 23–1 also creates this user and grants the necessary privileges. These privileges enable the user to run subprograms in packages related to Oracle Streams, create rule sets, create rules, and monitor the Oracle Streams environment by querying data dictionary views.

In Example 23–1, you connect to database `db01` as a user with administrative privileges.

---

**Figure 23–1 Example Oracle Streams Messaging Environment**

Oracle Database `oedb.net`
- `oe.enq_row_lcr` PL/SQL Procedure
  - Enqueue Events
- `oe.enq_proc` PL/SQL Procedure
  - Enqueue Events
- `strmadmin.oe_queue`
  - `customer_event_typ` payload event
  - `LCR_ROW_RECORD` payload event
  - `order_event_typ` payload event
- `explicit_dq` PL/SQL Procedure
  - Explicit Dequeue by Application
- `apply_oe` PL/SQL Procedure
  - Apply Process
  - Apply LCR Events Directly
- `oe.mes_handler` PL/SQL Procedure
  - Apply Changes
- `hr.customers`
- `hr.orders`
- `oe.enq_row_lcr`
- `oe.enq_proc`
- `oe.explicit_dq`
- `apply_oe`
- `oe.mes_handler`
Example 23–1  Setting Up ANYDATA Users

GRANT EXECUTE ON DBMS_AQ TO oe;
CREATE USER strmadmin IDENTIFIED BY strmadmin DEFAULT TABLESPACE example;
GRANT DBA, SELECT_CATALOG_ROLE TO strmadmin;
GRANT EXECUTE ON DBMS_APPLY_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_AQ TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;
BEGIN
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
END;
/

Note:

- The SELECT_CATALOG_ROLE is not required for the Oracle Streams administrator. It is granted in this example so that the Oracle Streams administrator can monitor the environment easily.
- If you plan to use the Oracle Streams tool in Oracle Enterprise Manager, then grant the Oracle Streams administrator SELECT ANY DICTIONARY privilege, in addition to the privileges shown in this step.

In Example 23–2, you connect to database db01 as administrator user strmadmin to create ANYDATA queue oe_queue. The SET_UP_QUEUE procedure creates a queue table for the queue and then creates and starts the queue.

Example 23–2 Creating an ANYDATA Queue

set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on
BEGIN
DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'oe_queue_table',
    queue_name  => 'oe_queue');
END;
/

In Example 23–3, you connect to database db01 as administrator user strmadmin to grant the oe user privileges on queue oe_queue, create agent explicit_enq that
will be used to perform explicit enqueue operations on the queue, and associate the oe user with the agent.

Queue oe_queue is a secure queue because it was created using `SET_UP_QUEUE`. For a user to perform enqueue and dequeue operations on a secure queue, the user must be configured as a secure queue user of the queue. Associating the oe user with agent explicit_enq enables the oe user to perform enqueue operations on this queue.

**Example 23–3  Enabling Enqueue on the ANYDATA Queue**

```
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
    SYS.DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
        privilege  => 'ALL',
        queue_name => 'strmadmin.oe_queue',
        grantee    => 'oe');
    SYS.DBMS_AQADM.CREATE_AQ_AGENT(
        agent_name  => 'explicit_enq');
    DBMS_AQADM.ENABLE_DB_ACCESS(
        agent_name  => 'explicit_enq',
        db_username => 'oe');
END;
/
```

Creating Enqueue Procedures

The examples in this section create two PL/SQL procedures that enqueue messages into the ANYDATA queue oe_queue. One procedure enqueues non-LCR messages, and the other procedure enqueues row LCR messages.

In **Example 23–4**, you connect to database db01 as sample schema user oe to create a type to represent orders based on the columns in the oe.orders table. This type is used for messages that are enqueued into the ANYDATA queue oe_queue. The type attributes include the columns in the oe.orders table, along with one extra attribute named action. The value of the action attribute for instances of this type is used to determine the correct action to perform on the instance (either apply process dequeue or explicit dequeue).

**Example 23–4  Creating an Orders Type**

```
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE TYPE order_event_typ AS OBJECT(
    order_id       NUMBER(12),
    order_date     TIMESTAMP(6) WITH LOCAL TIME ZONE,
    order_mode     VARCHAR2(8),
    customer_id    NUMBER(6),
    order_status   NUMBER(2),
    order_total    NUMBER(8,2),
    sales_rep_id   NUMBER(6),
);```
Creating Enqueue Procedures

Oracle Streams Messaging Example

23-5

In Example 23–5, you connect to database db01 as sample schema user oe to create a type to represent customers based on the columns in the oe.customers table. This type is used for messages that are enqueued into the ANYDATA queue oe_queue. The type attributes include the columns in the oe.customers table, along with one extra attribute named action. The value of the action attribute for instances of this type is used to determine the correct action to perform on the instance (either apply process dequeue or explicit dequeue).

Example 23–5 Creating a Customers Type

set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE TYPE customer_event_typ AS OBJECT(
    customer_id         NUMBER(6),
    cust_first_name     VARCHAR2(20),
    cust_last_name      VARCHAR2(20),
    cust_address        CUST_ADDRESS_TYP,
    phone_numbers       PHONE_LIST_TYP,
    nls_language        VARCHAR2(3),
    nls_territory       VARCHAR2(30),
    credit_limit        NUMBER(9,2),
    cust_email          VARCHAR2(30),
    account_mgr_id      NUMBER(6),
    date_of_birth       DATE,
    marital_status      VARCHAR2(20),
    gender              VARCHAR2(1),
    income_level        VARCHAR2(20),
    action              VARCHAR(7));
/

In Example 23–6, you connect to database db01 as sample schema user oe to create a PL/SQL procedure called enq_proc to enqueue non-LCR messages into ANYDATA queue oe_queue.

Example 23–6 Creating a Procedure to Enqueue Non-LCR Messages

set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

Note: This example assumes you have dropped the cust_geo_location column from the oe.customers table. This column is useful only with Oracle Spatial.

Note: A single enqueued message can be dequeued by both an apply process and an explicit dequeue, but the examples in this chapter do not illustrate this capability.
CREATE PROCEDURE oe.enq_proc (event IN ANYDATA) IS
    enqopt       DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_eventid  RAW(16);
BEGIN
    mprop.SENDER_ID := SYS.AQ$AGENT('explicit_enq', NULL, NULL);
    DBMS_AQ.ENQUEUE(
        queue_name => 'strmadmin.oe_queue',
        enqueue_options => enqopt,
        message_properties => mprop,
        payload => event,
        msgid => enq_eventid);
END;
/

In Example 23–7, you connect to database db01 as sample schema user oe to create a PL/SQL procedure called enq_row_lcr that constructs a row LCR and then enqueues the row LCR into ANYDATA queue oe_queue.

See Also: Oracle Database PL/SQL Packages and Types Reference for more information about LCR constructors

Example 23–7 Creating a Procedure to Construct and Enqueue Row LCR Events

set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE PROCEDURE oe.enq_row_lcr (source_dbname VARCHAR2, cmd_type VARCHAR2, obj_owner VARCHAR2, obj_name VARCHAR2, old_vals SYS.LCR$ROW_LIST, new_vals SYS.LCR$ROW_LIST) AS
    eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_msgid      RAW(16);
    row_lcr        SYS.LCR$ROW_RECORD;
BEGIN
    mprop.SENDER_ID := SYS.AQ$AGENT('explicit_enq', NULL, NULL);
    row_lcr := SYS.LCR$ROW_RECORD.CONSTRUCT(
        source_database_name => source_dbname,
        command_type => cmd_type,
        object_owner => obj_owner,
        object_name => obj_name,
        old_values => old_vals,
        new_values => new_vals);
    DBMS_AQ.ENQUEUE(
        queue_name => 'strmadmin.oe_queue',
        enqueue_options => eopt,
        message_properties => mprop,
        payload => ANYDATA.ConvertObject(row_lcr),
        msgid => enq_msgid);
END enq_row_lcr;
/
Configuring an Apply Process

The examples in this section configure an apply process to apply the user-enqueued messages in the ANYDATA queue oe_queue.

In Example 23–8, you connect to database db01 as sample schema user oe to create a function called get_oe_action and to grant EXECUTE privilege on the function to administrator user strmadmin.

This function determines the value of the action attribute in the messages in queue oe_queue. It is used in rules later in this chapter to determine the value of the action attribute for an event. Then, the clients of the rules engine perform the appropriate action for the event (either dequeue by apply process or explicit dequeue). In this example, the clients of the rules engine are the apply process and the oe.explicit_dq PL/SQL procedure.

Example 23–8 Creating a Function to Determine the Value of the Action Attribute

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE FUNCTION oe.get_oe_action (event IN ANYDATA) RETURN VARCHAR2 IS
  ord oe.order_event_typ;
  cust oe.customer_event_typ;
  num NUMBER;
  type_name VARCHAR2(61);
BEGIN
  type_name := event.GETTYPENAME;
  IF type_name = 'OE.ORDER_EVENT_TYP' THEN
    num := event.GETOBJECT(ord);
    RETURN ord.action;
  ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN
    num := event.GETOBJECT(cust);
    RETURN cust.action;
  ELSE
    RETURN NULL;
  END IF;
END;
/
GRANT EXECUTE ON get_oe_action TO strmadmin;
```

In Example 23–9, you connect to database db01 as sample schema user oe to create a PL/SQL procedure called mes_handler that is used as a message handler by the apply process. You also grant EXECUTE privilege on this procedure to administrator user strmadmin. This procedure takes the payload in a user-enqueued message of type oe.order_event_typ or oe.customer_event_typ and inserts it as a row in the oe.orders table or oe.customers table, respectively.

Example 23–9 Creating a Message Handler

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on
```
CREATE PROCEDURE oe_mes_handler (event IN ANYDATA) IS
    ord    oe.order_event_typ;
cust    oe.customer_event_typ;
num    NUMBER;
type_name VARCHAR2(61);
BEGIN
    type_name := event.GETTYPENAME;
    IF type_name = 'OE.ORDER_EVENT_TYP' THEN
        num := event.GETOBJECT(ord);
        INSERT INTO oe.orders VALUES (ord.order_id, ord.order_date,
            ord.order_mode, ord.customer_id, ord.order_status, ord.order_total,
            ord.sales_rep_id, ord.promotion_id);
    ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN
        num := event.GETOBJECT(cust);
        INSERT INTO oe.customers VALUES (cust.customer_id, cust.cust_first_name,
            cust.cust_last_name, cust.cust_address, cust.phone_numbers,
            cust.nls_language, cust.nls_territory, cust.credit_limit, cust.cust_email,
            cust.account_mgr_id, cust.date_of_birth, cust.marital_status,
            cust.gender, cust.income_level);
    END IF;
END IF;
END;
/
GRANT EXECUTE ON mes_handler TO strmadmin;

In Example 23–10, you connect to database db01 as administrator user strmadmin to create an evaluation context for the rule set.

**Example 23–10  Creating an Evaluation Context for the Rule Set**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on
DECLARE
table_alias SYS.RE$TABLE_ALIAS_LIST;
BEGIN
    table_alias := SYS.RE$TABLE_ALIAS_LIST(
        SYS.RE$TABLE_ALIAS('tab', 'strmadmin.oe_queue_table'));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
        evaluation_context_name => 'oe_eval_context',
        table_aliases => table_alias);
END;
/
```

In Example 23–11, you connect to database db01 as administrator user strmadmin to create a rule set for the apply process.

**Example 23–11  Creating a Rule Set for the Apply Process**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on
BEGIN
    DBMS_RULE_ADM.CREATE_RULE_SET(
```
In Example 23–12, you connect to database db01 as administrator user strmadmin to create a rule that evaluates to TRUE if the action value of a message is apply. Notice that tab.user_data is passed to the oe.get_oe_action function. The tab.user_data column holds the event payload in a queue table. The table alias for the queue table was specified as tab in Example 23–10 on page 23-8.

Example 23–12  Creating a Rule that Evaluates to TRUE if Action Is Apply

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name => 'strmadmin.apply_action',
        condition => 'oe.get_oe_action(tab.user_data) = ''APPLY'' ');
END;
/
```

In Example 23–13, you connect to database db01 as administrator user strmadmin to create a rule that evaluates to TRUE if the event in the queue is a row LCR that changes either the oe.orders table or the oe.customers table. This rule enables the apply process to apply user-enqueued changes to the tables directly.

For convenience, this rule uses the Oracle-supplied evaluation context SYS.STREAMS$_EVALUATION_CONTEXT because the rule is used to evaluate LCRs. When this rule is added to the rule set, the Oracle-supplied evaluation context is used for the rule during evaluation instead of evaluation context oe_eval_context created in Example 23–10 on page 23-8.

Example 23–13  Creating a Rule that Evaluates to TRUE for Row LCR Events

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name => 'apply_lcrs',
        condition => ':dml.GET_OBJECT_OWNER() = ''OE'' AND ' ||
                    ':dml.GET_OBJECT_NAME() = ''ORDERS'' OR ' ||
                    ':dml.GET_OBJECT_NAME() = ''CUSTOMERS'' );
        evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
END;
/
```

In Example 23–14, you connect to database db01 as administrator user strmadmin to add the apply_action rule created in Example 23–12 on page 23-9 and the apply_lcrs rule created in Example 23–13 on page 23-9 to the apply_oe_rs rule set created in Example 23–11 on page 23-8.
Example 23–14 Adding Rules to the Rule Set

```
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'apply_action',
    rule_set_name => 'apply_oe_rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'apply_lcrs',
    rule_set_name => 'apply_oe_rs');
END;
/
```

In Example 23–15, you connect to database db01 as administrator user strmadmin to create an apply process that is associated with queue oe_queue, that uses the apply_oe_rs rule set, and that uses the mes_handler procedure as a message handler.

Example 23–15 Creating an Apply Process

```
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name => 'strmadmin.oe_queue',
    apply_name => 'apply_oe',
    rule_set_name => 'strmadmin.apply_oe_rs',
    message_handler => 'oe.mes_handler',
    apply_user => 'oe',
    apply_captured => false);
END;
/
```

Because oe was specified as the apply user when the apply process was created in Example 23–15 on page 23-10, you must grant this user EXECUTE privilege on the strmadmin.apply_oe_rs rule set used by the apply process. You connect to database db01 as administrator user strmadmin to accomplish this in Example 23–16.

Example 23–16 Granting EXECUTE Privilege on the Rule Set To oe User

```
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege => DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => 'strmadmin.apply_oe_rs',
    grantee => 'oe',
    grant_option => FALSE);
END;
```
In Example 23–17, you connect to database db01 as administrator user strmadmin to start the apply process with the `disable_on_error` parameter set to `n` so that the apply process is not disabled if it encounters an error.

**Example 23–17  Starting the Apply Process**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_oe',
    parameter   => 'disable_on_error',
    value       => 'n');
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply_oe');
END;
/```

**Configuring Explicit Dequeue**

The examples in this section illustrate how to configure explicit dequeue of messages based on message contents.

In Example 23–18, you connect to database db01 as administrator user strmadmin to create agent `explicit_dq`. This agent is used to perform explicit dequeue operations on the `oe_queue` queue.

**Example 23–18  Creating an Agent for Explicit Dequeue**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on
BEGIN
  SYS.DBMS_AQADM.CREATE_AQAGENT{
    agent_name => 'explicit_dq'};
END;
/```

The `oe_queue` queue is a secure queue because it was created using `SET_UP_QUEUE` in Example 23–2 on page 23-3. For a user to perform enqueue and dequeue operations on a secure queue, the user must be configured as a secure queue user of the queue.

In Example 23–19, you connect to database db01 as administrator user strmadmin to associate the `oe` user with agent `explicit_dq`. The `oe` user is able to perform dequeue operations on the `oe_queue` queue when the agent is used to create a `subscriber` to the queue in Example 23–20 on page 23-12.

**Example 23–19  Associating User oe with Agent explicit_dq**

```sql
set echo off
/```
Configuring Explicit Dequeue

```sql
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

BEGIN
DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_dq',
    db_username => 'oe');
END;
/
```

In Example 23–20, you connect to database db01 as administrator user strmadmin to add a subscriber to the oe_queue queue. This subscriber will perform explicit dequeues of messages. A subscriber rule is used to dequeue any messages where the action value is not apply. If the action value is apply for a message, then the message is ignored by the subscriber. Such messages are dequeued and processed by the apply process.

**Example 23–20  Adding a Subscriber to the oe_queue Queue**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for STRMADMIN: ' HIDE
CONNECT strmadmin/&password@db01;
set echo on

DECLARE
    subscriber SYS.AQ$_AGENT;
BEGIN
    subscriber := SYS.AQ$_AGENT('explicit_dq', NULL, NULL);
    SYS.DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name  => 'strmadmin.oe_queue',
        subscriber  => subscriber,
        rule        => 'oe.get_oe_action(tab.user_data) != ''APPLY''');
END;
/
```

In Example 23–21, you connect to database db01 as sample schema user oe to create a PL/SQL procedure called explicit_dq to dequeue messages explicitly using the subscriber created in Example 23–20 on page 23-12.

The procedure commits after the dequeue of the messages. The commit informs the queue that the dequeued messages have been consumed successfully by this subscriber.

The procedure can process multiple transactions and uses two exception handlers. Exception handler next_trans moves to the next transaction, and exception handler no_messages exits the loop when there are no more messages.

**Example 23–21  Creating a Procedure to Dequeue Messages Explicitly**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE PROCEDURE oe.explicit_dq (consumer IN VARCHAR2) AS
deqopt    DBMS_AQ.DEQUEUE_OPTIONS_T;
```
mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
msgid        RAW(16);
payload      ANYDATA;
new_messages BOOLEAN := TRUE;
ord          oe.order_event_typ;
cust         oe.customer_event_typ;
tc           pls_integer;
next_trans   EXCEPTION;
no_messages  EXCEPTION;
pragma exception_init (next_trans, -25235);
predicate exception_init (no_messages, -25228);
BEGIN
deqopt.consumer_name := consumer;
deqopt.wait := 1;
WHILE (new_messages) LOOP
BEGIN
DBMS_AQ.DEQUEUE(
    queue_name          =>  'strmadmin.oe_queue',
dequeue_options     =>  deqopt,
message_properties  =>  mprop,
payload             =>  payload,
msgid               =>  msgid);
COMMIT;
deqopt.navigation := DBMS_AQ.NEXT;
DBMS_OUTPUT.PUT_LINE('Message Dequeued');
DBMS_OUTPUT.PUT_LINE('Type Name := ' || payload.GetTypeName);
IF (payload.GetTypeName = 'OE.ORDER_EVENT_TYP') THEN
tc := payload.GetObject(ord);
DBMS_OUTPUT.PUT_LINE('order_id     - ' || ord.order_id);
DBMS_OUTPUT.PUT_LINE('order_date   - ' || ord.order_date);
DBMS_OUTPUT.PUT_LINE('order_mode   - ' || ord.order_mode);
DBMS_OUTPUT.PUT_LINE('customer_id  - ' || ord.customer_id);
DBMS_OUTPUT.PUT_LINE('order_status - ' || ord.order_status);
DBMS_OUTPUT.PUT_LINE('order_total  - ' || ord.order_total);
DBMS_OUTPUT.PUT_LINE('sales_rep_id - ' || ord.sales_rep_id);
DBMS_OUTPUT.PUT_LINE('promotion_id - ' || ord.promotion_id);
END IF;
IF (payload.GetTypeName = 'OE.CUSTOMER_EVENT_TYP') THEN
tc := payload.GetObject(cust);
DBMS_OUTPUT.PUT_LINE('customer_id       - ' || cust.customer_id);
DBMS_OUTPUT.PUT_LINE('cust_first_name   - ' || cust.cust_first_name);
DBMS_OUTPUT.PUT_LINE('cust_last_name    - ' || cust.cust_last_name);
DBMS_OUTPUT.PUT_LINE('street_address    - ' || cust.cust_address.street_address);
DBMS_OUTPUT.PUT_LINE('postal_code       - ' || cust.cust_address.postal_code);
DBMS_OUTPUT.PUT_LINE('city              - ' || cust.cust_address.city);
DBMS_OUTPUT.PUT_LINE('state_province    - ' || cust.cust_address.state_province);
DBMS_OUTPUT.PUT_LINE('country_id        - ' || cust.cust_address.country_id);
DBMS_OUTPUT.PUT_LINE('phone_number1     - ' || cust.phone_numbers(1));
DBMS_OUTPUT.PUT_LINE('phone_number2     - ' || cust.phone_numbers(2));
DBMS_OUTPUT.PUT_LINE('phone_number3     - ' || cust.phone_numbers(3));
DBMS_OUTPUT.PUT_LINE('nls_language      - ' || cust.nls_language);
DBMS_OUTPUT.PUT_LINE('nls_territory     - ' || cust.nls_territory);
DBMS_OUTPUT.PUT_LINE('credit_limit      - ' || cust.credit_limit);
DBMS_OUTPUT.PUT_LINE('cust_email        - ' || cust.cust_email);
DBMS_OUTPUT.PUT_LINE('account_mgr_id    - ' || cust.account_mgr_id);
DBMS_OUTPUT.PUT_LINE('date_of_birth     - ' || cust.date_of_birth);
END IF;
END;
END;
END;
Enqueuing Messages

The examples in this section illustrate how to enqueue non-LCR messages and row LCR messages into a queue.

---

**Note:** It is possible to dequeue user-enqueued LCRs explicitly, but these examples do not illustrate this capability.

---

In Example 23–22, you connect to database db01 as sample schema user oe to enqueue two messages with apply for the action value. Based on the apply process rules, the apply process dequeues and processes these messages with the oe.mes_handler message handler procedure created in Example 23–9 on page 23-7. The COMMIT after the enqueues makes these two enqueues part of the same transaction. An enqueued message is not visible until the session that enqueued it commits the enqueue.

**Example 23–22   Enqueuing Non-LCR Messages to Be Dequeued by an Apply Process**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

BEGIN
  oe.enq_proc(ANYDATA.convertobject(oe.order_event_typ(
      2500,'05-MAY-01','online',117,3,44699,161,NULL,'APPLY')));
END;
/

BEGIN
  oe.enq_proc(ANYDATA.convertobject(oe.customer_event_typ(
      990,'Hester','Prynne',oe.cust_address_typ('555 Beacon Street','02109','Boston','MA','US'),oe.phone_list_typ('+1 617 123 4104','+1 617 083 4381','+1 617 742 5813'),'i','AMERICA',5000,'a@scarlet_letter.com',145,NULL,'SINGLE','F','UNDER 50,000','APPLY')));
END;
/
COMMIT;
```

In Example 23–23, you connect to database db01 as sample schema user oe to enqueue two messages with dequeue for the action value. The oe.explicit_dq
procedure created in Example 23–21 on page 23-12 dequeues these messages because the action is not apply. Based on the apply process rules, the apply process ignores these messages. The COMMIT after the enqueues makes these two enqueues part of the same transaction.

**Example 23–23  Enqueuing Non-LCR Messages to Be Dequeued Explicitly**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

BEGIN
  oe.enq_proc(ANYDATA.convertobject(oe.order_event_typ(2501,'22-JAN-00','direct',117,3,22788,161,NULL,'DEQUEUE')));
END;
/
BEGIN
  oe.enq_proc(ANYDATA.convertobject(oe.customer_event_typ(991,'Nick','Carraway',oe.cust_address_typ('10th Street','11101','Long Island','NY','US'),oe.phone_list_typ('+1 718 786 2287','+1 718 888 4832','+1 718 511 9114'),'i','AMERICA',3000,'nick@great_gatsby.com',149,NULL,'MARRIED','M','OVER 150,000','DEQUEUE')));
END;
/
COMMIT;
```

In Example 23–24, you connect to database db01 as sample schema user oe to create a row LCR that inserts a row into the oe.orders table and another LCR that updates that row. The apply process applies these messages directly.

**Note:** Enqueued LCRs should commit at transaction boundaries. In this example, a COMMIT statement is run after each enqueue, making each enqueue a separate transaction. However, you can perform multiple LCR enqueues before a commit if there is more than one LCR in a transaction.

**Example 23–24  Enqueuing Row LCRs to Be Dequeued by an Apply Process**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

DECLARE
  newunit1  SYS.LCR$_ROW_UNIT;
  newunit2  SYS.LCR$_ROW_UNIT;
  newunit3  SYS.LCR$_ROW_UNIT;
  newunit4  SYS.LCR$_ROW_UNIT;
  newunit5  SYS.LCR$_ROW_UNIT;
  newunit6  SYS.LCR$_ROW_UNIT;
  newunit7  SYS.LCR$_ROW_UNIT;
  newunit8  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
```
The examples in this section illustrate how to dequeue messages explicitly and query messages that were applied by the apply process. The examples use messages that were enqueued in the previous section.

In Example 23–25, you connect to database db01 as sample schema user oe to run procedure explicit_dq, created in Example 23–21 on page 23-12. You specify subscriber explicit_dq, added in Example 23–20 on page 23-12, as the consumer of the messages you want to dequeue. In these examples, messages that are not dequeued explicitly by this procedure are dequeued by the apply process.

**Example 23–25 Dequeuing Messages Explicitly**

```sql
set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
```

```sql
'ORDER_ID', ANYDATA.ConvertNumber(2502), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit2 := SYS.LCR_ROW_UNIT(
    'ORDER_DATE', ANYDATA.ConvertTimestampLTZ('04-NOV-00'), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit3 := SYS.LCR_ROW_UNIT(
    'ORDER_MODE', ANYDATA.ConvertVarchar2('online'), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit4 := SYS.LCR_ROW_UNIT(
    'CUSTOMER_ID', ANYDATA.ConvertNumber(145), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit5 := SYS.LCR_ROW_UNIT(
    'ORDER_STATUS', ANYDATA.ConvertNumber(3), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit6 := SYS.LCR_ROW_UNIT(
    'ORDER_TOTAL', ANYDATA.ConvertNumber(35199), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit7 := SYS.LCR_ROW_UNIT(
    'SALES_REP_ID', ANYDATA.ConvertNumber(160), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newunit8 := SYS.LCR_ROW_UNIT(
    'PROMOTION_ID', ANYDATA.ConvertNumber(1), DBMS_LCR.NOT_A_LOB, NULL, NULL);
newvals := SYS.LCR_ROW_LIST(
    newunit1, newunit2, newunit3, newunit4, newunit5, newunit6, newunit7, newunit8);
end;
/
COMMIT;
```

```sql
DECLARE
oldunit1 SYS.LCR_ROW_UNIT;
oldunit2 SYS.LCR_ROW_UNIT;
oldvals SYS.LCR_ROW_LIST;
nnewunit1 SYS.LCR_ROW_UNIT;
nnewvals SYS.LCR_ROW_LIST;
BEGIN
oldunit1 := SYS.LCR_ROW_UNIT(
    'ORDER_ID', ANYDATA.ConvertNumber(2502), DBMS_LCR.NOT_A_LOB, NULL, NULL);
oldunit2 := SYS.LCR_ROW_UNIT(
    'ORDER_TOTAL', ANYDATA.ConvertNumber(35199), DBMS_LCR.NOT_A_LOB, NULL, NULL);
oldvals := SYS.LCR_ROW_LIST(oldunit1, oldunit2);
nnewunit1 := SYS.LCR_ROW_UNIT(
    'ORDER_TOTAL', ANYDATA.ConvertNumber(5235), DBMS_LCR.NOT_A_LOB, NULL, NULL);
nnewvals := SYS.LCR_ROW_LIST(nnewunit1);
oe.enq_row_lcr('DB01', 'UPDATE', 'OE', 'ORDERS', oldvals, nnewvals);
end;
/
COMMIT;
```
set echo on

CREATE PROCEDURE oe.enq_proc (payload ANYDATA) IS
  SET SERVEROUTPUT ON SIZE 100000;
  EXEC oe.explicit_dq('explicit_dq');

The example returns the payload of the messages enqueued in Example 23–23 on page 23-15:

Message Dequeued
Type Name := OE.ORDER_EVENT_TYP
order_id - 2501
order_date - 22-JAN-00 12.00.00.000000 AM
order_mode - direct
customer_id - 117
order_status - 3
order_total - 22788
sales_rep_id - 161
promotion_id -

Message Dequeued
Type Name := OE.CUSTOMER_EVENT_TYP
customer_id - 991
cust_first_name - Nick
cust_last_name - Carraway
street_address - 10th Street
postal_code - 11101
city - Long Island
state_province - NY
country_id - US
phone_number1 - +1 718 786 2287
phone_number2 - +1 718 511 9114
phone_number3 - +1 718 888 4832
nls_language - i
nls_territory - AMERICA
credit_limit - 3000
cust_email - nick@great_gatsby.com
account_mgr_id - 149
date_of_birth - 
marital_status - MARRIED
gender - M
income_level - OVER 150,000
No more messages

Example 23–26, you connect to database db01 as sample schema user oe to query the oe.orders and oe.customers tables to see the rows corresponding to the messages applied by apply process apply_oe, created in Example 23–15 on page 23-10.

Example 23–26  Querying for Applied Messages

set echo off
set verify off
ACCEPT password CHAR PROMPT 'Enter the password for OE: ' HIDE
CONNECT oe/&password@db01;
set echo on

CREATE PROCEDURE oe.enq_proc (payload ANYDATA) IS
  SELECT order_id, order_date, customer_id, order_total
  FROM oe.orders WHERE order_id = 2500;
  SELECT cust_first_name, cust_last_name, cust_email
  FROM oe.customers WHERE customer_id = 990;
Enqueuing and Dequeuing Messages Using JMS

The examples in this section illustrate how to enqueue non-LCR messages and row LCRs into a queue and then dequeue them using Java Message Service (JMS).

The following jar and zip files should be in the CLASSPATH based on the release of JDK you are using.

For JDK 1.4.x, the CLASSPATH must contain:

```
ORACLE_HOME/jdbc/lib/classes12.jar
ORACLE_HOME/jdbc/lib/ojdbc14.jar
ORACLE_HOME/jlib/jndi.jar
ORACLE_HOME/lib/jta.jar
ORACLE_HOME/rdbms/jlib/aqapi13.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
ORACLE_HOME/rdbms/jlib/xdb.jar
ORACLE_HOME/xdk/lib/xmlparserv2.jar
```

For JDK 1.3.x, the CLASSPATH must contain:

```
ORACLE_HOME/jdbc/lib/classes12.jar
ORACLE_HOME/jlib/jndi.jar
ORACLE_HOME/rdbms/jlib/aqapi13.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
ORACLE_HOME/rdbms/jlib/xdb.jar
ORACLE_HOME/lib/jta.jar
ORACLE_HOME/xdk/lib/xmlparserv2.jar
```

For JDK 1.2.x, the CLASSPATH must contain:

```
ORACLE_HOME/jdbc/lib/classes12.jar
ORACLE_HOME/jlib/jndi.jar
ORACLE_HOME/lib/jta.jar
ORACLE_HOME/rdbms/jlib/aqapi12.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
ORACLE_HOME/rdbms/jlib/xdb.jar
```

```
SELECT order_id, order_date, customer_id, order_total
FROM oe.orders WHERE order_id = 2502;
```

The example returns three rows:

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>ORDER_DATE</th>
<th>CUSTOMER_ID</th>
<th>ORDER_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>05-MAY-01 12.00.00.000000 AM</td>
<td>117</td>
<td>44699</td>
</tr>
</tbody>
</table>

1 row selected.

<table>
<thead>
<tr>
<th>CUST_FIRST_NAME</th>
<th>CUST_LAST_NAME</th>
<th>CUST_EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hester</td>
<td>Prynne</td>
<td><a href="mailto:a@scarlet_letter.com">a@scarlet_letter.com</a></td>
</tr>
</tbody>
</table>

1 row selected.

<table>
<thead>
<tr>
<th>ORDER_ID</th>
<th>ORDER_DATE</th>
<th>CUSTOMER_ID</th>
<th>ORDER_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2502</td>
<td>04-NOV-00 12.00.00.000000 AM</td>
<td>145</td>
<td>5235</td>
</tr>
</tbody>
</table>

1 row selected.
Also, make sure LD_LIBRARY_PATH (Linux and Solaris) or PATH (Windows) includes ORACLE_HOME/lib.

These examples show sample schema user oe enqueuing JMS messages into a queue and agent explicit_dq dequeuing them. Agent explicit_dq was created in Example 23–18 on page 23-11, associated with sample schema user oe in Example 23–19 on page 23-11, and made a subscriber to queue oe_queue in Example 23–20 on page 23-12.

Sample schema user oe was granted EXECUTE on DBMS_AQ in Example 23–1 on page 23-3. In order for this user to use the Oracle JMS interface, it must have EXECUTE privilege on DBMS_AQIN as well. In Example 23–27, you connect to database db01 as a user with administrative privileges to grant the necessary privilege to oe.

See Also: “Accessing Standard and Oracle JMS Applications” on page 3-5

Example 23–27  Granting EXECUTE on DBMS_AQIN to User oe

GRANT EXECUTE on DBMS_AQIN to oe;

Enqueue of JMS types and XML types does not work with Oracle Streams ANYDATA queues unless you call DBMS_AQADM.ENABLE_JMS_TYPES(queue_table_name) after DBMS_STREAMS_ADM.SET_UP_QUEUE(). In Example 23–28, you connect to database db01 as administrator user strmadmin, created in Example 23–1 on page 23-3, to run ENABLE_JMS_TYPES on ANYDATA queue table oe_queue_table, created in Example 23–2 on page 23-3.

Example 23–28  Enabling JMS Types on an ANYDATA Queue

CONNECT strmadmin;
Enter password: password
BEGIN
    DBMS_AQADM.ENABLE_JMS_TYPES('oe_queue_table');
END;
/

Note: Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

In Example 23–29, you connect to database db01 as sample schema user oe to create types address and person.

Example 23–29  Creating Oracle Object Types address and person

CONNECT oe;
Enter password: password
CREATE TYPE address AS OBJECT (street VARCHAR (30), num NUMBER)
/
CREATE TYPE person AS OBJECT (name VARCHAR (30), home ADDRESS)
/

In Example 23–30, you use JPublisher to generate two Java classes named JPerson and JAddress for the person and address types, respectively. The input to JPublisher is a file called input.typ with the following lines:

SQL PERSON AS JPerson
SQL ADDRESS AS JAddress
**Example 23–30  Creating Java Classes That Map to Oracle Object Types**

jpub -input=input.typ -user=OE/OE

Example 23–31 is the Java code that you use to publish JMS text messages, LCRs, and non-LCR ADT messages into an Oracle Streams topic. It does the following:

- Creates a `TopicConnectionFactory` using the JDBC OCI driver
- Creates a `TopicSession`
- Starts the connection
- Creates method `publishUserMessages()` to publish an ADT message and a JMS text message to an Oracle Streams topic
- Creates method `publishLcrMessages()` to publish an XML LCR message to an Oracle Streams topic
- Publishes three messages, providing feedback as it proceeds

Method `publishUserMessages()` does the following:

- Gets the topic
- Creates a publisher
- Specifies agent `explicit_enq` to access queue `oe_queue`
- Creates a PERSON ADT message
- Sets the payload in the message
- Specifies `explicit_dq` as the recipient
- Publishes the PERSON ADT message
- Creates a JMS Text message
- Publishes the JMS Text message

Method `publishLcrMessages()` does the following:

- Gets the topic
- Creates a publisher
- Gets the JDBC connection
- Specifies agent `explicit_enq` to access queue `oe_queue`
- Creates an ADT message
- Creates the LCR representation in XML
- Creates the `XMLType` containing the LCR
- Sets the payload in the message
- Specifies `explicit_dq` as the recipient
- Publishes the LCR

The code is compiled in Example 23–33 on page 23-27. For now, just save it as `StreamsEnq.java`.

**Note:** The JDBC OCI driver is your only choice for accessing Oracle Streams through JMS.
Example 23–31  Java Code for Enqueuing Messages

```
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;

public class StreamsEnq
{
    public static void main (String args [])
    throws java.sql.SQLException, ClassNotFoundException, JMSException
    {
        TopicConnectionFactory tc_fact= null;
        TopicConnection t_conn = null;
        TopicSession t_sess = null;
        try
        {
            if (args.length < 3 )
                System.out.println("Usage:java filename [SID] [HOST] [PORT]*");
            else
                {
                    tc_fact = AQjmsFactory.getTopicConnectionFactory( 
                        args[1], args[0], Integer.parseInt(args[2]), "oci8");
                    t_conn = tc_fact.createTopicConnection("OE","OE");
                    t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
                    t_conn.start();
                    publishUserMessages(t_sess);
                    publishLcrMessages(t_sess);
                    t_sess.close();
                    t_conn.close();
                    System.out.println("End of StreamsEnq Demo") ;
                }
        }
        catch (Exception ex)
        {
            System.out.println("Exception-1: " + ex);
            ex.printStackTrace();
        }
    }

    public static void publishUserMessages(TopicSession t_sess) throws Exception
    {
        Topic topic = null;
        TopicPublisher t_pub = null;
        JPerson pers = null;
        JAddress addr = null;
        TextMessage t_msg = null;
        AdtMessage adt_msg = null;
        AQjmsAgent agent = null;
        AQjmsAgent[] recipList = null;
        try
        {
            topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");
            t_pub = t_sess.createPublisher(topic);
            agent = new AQjmsAgent("explicit_enq", null);
            adt_msg = ((AQjmsSession)t_sess).createAdtMessage();
            pers = new JPerson();
            addr = new JAddress();
```
addr.setNum(new java.math.BigDecimal(500));
addr.setStreet("Oracle Pkwy");
pers.setName("Mark");
pers.setHome(addr);
adt_msg.setAdtPayload(pers);
((AQjmsMessage)adt_msg).setSenderID(agent);
System.out.println("Publish message 1 - type PERSON
");
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);
((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);
t_sess.commit();


t_msg = t_sess.createTextMessage();
t_msg.setText("Test message");
t_msg.setStringProperty("color", 'BLUE');
t_msg.setIntProperty("year", 1999);
((AQjmsMessage)t_msg).setSenderID(agent);
System.out.println("Publish message 2 - type JMS TextMessage
");
((AQjmsTopicPublisher)t_pub).publish(topic, t_msg, recipList);
t_sess.commit();
}

}catch (JMSException jms_ex)
{
    System.out.println("JMS Exception: " + jms_ex);
    if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " + jms_ex.getLinkedException());
}

public static void publishLcrMessages(TopicSession t_sess) throws Exception
{
    Topic topic = null;
    TopicPublisher t_pub = null;
    XMLType xml_lcr = null;
    AdtMessage adt_msg = null;
    AQjmsAgent agent = null;
    StringBuffer lcr_data = null;
    AQjmsAgent[] recipList = null;
    java.sql.Connection db_conn = null;

    try
    {
        topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");
t_pub = t_sess.createPublisher(topic);
db_conn = ((AQjmsSession)t_sess).getDBConnection();
agent = new AQjmsAgent("explicit_eng", null);
adt_msg = ((AQjmsSession)t_sess).createAdtMessage();
lcr_data = new StringBuffer();
lcr_data.append("<ROW_LCR ");
lcr_data.append("xmlns='http://xmlns.oracle.com/streams/schemas/lcr' \n");
lcr_data.append("xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' \n");
lcr_data.append("xsi:schemaLocation='http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd' \n");
lcr_data.append("<source_database_name>source_dbname</source_database_name> \n");
lcr_data.append("<command_type>INSERT</command_type> \n");
lcr_data.append("<object_owner>Ram</object_owner> \n");
lcr_data.append("<object_name>Emp</object_name> \n");

Enqueuing and Dequeuing Messages Using JMS

Oracle Streams Messaging Example

```java
lcr_data.append("<tag>0ABC</tag> 
");
lcr_data.append("<transaction_id>0.0.0</transaction_id> 
");
lcr_data.append("<scn>0</scn> 
");
lcr_data.append("<old_values> 
");
lcr_data.append("<old_value> 
");
lcr_data.append("<column_name>C01</column_name> 
");
lcr_data.append("<data><varchar2>Clob old</varchar2></data> 
");
lcr_data.append("</old_value> 
");
lcr_data.append("<old_value> 
");
lcr_data.append("<column_name>C02</column_name> 
");
lcr_data.append("<data><varchar2>Al23FF</varchar2></data> 
");
lcr_data.append("</old_value> 
");
lcr_data.append("<old_value> 
");
lcr_data.append("<column_name>C03</column_name> 
");
lcr_data.append("<data> 
");
lcr_data.append("<date><value>1997-11-24</value><format>SYYYY-MM-DD</format></date> 
");
lcr_data.append("</data> 
");
```

```java
xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());
adt_msg.setAdtPayload(xml_lcr);
((AQjmsMessage)adt_msg).setSenderID(agent);
System.out.println("Publish message 3 - XMLType containing LCR ROW
");
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);
```

```java
lcr_data.append("<data><varchar2>Al23FF</varchar2></data> 
");
lcr_data.append("<new_value> 
");
lcr_data.append("<column_name>C01</column_name> 
");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> 
");
lcr_data.append("<new_value> 
");
lcr_data.append("<column_name>C02</column_name> 
");
lcr_data.append("<data><number>35.23</number></data> 
");
lcr_data.append("<new_value> 
");
lcr_data.append("<column_name>C03</column_name> 
");
lcr_data.append("<data> 
");
lcr_data.append("<number>-100000</number></data> 
");
```

```java
xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());
adt_msg.setAdtPayload(xml_lcr);
((AQjmsMessage)adt_msg).setSenderID(agent);
System.out.println("Publish message 3 - XMLType containing LCR ROW
");
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);
```

```java
lcr_data.append("<data><varchar2>Hello</varchar2></data> 
");
lcr_data.append("<new_value> 
");
lcr_data.append("<column_name>C05</column_name> 
");
lcr_data.append("<data><char>world</char></data> 
");
```

```java
xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());
adt_msg.setAdtPayload(xml_lcr);
((AQjmsMessage)adt_msg).setSenderID(agent);
System.out.println("Publish message 3 - XMLType containing LCR ROW
");
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);
```

```java
```
Enqueuing and Dequeuing Messages Using JMS

Example 23–32 is the Java code you use to receive messages from a Oracle Streams topic. It does the following:

- Creates a `TopicConnectionFactory` using the JDBC OCI driver
- Creates a `TopicSession`
- Starts the connection
- Creates method `receiveMessages()` to receive messages from an Oracle Streams topic
- Receives three messages, providing feedback as it proceeds

Method `receiveMessages()` does the following:

- Gets the topic
- Creates a `TopicReceiver` to receive messages for consumer `explicit_dq`
- Registers mappings for `ADDRESS` and `PERSON` in the JMS typemap
- Registers a mapping for `XMLType` in the typemap (required for LCRs)
- Receives the enqueued messages

The code is compiled in `Example 23–33` on page 23-27. For now, just save it as `StreamsDeq.java`.

**Example 23–32 Java Code for Dequeuing Messages**

```java
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;
import java.sql.SQLException;

public class StreamsDeq
{
    public static void main (String args [])
        throws java.sql.SQLException, ClassNotFoundException, JMSException
    {
        TopicConnectionFactory tc_fact= null;
        TopicConnection        t_conn = null;
        TopicSession           t_sess = null;
```

**Note:** The JDBC OCI driver is your only choice for accessing Oracle Streams through JMS.
try {
    if (args.length < 3 )
        System.out.println("Usage:java filename [SID] [HOST] [PORT]");
    else {
        tc_fact = AQjmsFactory.getTopicConnectionFactory(
            args[1], args[0], Integer.parseInt(args[2]), "oci8");
        t_conn = tc_fact.createTopicConnection( "OE","OE");
        t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
        t_conn.start();
        receiveMessages(t_sess);
        t_sess.close();
        t_conn.close();
        System.out.println("End of StreamsDeq Demo");
    }
} catch (Exception ex) {
    System.out.println("Exception-1: " + ex);
    ex.printStackTrace();
}

public static void receiveMessages(TopicSession t_sess) throws Exception {
    Topic           topic   = null;
    JPerson         pers    = null;
    JAddress        addr    = null;
    XMLType         xtype   = null;
    TextMessage     t_msg   = null;
    AdtMessage      adt_msg = null;
    Message         jms_msg = null;
    TopicReceiver   t_recv  = null;
    int             i       = 0;
    java.util.Map   map    = null;
    try {
        topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");
        t_recv = ((AQjmsSession)t_sess).createTopicReceiver(topic, "explicit_dq", null);
        map = ((AQjmsSession)t_sess).getTypeMap();
        map.put("OE.PERSON", Class.forName("JPerson"));
        map.put("OE.ADDRESS", Class.forName("JAddress"));
        System.out.println("Receive messages ...
        do {
            try {
                jms_msg = (t_recv.receive(10));
                i++;
                ((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_MESSAGE);
            } catch (JMSException jms_ex2) {
                jms_ex2.printStackTrace();
            }
        }
    }
if((jms_ex2.getLinkedException() != null) &&
   (jms_ex2.getLinkedException() instanceof SQLException))
{
    SQLException sql_ex2 = (SQLException)(jms_ex2.getLinkedException());
    if(sql_ex2.getErrorCode() == 25235) {
      ((AQjmsTopicReceiver)t_recv).setNavigationMode(
          AQjmsConstants.NAVIGATION_NEXT_TRANSACTION);
      continue;
    } else
      throw jms_ex2;
  }
else
  throw jms_ex2;
}
if(jms_msg == null) {
  System.out.println("\nNo more messages");
} else {
  if(jms_msg instanceof AdtMessage) {
    adt_msg = (AdtMessage)jms_msg;
    System.out.println("Retrieved message ' + i + ': " +
        adt_msg.getAdtPayload());
    if(adt_msg.getAdtPayload() instanceof JPerson) {
      pers = (JPerson) adt_msg.getAdtPayload();
      System.out.println("PERSON: Name: ' + pers.getName());
    } else if(adt_msg.getAdtPayload() instanceof JAddress) {
      addr = (JAddress) adt_msg.getAdtPayload();
      System.out.println("ADDRESS: Street ' + addr.getStreet());
    } else if(adt_msg.getAdtPayload() instanceof oracle.xdb.XMLType) {
      xtype = (XMLType) adt_msg.getAdtPayload();
      System.out.println("XMLType: Data: \n ' + xtype.getStringVal());
      System.out.println("Msg id: ' + adt_msg.getJMSMessageID());
      System.out.println();
    } else if(jms_msg instanceof TextMessage) {
      t_msg = (TextMessage)jms_msg;
      System.out.println("Retrieved message ' + i + ': " +
          t_msg.getText());
      System.out.println("Msg id: ' + t_msg.getJMSMessageID());
      System.out.println();
    } else
      System.out.println("Invalid message type");
  }
} while (jms_msg != null);
t_sess.commit();
} catch (JMSException jms_ex)
{
    System.out.println('JMS Exception: ' + jms_ex);
    if(jms_ex.getLinkedException() != null)
        System.out.println('Linked Exception: ' + jms_ex.getLinkedException());
    t_sess.rollback();
}
}
}

In Example 23–33, you compile the scripts.

**Example 23–33  Compiling StreamsEnq.java and StreamsDeq.java**

javac StreamsEnq.java StreamsDeq.java JPerson.java JAddress.java

In Example 23–34, you run the enqueue program, specifying values for **ORACLE_SID**, **HOST**, and **PORT** that are appropriate for your testing environment.

**Example 23–34  Running StreamsEnq**

```
java StreamsEnq  ORACLE_SID HOST PORT
```

The example returns:

Publish message 1 -type PERSON
Publish message 2 -type JMS TextMessage
Publish message 3 - XMLType containing LCR ROW
End of StreamsEnq Demo

In Example 23–35, you run the dequeue program, specifying values for **ORACLE_SID**, **HOST**, and **PORT** that are appropriate for your testing environment.

**Example 23–35  Running StreamsDeq**

```
java StreamsDeq  ORACLE_SID HOST PORT
```
This appendix describes nonpersistent queues, which are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that you use buffered messaging instead.

See Also: "Buffered Messaging" on page 1-12

Oracle® Database can deliver nonpersistent messages asynchronously to subscribers. These messages can be event-driven and do not persist beyond the failure of the system (or instance). The messages are stored in a system-created queue table. Oracle® Database supports persistent and nonpersistent messages with a common API.

Nonpersistent queues, which can be either single-consumer or multiconsumer, provide a mechanism for notification to all currently connected users. Subscribers can be added to multiconsumer nonpersistent queues, and nonpersistent queues can be destinations for propagation.

You use the enqueue interface to enqueue messages into a nonpersistent queue in the usual way. You can enqueue RAW and Oracle object type messages into a nonpersistent queue. OCI notifications are used to deliver such messages to users that are currently registered for notification.

This appendix contains these topics:
- Creating Nonpersistent Queues
- Managing Nonpersistent Queues
- Compatibility of Nonpersistent Queues
- Nonpersistent Queue Notification
- Restrictions on Nonpersistent Queues

Creating Nonpersistent Queues

```sql
DBMS_AQADM.CREATE_NP_QUEUE ( 
  queue_name              IN        VARCHAR2,
  multiple_consumers      IN        BOOLEAN  DEFAULT FALSE,
  comment                 IN        VARCHAR2 DEFAULT NULL);
```

This procedure creates a nonpersistent queue.

Only local recipients are supported for nonpersistent queues. The queue can be either single-consumer or multiconsumer. All queue names must be unique within a schema. The queues are created in an 8.1-compatible system-created queue table (AQ$_MEM_SC or AQ$_MEM_MC) in the same schema as that specified by the queue name. If the queue
name does not specify a schema name, then the queue is created in the login user’s schema.

---

**Note:** Names of nonpersistent queues must not be longer than 24 characters. If you attempt to create a nonpersistent queue with a longer name, error ORA-24019 results.

### Managing Nonpersistent Queues

Once a queue is created with `CREATE_NP_QUEUE`, it can be enabled by calling `START_QUEUE`. By default, the queue is created with both enqueue and dequeue disabled.

You can enqueue RAW and Oracle object type messages into a nonpersistent queue. You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the Oracle Call Interface (OCI) notification mechanism. You cannot invoke the listen call on a nonpersistent queue.

A nonpersistent queue can be dropped only by its owner.

### Compatibility of Nonpersistent Queues

For 8.1-style or higher queues, the `compatible` parameter of `init.ora` and the `compatible` parameter of the queue table should be set to 8.1 or higher to use nonpersistent queues.

### Nonpersistent Queue Notification

For nonpersistent queues, the message is delivered as part of the notification. **Table A–1** shows the actions performed for nonpersistent queues for different notification mechanisms when RAW presentation is specified. **Table A–2** shows the actions performed when XML presentation is specified.

<table>
<thead>
<tr>
<th>Queue Payload Type</th>
<th>OCI Callback</th>
<th>E-mail</th>
<th>PL/SQL Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>OCI callback receives the RAW data in the payload.</td>
<td>Not supported</td>
<td>PL/SQL callback receives the RAW data in the payload.</td>
</tr>
<tr>
<td>Oracle object type</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue Payload Type</th>
<th>OCI Callback</th>
<th>E-mail</th>
<th>PL/SQL Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>OCI callback receives the XML data in the payload.</td>
<td>XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.</td>
<td>PL/SQL callback receives the XML data in the payload.</td>
</tr>
<tr>
<td>Oracle object type</td>
<td>OCI callback receives the XML data in the payload.</td>
<td>XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.</td>
<td>PL/SQL callback receives the XML data in the payload.</td>
</tr>
</tbody>
</table>

### Restrictions on Nonpersistent Queues

You can create nonpersistent queues of RAW and Oracle object type. You are limited to sending messages only to subscribers and explicitly specified recipients who are local.
Propagation is not supported from nonpersistent queues. When retrieving messages, you cannot use the dequeue call, but must instead employ the asynchronous notification mechanism, registering for the notification by mean of OCISubscriptionRegister.

The visibility attribute of enqueue_options must be set to IMMEDIATE for nonpersistent messages.

See Also: "Enqueue Options" on page 10-2
A list of error messages is provided to aid you in troubleshooting problems.

**JMS Error Messages**

**JMS-101 Invalid delivery mode (string)**
- **Cause:** The delivery mode is not supported
- **Action:** The valid delivery mode is AQjmsConstants.PERSISTENT

**JMS-102 Feature not supported (string)**
- **Cause:** This feature is not supported in the current release
- **Action:** Self-explanatory

**JMS-104 Message Payload must be specified**
- **Cause:** The message payload was null
- **Action:** Specify a non-null payload for the message

**JMS-105 Agent must be specified**
- **Cause:** AQjmsAgent object was null
- **Action:** Specify a valid AQjmsAgent representing the remote subscriber

**JMS-106 Cannot have more than one open Session on a JMSConnection**
- **Cause:** There is already one open JMS session on the connection. Cannot have more than one open session on a connection
- **Action:** Close the open session and then open a new one

**JMS-107 Operation not allowed on (string)**
- **Cause:** The specified operation is not allowed on this object
- **Action:** Self-explanatory

**JMS-108 Messages of type (string) not allowed with Destinations containing payload of type (string)**
- **Cause:** There was a mismatch between the message type being used and the payload type specified for the destination
- **Action:** Use the message type that maps to the payload specified for the queue table that contains this destination

**JMS-109 Class not found: (string)**
- **Cause:** The specified class was not found
**JMS Error Messages**

**Action:** Make sure your CLASSPATH contains the class

**JMS-110 Property (string) not writable**
- **Cause:** An attempt was made to update a read-only message header field or property
- **Action:** Self-explanatory

**JMS-111 Connection must be specified**
- **Cause:** The connection object was null
- **Action:** Specify a non-null JDBC connection

**JMS-112 Connection is invalid**
- **Cause:** The JDBC connection is invalid
- **Action:** Specify a non-null oracle JDBC connection

**JMS-113 Connection is in stopped state**
- **Cause:** An attempt was made to receive messages on a connection that is in stopped state
- **Action:** Start the connection

**JMS-114 Connection is closed**
- **Cause:** An attempt was made to use a Connection that has been closed
- **Action:** Create a new connection

**JMS-115 Consumer is closed**
- **Cause:** An attempt was mode to use a Consumer that has been closed
- **Action:** Create a new Message Consumer

**JMS-116 Subscriber name must be specified**
- **Cause:** Subscriber name was null
- **Action:** Specify a non-null subscription name

**JMS-117 Conversion failed - invalid property type**
- **Cause:** An error occurred while converting the property to the requested type
- **Action:** Use the method corresponding to the property data type to retrieve it

**JMS-119 Invalid Property value**
- **Cause:** The property value specified is invalid
- **Action:** Use the appropriate type of value for the property being set

**JMS-120 Dequeue failed**
- **Cause:** An error occurred while receiving the message
- **Action:** See message inside the JMSException and linked SQLException for more information

**JMS-121 DestinationProperty must be specified**
- **Cause:** A null AQjmsDestinationProperty was specified while creating a queue/topic
- **Action:** Specify a non-null AQjmsDestinationProperty for the destination

**JMS-122 Internal error (string)**
- **Cause:** Internal error occurred
Action: Call Support

JMS-123 Interval must be at least (integer) seconds
Cause: An invalid interval was specified
Action: The interval must be greater than 30 seconds

JMS-124 Invalid Dequeue mode
Cause: Invalid dequeue mode was specified
Action: Valid Dequeue modes are AQConstants.DEQUEUE_BROWSE,
AQConstants.DEQUEUE_REMOVE, AQConstants.DEQUEUE_LOCKED,
AQConstants.DEQUEUE_REMOVE_NODATA

JMS-125 Invalid Queue specified
Cause: An invalid Queue object was specified
Action: Specify a valid Queue handle

JMS-126 Invalid Topic specified
Cause: An invalid Topic object was specified
Action: Specify a valid Topic handle

JMS-127 Invalid Destination
Cause: An invalid destination object was specified
Action: Specify a valid destination (Queue/Topic) object

JMS-128 Invalid Navigation mode
Cause: An invalid navigation mode was specified
Action: The valid navigation modes are AQjmsConstants.NAVIGATION_FIRST_MESSAGE,
AQjmsConstants.NAVIGATION_NEXT_MESSAGE,
AQjmsConstants.NAVIGATION_NEXT_TRANSACTION

JMS-129 Invalid Payload type
Cause: There was a mismatch between the message type being used and the
payload type specified for the destination
Action: Use the message type that maps to the payload specified for the queue
    table that contains this destination. For ADT messages, use the appropriate
    CustomDatum factory to create the message consumer

JMS-130 JMS queue cannot be multi-consumer enabled
Cause: An attempt was made to get a AQ multi-consumer queue as a JMS queue
Action: JMS queues cannot be multi-consumer enabled

JMS-131 Session is closed
Cause: An attempt was made to use a session that has been closed
Action: Open a new session

JMS-132 Maximum number of properties (integer) exceeded
Cause: Maximum number of user defined properties for the message has been exceeded
Action: Self-explanatory

JMS-133 Message must be specified
Cause: Message specified was null
JMS Error Messages

**Action**: Specify a non-null message

**JMS-134 Name must be specified**

**Cause**: Queue or Queue table Name specified was null

**Action**: Specify a non-null name

**JMS-135 Driver (string) not supported**

**Cause**: The specified driver is not supported

**Action**: Valid drivers are oci8 and thin. To use the kprb driver get the kprb connection using getDefaultConnection() and use the static createTopicConnection and createQueueConnection methods

**JMS-136 Payload factory can only be specified for destinations with ADT payloads**

**Cause**: A CustomDatumFactory was specified for consumers on destinations not containing ADT payloads

**Action**: This field must be set to null for destinations containing payloads of type SYS.AQ$_JMS_TEXT_MESSAGE, SYS.AQ$_JMS_BYTES_MESSAGE, SYS.AQ$_JMS_MAP_MESSAGE, SYS.AQ$_JMS_OBJECT_MESSAGE, SYS.AQ$_JMS_STREAM_MESSAGE

**JMS-137 Payload factory must be specified for destinations with ADT payloads**

**Cause**: CustomDatumFactory was not specified for destinations containing ADT payloads

**Action**: For destinations containing ADT messages, a CustomDatumFactory for a java class that maps to the SQL ADT type of the destination must be specified

**JMS-138 Producer is closed**

**Cause**: An attempt was made to use a producer that has been closed

**Action**: Create a new Message Producer

**JMS-139 Property name must be specified**

**Cause**: Property name was null

**Action**: Specify a non-null property name

**JMS-140 Invalid System property**

**Cause**: Invalid system property name specified.

**Action**: Specify one of the valid JMS system properties

**JMS-142 JMS topic must be created in multi-consumer enabled queue tables**

**Cause**: An attempt was made to create a JMS topic in a single-consumer queue table

**Action**: JMS topics can only be created in queue tables that are multi-consumer enabled

**JMS-143 Queue must be specified**

**Cause**: Null queue was specified

**Action**: Specify a non-null queue

**JMS-144 JMS queue cannot be created in multi-consumer enabled queue tables**

**Cause**: An attempt was made to create a JMS queue in a multi-consumer queue table
Action: JMS queues can only be created in queue tables that are not multi-consumer enabled

JMS-145 Invalid recipient list
Cause: The recipient list specified was empty
Action: Specify a recipient list with at least one recipient

JMS-146 Registration failed
Cause: An error occurred while registering the type in the type map
Action: Self-explanatory

JMS-147 Invalid ReplyTo destination type
Cause: The ReplyTo destination object type is invalid
Action: The ReplyTo destination must be of type AQjmsAgent

JMS-148 Property name size exceeded
Cause: The property name is greater than the maximum size
Action: Specify a property name that is less than 100 characters

JMS-149 Subscriber must be specified
Cause: Subscriber specified was null
Action: Specify a non-null subscriber

JMS-150 Property not supported
Cause: An attempt was made to use a property that is not supported
Action: Self-explanatory

JMS-151 Topics cannot be of type EXCEPTION
Cause: Topics cannot be of type AQjmsConstants.EXCEPTION
Action: Specify topics to be of type AQjmsConstants.NORMAL

JMS-153 Invalid System property type
Cause: The type of the value specified does not match the type defined for the system property being set
Action: Use the correct type for the setting the system property

JMS-154 Invalid value for sequence deviation
Cause: The sequence deviation is invalid
Action: Valid values are AQEnqueueOption.DEVIATION_BEFORE, AQEnqueueOption.DEVIATION_TOP

JMS-155 AQ Exception (string)
Cause: An error occurred in the AQ java layer
Action: See the message inside the JMSException and the linked exception for more information

JMS-156 Invalid Class (string)
Cause: Class specified is invalid
Action: Make sure your CLASSPATH has the specified class

JMS-157 IO Exception (string)
Cause: IO exception
**JMS-158 SQL Exception (string)**
*Cause:* SQL Exception
*Action:* See message inside linked SQLException for details

**JMS-159 Invalid selector (string)**
*Cause:* The selector specified is either invalid or too long
*Action:* Check the syntax of the selector

**JMS-160 EOF Exception (string)**
*Cause:* EOF exception occurred while reading the byte stream
*Action:* Self-explanatory

**JMS-161 MessageFormat Exception (string)**
*Cause:* An error occurred while converting the stream data to specified type
*Action:* Check the type of data expected on the stream and use the appropriate read method

**JMS-162 Message not Readable**
*Cause:* Message is in write-only mode
*Action:* Call the reset method to make the message readable

**JMS-163 Message not Writeable**
*Cause:* Message is in read-only mode
*Action:* Use the clearBody method to make the message writeable

**JMS-164 No such element**
*Cause:* Element with specified name was not found in the map message
*Action:* Self-explanatory

**JMS-165 Maximum size of property value exceeded**
*Cause:* The property value exceeded the maximum length allowed
*Action:* Values for JMS defined properties can be a maximum of length of 100, Values for User defined properties can have a maximum length of 2000

**JMS-166 Topic must be specified**
*Cause:* Topic specified was null
*Action:* Specify a non-null topic

**JMS-167 Payload factory or Sql_data_class must be specified**
*Cause:* Payload factory or Sql_data_class not specified for queues containing object payloads
*Action:* Specify a CustomDatumFactory or the SQLData class of the java object that maps to the ADT type defined for the queue.

**JMS-168 Cannot specify both payload factory and sql_data_class**
*Cause:* Both CustomDatumFactory and SQLData class were specified during dequeue
*Action:* Specify either the CustomDatumFactory or the SQLData class of the java object that maps to the ADT type defined for the queue.
JMS-169 Sql_data_class cannot be null
  
  **Cause:** SQLData class specified is null

  **Action:** Specify the SQLData class that maps to the ADT type defined for the queue

JMS-171 Message is not defined to contain (string)

  **Cause:** Invalid payload type in message

  **Action:** Check if the queue is defined to contain RAW or OBJECT payloads and use the appropriate payload type in the message

JMS-172 More than one queue table matches query (string)

  **Cause:** More than one queue table matches the query

  **Action:** Specify both owner and queue table name

JMS-173 Queue Table (string) not found

  **Cause:** The specified queue table was not found

  **Action:** Specify a valid queue table

JMS-174 Class must be specified for queues with object payloads. Use dequeue(deq_option,payload_fact) or dequeue(deq_option, sql_data_cl)

  **Cause:** This dequeue method cannot be used to dequeue from queues with OBJECT payloads

  **Action:** Use the either dequeue(deq_option, payload_fact) or dequeue(deq_option, sql_data_cl)

JMS-175 DequeueOption must be specified

  **Cause:** DequeueOption specified is null

  **Action:** Specify a non-null dequeue option

JMS-176 EnqueueOption must be specified

  **Cause:** EnqueueOption specified is null

  **Action:** Specify a non-null enqueue option

JMS-177 Invalid payload type: Use dequeue(deq_option) for raw payload queues

  **Cause:** This method cannot be used to dequeue from queues with RAW payload

  **Action:** Use the dequeue(deq_option) method

JMS-178 Invalid Queue name - (string)

  **Cause:** The queue name specified is null or invalid

  **Action:** Specify a queue name that is not null. The queue name must not be qualified with the schema name. The schema name must be specified as the value of the owner parameter

JMS-179 Invalid Queue Table name - (string)

  **Cause:** The queue table name specified is null or invalid

  **Action:** Specify a queue table name that is not null. The queue table name must not be qualified with the schema name. The schema name must be specified as the value of the owner parameter

JMS-180 Invalid Queue Type

  **Cause:** Queue type is invalid
JMS Error Messages

**JMS-181 Invalid value for wait_time**

*Cause:* Invalid value for wait type

*Action:* Wait time can be AQDequeueOption.WAIT_FOREVER, AQDequeueOption.WAIT_NONE or any value greater than 0

**JMS-182 More than one queue matches query**

*Cause:* More than one queue matches query

*Action:* Specify both the owner and name of the queue

**JMS-183 No AQ driver registered**

*Cause:* No AQDriver registered

*Action:* Make sure that the AQ java driver is registered. Use Class.forName("oracle.AQ.AQOracleDriver")

**JMS-184 Queue object is invalid**

*Cause:* The queue object is invalid

*Action:* The underlying JDBC connection may have been closed. Get the queue handle again

**JMS-185 QueueProperty must be specified**

*Cause:* AQQueueProperty specified is null

*Action:* Specify a non-null AQQueueProperty

**JMS-186 QueueTableProperty must be specified**

*Cause:* QueueTableProperty specified is null

*Action:* Specify a non-null AQQueueTableProperty

**JMS-187 Queue Table must be specified**

*Cause:* Queue Table specified is null

*Action:* Specify a non-null queue table

**JMS-188 QueueTable object is invalid**

*Cause:* The queue table object is invalid

*Action:* The underlying JDBC connection may have been closed. Get the queue table handle again

**JMS-189 Byte array too small**

*Cause:* The byte array given is too small to hold the data requested

*Action:* Specify a byte array that is large enough to hold the data requested or reduce the length requested

**JMS-190 Queue (string) not found**

*Cause:* The specified queue was not found

*Action:* Specify a valid queue

**JMS-191 sql_data_cl must be a class that implements SQLData interface**

*Cause:* The class specified does not support the java.sql.SQLData interface

*Action:* Self-explanatory

**JMS-192 Invalid Visibility value**
Cause: Visibility value specified is invalid
Action: Valid values are AQConstants.VISIBILITY_ONCOMMIT, AQConstants.VISIBILITY_IMMEDIATE

JMS-193 JMS queues cannot contain payload of type RAW
Cause: An attempt was made to create a JMS queue with RAW payload
Action: JMS queues/topics cannot contain RAW payload

JMS-194 Session object is invalid
Cause: Session object is invalid
Action: The underlying JDBC connection may have been closed. Create a new session

JMS-195 Invalid object type: object must implement CustomDatum or SQLData interface
Cause: Invalid object type specified
Action: Object must implement CustomDatum or SQLData interface

JMS-196 Cannot have more than one open QueueBrowser for the same destination on a JMS Session
Cause: There is already an open QueueBrowser for this queue on this session
Action: There cannot be more than one queue browser for the same queue in a particular session. Close the existing QueueBrowser and then open a new one

JMS-197 Agent address must be specified for remote subscriber
Cause: Address field is null for remote subscriber
Action: The address field must contain the fully qualified name of the remote topic

JMS-198 Invalid operation: Privileged message listener set for the Session
Cause: The client tried to use a message consumer to receive messages when the session message listener was set.
Action: Use the session's message listener to consume messages. The consumer's methods for receiving messages must not be used.

JMS-199 Registration for notification failed
Cause: Listener Registration failed
Action: See error message in linked Exception for details

JMS-200 Destination must be specified
Cause: Destination is null
Action: Specify a non-null destination

JMS-201 All Recipients in recipient_list must be specified
Cause: One or more elements in the recipient list are null
Action: All AQjmsAgents in the recipient list must be specified

JMS-202 Unregister for asynchronous receipt of messages failed
Cause: An error occurred while removing the registration of the consumer with the database for asynchronous receipt
Action: Check error message in linked exception for details
JMS-203Payload Factory must be specified
   Cause: Null Payload Factory was specified
   Action: Specify a non null payload factory

JMS-204An error occurred in the AQ JNI layer
   Cause: JNI Error
   Action: Check error message in linked exception for details

JMS-205Naming Exception
   Cause: Naming exception
   Action: Check error message in linked exception for details

JMS-206XA Exception XAError-{0} :: OracleError-{1}
   Cause: An error occurred in the XA layer
   Action: See the message inside the linked XAException for more information

JMS-207JMS Exception {0}
   Cause: An error occurred in the JMS layer
   Action: See the message inside the linked JMSException for more information

JMS-208XML SQL Exception
   Cause: An error occurred in the XML SQL layer
   Action: See the message inside the linked AQxmlException for more information

JMS-209XML SAX Exception
   Cause: An error occurred in the XML SAX layer
   Action: See the message inside the linked AQxmlException for more information

JMS-210XML Parse Exception
   Cause: An error occurred in the XML Parser layer
   Action: See the message inside the linked AQxmlException for more information

JMS-220Connection no longer available
   Cause: Connection to the database no longer available.
   Action: Comment: This may happen if the database/network/machine is not accessible. This may be a transient failure.

JMS-221Free physical database connection unavailable in connection pool
   Cause: A free physical database connection was not available in the OCI connection pool in order to perform the specified operation.
   Action: Try performing the operation later

AQ XML Servlet Error Messages

JMS-400Destination name must be specified
   Cause: A null Destination name was specified
   Action: Specify a non-null destination name

JMS-402Class not found: {0}
   Cause: The specified class was not found
Action: Make sure your CLASSPATH contains the class specified in the error message

JMS-403 IO Exception {0}
Cause: IO exception
Action: See the message inside the linked AQxmlException for more information

JMS-404 XML Parse Exception
Cause: An error occurred in the XML Parser layer
Action: See the message inside the linked AQxmlException for more information

JMS-405 XML SAX Exception
Cause: An error occurred in the XML SAX layer
Action: See the message inside the linked AQxmlException for more information

JMS-406 JMS Exception {0}
Cause: An error occurred in the JMS layer
Action: See the message inside the linked JMSException for more information

JMS-407 Operation not allowed on {0}
Cause: The specified operation is not allowed on this object
Action: Check that the user performing the operation has the required privileges

JMS-408 Conversion failed - invalid property type
Cause: An error occurred while converting the property to the requested type
Action: Use the method corresponding to the property data type to retrieve it

JMS-409 No such element
Cause: Element with specified name was not found in the map message
Action: Specify a valid element name

JMS-410 XML SQL Exception
Cause: An error occurred in the JDBC SQL layer
Action: See the message inside the linked SQLException for more information

JMS-411 Payload body cannot be null
Cause: An invalid body string or document was specified
Action: Specify a non-null body string or document for the payload

JMS-412 Byte conversion failed
Cause: An invalid username/password was specified
Action: Specify a non-null username and password

JMS-413 Autocommit not allowed for operation
Cause: The autocommit flag cannot be set for this operation
Action: Do not set the autocommit flag

JMS-414 Destination owner must be specified
Cause: A null Destination owner was specified
Action: Specify a non-null destination name
JMS-415 Invalid Visibility value

**Cause:** Visibility value specified is invalid

**Action:** Valid values are AQxmlConstants.VISIBILITY_ONCOMMIT, AQxmlConstants.VISIBILITY_IMMEDIATE

JMS-416 Invalid Dequeue mode

**Cause:** Invalid dequeue mode was specified

**Action:** Valid Dequeue modes are AQxmlConstants.DEQUEUE_BROWSE, AQxmlConstants.DEQUEUE_REMOVE, AQxmlConstants.DEQUEUE_LOCKED, AQxmlConstants.DEQUEUE_REMOVE_NODATA

JMS-417 Invalid Navigation mode

**Cause:** An invalid navigation mode was specified

**Action:** The valid navigation modes are:

- AQxmlConstants.NAVIGATION_FIRST_MESSAGE
- AQxmlConstants.NAVIGATION_NEXT_MESSAGE
- AQxmlConstants.NAVIGATION_NEXT_TRANSACTION

JMS-418 Invalid value for wait_time

**Cause:** Invalid value for wait type

**Action:** Wait time can be AQDequeueOption.WAIT_FOREVER, AQDequeueOption.WAIT_NONE, or any value greater than 0

JMS-419 Invalid ConnectionPoolDataSource

**Cause:** A null or invalid ConnectionPoolDataSource was specified

**Action:** Specify a valid OracleConnectionPoolDataSource object with the correct URL and user/password

JMS-420 Invalid value for cache_size

**Cause:** An invalid cache_size was specified

**Action:** Cache size must be greater than 0

JMS-421 Invalid value for cache_scheme

**Cause:** An invalid cache scheme was specified

**Action:** The valid cache schemes are:

- OracleConnectionCacheImpl
- .DYNAMIC_SCHEME
- OracleConnectionCacheImpl.FIXED_WAIT_SCHEME

JMS-422 Invalid tag - {0}

**Cause:** An invalid tag was encountered in the XML document

**Action:** Verify that the XML document conforms to the AQ schema

JMS-423 Invalid value

**Cause:** An invalid value was specified

**Action:** Verify that the value specified in the XML document conforms to those specified in the AQ schema

JMS-424 Invalid message header
Cause: The message header specified is null or invalid
Action: Specify a valid message header

JMS-425 Property name must be specified
Cause: Property name was null
Action: Specify a non-null property name

JMS-426 Property does not exist
Cause: Invalid property name specified. The property does not exist
Action: The property does not exist

JMS-427 Subscriber name must be specified
Cause: Subscriber name was null
Action: Specify a non-null subscription name

JMS-428 Valid message must be specified
Cause: Message was null
Action: Specify a non-null message

JMS-429 Register Option must be specified
Cause: Register option is null
Action: Specify a non-null Register Option

JMS-430 Database Link must be specified
Cause: DB Link is null
Action: Specify a non-null Register Option

JMS-431 Sequence Number must be specified
Cause: Register option is null
Action: Specify a non-null Register Option

JMS-432 Status must be specified
Cause: Status option is null
Action: Specify a non-null Register Option

JMS-433 User not authenticated
Cause: User is not authenticated
Action: Check that the user was authenticated by the webserver before connecting to the Servlet

JMS-434 Invalid data source
Cause: Data source is null or invalid
Action: Specify a valid data source for connecting to the database

JMS-435 Invalid schema location
Cause: Schema location is null or invalid
Action: Specify a valid URL for the schema

JMS-436 AQ Exception
Cause: An error occurred in the AQ java layer
Action: See the message inside the AQxmlException and the linked exception for more information

JMS-437 Invalid Destination
  Cause: An invalid destination object was specified
  Action: Specify a valid destination (Queue/Topic) object

JMS-438 AQ agent {0} not mapped to a valid database user
  Cause: The AQ agent specified does not map to a database user which has privileges to perform the requested operation
  Action: Use DBMS_AQADM.ENABLE_DB_ACCESS to map the agent to a database user with the required queue privileges

JMS-439 Invalid schema document
  Cause: The schema document specified is not valid
  Action: Specify a valid URL for the schema document

JMS-440 Invalid operations - agent {0} maps to more than one database user
  Cause: The AQ agent mapped to more than one database user in the same session
  Action: Map the AQ agent to only one database user. Check the AQ$internet_users view for database users that map to this agent.
ADT
Abstract data type.

API
See application programming interface.

application programming interface
The calling conventions by which an application program accesses operating system and other services.

approximate CSCN
An approximate system change number value, based on the current SCN of the database when a transaction that has enqueued messages into a commit-time queue is committed.

asynchronous
A process in a multitasking system is asynchronous if its execution can proceed independently in the background. Other processes can be started before the asynchronous process has finished. The opposite of synchronous.

BFILE
An external binary file that exists outside the database tablespaces residing in the operating system.

binary large object
A large object datatype whose content consists of binary data. This data is considered raw, because its structure is not recognized by the database.

BLOB
See binary large object.

broadcast
A publish/subscribe mode in which the message producer does not know the identity of any message consumer. This mode is similar to a radio or television station.

buffered queue
Buffered queues support queuing of messages with buffered attributes (buffered messages) and materialize them in memory. If the memory devoted to a buffered message is required for a newer message, or if a buffered message has exceeded a stipulated duration in memory, then that buffered message is swapped to the
underlying queue table. The memory for buffered messages comes from a separate pool called the streams pool. Buffered messages cannot be recovered if the database is bounced. Messages that have no buffered attributes set are queued as persistent messages in the underlying persistent queue.

**canonical**
The usual or standard state or manner of something.

**character large object**
The *large object* datatype whose value is composed of character data corresponding to the database character set. A character large object can be indexed and searched by the Oracle Text search engine.

**ConnectionFactory**
A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a Java Message Service provider.

**CLOB**
See *character large object*.

**commit-time queue**
A queue in which messages are ordered by their approximate CSCN values.

**consumer**
A user or application that can dequeue messages.

**data manipulation language**
Data manipulation language (DML) statements manipulate database data. For example, querying, inserting, updating, and deleting rows of a table are all DML operations; locking a table or view and examining the execution plan of an SQL statement are also DML operations.

**Database Configuration Assistant**
An Oracle Database tool for creating and deleting databases and for managing database templates.

**DBCA**
See Database Configuration Assistant.

**dequeue**
To retrieve a *message* from a queue

**DML**
See *data manipulation language*.

**enqueue**
To place a *message* in a queue. The JMS equivalent of enqueue is *send*.

**exception queue**
Messages are transferred to an exception *queue* if they cannot be retrieved and processed for some reason.
IDAP
See Internet Data Access Presentation.

index-organized table
Unlike an ordinary table whose data is stored as an unordered collection, data for an index-organized table is stored in a B-tree index structure sorted on a primary key. Besides storing the primary key column values of an index-organized table row, each index entry in the B-tree stores the nonkey column values as well.

Internet Data Access Presentation
The Simple Object Access Protocol (SOAP) specification for Oracle® Database operations. IDAP defines the XML message structure for the body of the SOAP request. An IDAP-structured message is transmitted over the Internet using HTTP(S).

Inter-process Communication
Exchange of data between one process and another, either within the same computer or over a network. It implies a protocol that guarantees a response to a request.

IOT
See index-organized table.

IPC
See Inter-process Communication.

Java Database Connectivity
An industry-standard Java interface for connecting to a relational database from a Java program, defined by Sun Microsystems.

Java Message Service
A messaging standard defined by Sun Microsystems, Oracle, IBM, and other vendors. JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.

Java Naming and Directory Interface
A programming interface from Sun for connecting Java programs to naming and directory services.

Java Virtual Machine
The Java interpreter that converts the compiled Java bytecode into the machine language of the platform and runs it. JVMs can run on a client, in a browser, in a middle tier, on an intranet, on an application server such as Oracle Application Server 10g, or in a database server such as Oracle Database 10g.

JDBC
See Java Database Connectivity.

JDBC driver
The vendor-specific layer of Java Database Connectivity that provides access to a particular database. Oracle Database provides three JDBC drivers--Thin, OCI, and KPRB.

JMS
See Java Message Service.
**JMS connection**
An active connection of a client to its JMS provider, typically an open TCP/IP socket (or a set of open sockets) between a client and a provider's service daemon.

**JMS message**
JMS messages consist of a header, one or more optional properties, and a message payload.

**JMS session**
A single threaded context for producing and consuming messages.

**JMS topic**
Equivalent to a multiconsumer queue in the other Oracle® Database interfaces.

**JNDI**
See [Java Naming and Directory Interface](#).

**Jnnn**
Job queue process

**JServer**
The Java Virtual Machine that runs within the memory space of Oracle Database.

**JVM**
See [Java Virtual Machine](#).

**large object**
The class of SQL datatype consisting of BFILE, BLOB, CLOB, and NCLOB objects.

**LDAP**
See [Lightweight Directory Access Protocol](#).

**Lightweight Directory Access Protocol**
A standard, extensible directory access protocol. It is a common language that LDAP clients and servers use to communicate. The framework of design conventions supporting industry-standard directory products, such as the Oracle Internet Directory.

**LOB**
See [large object](#).

**local consumer**
A local consumer dequeues the message from the same queue into which the producer enqueued the message.

**logical change record**
An object with a specific format that describes a database change, captured from the redo log by a capture process or user application. Capture processes enqueue messages containing logical change records (LCRs) only into ANYDATA queues. For improved performance, these LCRs are always stored in a buffered queue.
**message**
The smallest unit of information inserted into and retrieved from a **queue**. A message consists of control information (metadata) and payload (data).

**multicast**
A **publish/subscribe** mode in which the **message producer** knows the identity of each **consumer**. This mode is also known as point-to-multipoint.

**national character large object**
The **large object** datatype whose value is composed of character data corresponding to the database national character set.

**NCLOB**
See **national character large object**.

**nonpersistent**
Nonpersistent queues store messages in memory. They are generally used to provide an **asynchronous** mechanism to send notifications to all users that are currently connected. Nonpersistent queues are deprecated in Oracle® Database 10g Release 2 (10.2). Oracle recommends that you use buffered messaging instead.

**nontransactional**
Allowing enqueuing and dequeuing of only one **message** at a time.

**object type**
An object type encapsulates a data structure along with the functions and procedures needed to manipulate the data. When you define an object type using the **CREATE TYPE** statement, you create an abstract template that corresponds to a real-world object.

**OCI**
See **Oracle Call Interface**.

**OJMS**
See **Oracle Java Message Service**.

**OLTP**
See **Online Transaction Processing**.

**Online Transaction Processing**
Online transaction processing systems are optimized for fast and reliable transaction handling. Compared to data warehouse systems, most OLTP interactions involve a relatively small number of rows, but a larger group of tables.

**OO4O**
See **Oracle Objects for OLE**.

**Oracle Call Interface**
An application programming interface that enables data and **schema** manipulation in Oracle Database.
Oracle Java Message Service

Oracle Java Message Service (OJMS) provides a Java API for Oracle® Database based on the Java Message Service (JMS) standard. OJMS supports the standard JMS interfaces and has extensions to support the Oracle® Database administrative operations and other Oracle® Database features that are not a part of the standard.

Oracle Objects for OLE

A custom control (OCX or ActiveX) combined with an object linking and embedding (OLE) in-process server that lets you plug native Oracle Database functionality into your Windows applications.

producer

A user or application that can enqueue messages.

propagation

Copying messages from one queue to another (local or remote) queue.

publish/subscribe

A type of messaging in which a producer enqueues a message to one or more multiconsumer queues, and then the message is dequeued by several subscribers. The published message can have a wide dissemination mode called broadcast or a more narrowly aimed mode called multicast.

QMNC

Queue monitor coordinator. It dynamically spawns slaves qXXX depending on the system load. The slaves do various background tasks.

QMNN

Queue monitor process.

queue

The abstract storage unit used by a messaging system to store messages.

queue table

A database table where queues are stored. Each queue table contains a default exception queue.

recipient

An agent authorized by the enqueuer or queue administrator to retrieve messages. The enqueuer can explicitly specify the consumers who can retrieve the message as recipients of the message. A queue administrator can specify a default list of recipients who can retrieve messages from a queue. A recipient specified in the default list is known as a subscriber. If a message is enqueued without specifying the recipients, then the message is sent to all the subscribers. Specific messages in a queue can be directed toward specific recipients, who may or may not be subscribers to the queue, thereby overriding the subscriber list.

If only the name of the recipient is specified, then the recipient must dequeue the message from the queue in which the message was enqueued. If the name and an address of the recipient are specified with a protocol value of 0, then the address should be the name of another queue in the same database or another installation of Oracle Database. If the recipient's name is NULL, then the message is propagated to the specified queue in the address and can be dequeued by any subscriber of the queue specified in the address. If the protocol field is nonzero, then the name and address are
not interpreted by the system, and the message can be dequeued by a special consumer.

**remote consumer**
A remote consumer dequeues from a queue that is different from the queue where the message was enqueued.

**rules**
Boolean expressions that define subscriber interest in subscribing to messages. The expressions use syntax similar to the WHERE clause of a SQL query and can include conditions on: message properties (currently priority and correlation identifier), user data properties (object payloads only), and functions. If a rule associated with a subscriber evaluates to TRUE for a message, then the message is sent to that subscriber even if the message does not have a specified recipient.

**rules engine**
Oracle Database software that evaluates rules. Rules are database objects that enable a client to perform an action when an event occurs and a condition is satisfied. Rules are similar to conditions in WHERE clauses of SQL queries. Both user-created applications and Oracle Database features, such as Oracle® Database, can be clients of the rules engine.

**schema**
A collection of database objects, including logical structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. A schema has the name of the user who controls it.

**send**
The JMS equivalent of enqueue.

**servlet**
A Java program that runs as part of a network service and responds to requests from clients. It is typically an HTTP server.

**SGA**
See System Global Area.

**Simple Object Access Protocol**
A minimal set of conventions for invoking code using XML over HTTP defined by World Wide Web Consortium.

**SOAP**

**subscriber**
An agent authorized by a queue administrator to retrieve messages from a queue.

**System Global Area**
A group of shared memory structures that contain data and control information for one Oracle Database instance. The SGA and Oracle Database processes constitute an Oracle Database instance. Oracle Database automatically allocates memory for an SGA whenever you start an instance and the operating system reclaims the memory when you shut down the instance. Each instance has one and only one SGA.
**synchronous**
Two or more processes are synchronous if they depend upon the occurrences of specific events such as common timing signals. The opposite of asynchronous.

**transactional**
Allowing simultaneous enqueuing or dequeuing of multiple messages as part of a group.

**transformation**
A mapping from one Oracle data type to another, represented by a SQL function that takes the source data type as input and returns an object of the target data type. A transformation can be specified during enqueue, to transform the message to the correct type before inserting it into the queue. It can be specified during dequeue to receive the message in the wanted format. If specified with a remote consumer, then the message is transformed before propagating it to the destination queue.

**user queue**
A queue for normal message processing.

**VARRAY**
An ordered set of data elements. All elements of a given array are of the same datatype. Each element has an index, which is a number corresponding to the element's position in the array. The number of elements in an array is the size of the array. Oracle Database allows arrays to be of variable size.

**wildcard**
A special character or character sequence which matches any character in a string comparison.

**workflow**
The set of relationships between all the activities in a project or business transaction, from start to finish. Activities are related by different types of trigger relations. Activities can be triggered by external events or by other activities.
Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G)V$BUFFERED_PUBLISHERS</td>
<td>All Buffered Publishers in the Instance, 9-9</td>
</tr>
<tr>
<td>(G)V$BUFFERED_SUBSCRIBERS</td>
<td>Subscribers for All Buffered Queues in the Instance, 9-9</td>
</tr>
<tr>
<td>(G)V$PERSISTENT_PUBLISHERS</td>
<td>All Active Publishers of the Persistent Queues in the Database, 9-9</td>
</tr>
<tr>
<td>(G)V$PERSISTENT_QMN_CACHE</td>
<td>Performance Statistics on Background Tasks for Persistent Queues, 9-9</td>
</tr>
<tr>
<td>(G)V$PERSISTENT_QUEUES</td>
<td>All Active Persistent Queues in the Database, 9-9</td>
</tr>
<tr>
<td>(G)V$PERSISTENT_SUBSCRIBERS</td>
<td>All Active Subscribers of the Persistent Queues in the Database, 9-9</td>
</tr>
<tr>
<td>(G)V$PROPAGATION_RECEIVER</td>
<td>Buffer Queue Propagation Schedules on the Receiving (Destination) Side, 9-10</td>
</tr>
<tr>
<td>(G)V$PROPAGATION_SENDER</td>
<td>Buffer Queue Propagation Schedules on the Sending (Source) Side, 9-10</td>
</tr>
<tr>
<td>(G)V$QUEUEMETRIC</td>
<td>Queue Metrics for the Most Recent Interval, 9-11</td>
</tr>
<tr>
<td>(G)V$QUEUEMETRIC_HISTORY</td>
<td>Queue Metrics Over Past Hour, 9-11</td>
</tr>
<tr>
<td>(G)V$STREAMMETRIC</td>
<td>Streams Metrics for the Most Recent Interval, 9-10</td>
</tr>
<tr>
<td>(G)V$STREAMMETRIC_HISTORY</td>
<td>Streams Metrics Over Past Hour, 9-11</td>
</tr>
<tr>
<td>(G)V$SUBSCR_REGISTRATION_STATS</td>
<td>Diagnosability of Notifications, 9-10</td>
</tr>
</tbody>
</table>

A

access
  object types, 4-3
access control
  destination level in JMS, 11-8
  queue-level, 1-2
  system level, 1-2
  in JMS, 11-7
  adding subscribers, 8-20
administration
  Messaging Gateway, 17-3
administrative interfaces to Oracle Streams AQ
  comparison, 3-6
AdtMessage
  about, 11-13
  creating, 15-4
agent user
  creating Messaging Gateway agent, 18-4
agents
  AQ.jms agent
    creating, 15-8
  Messaging Gateway
    about, 17-4
    configuring agent, 19-1
    monitoring, 21-3
    running agent on RAC, 19-5
    shutting down agent, 19-3
    starting agent, 19-3
alias
  adding to LDAP server, 8-30
  deleting from LDAP server, 8-30
parameters
  alias, 8-30
  obj_location, 8-30
ALL_QUEUE_SUBSCRIBERS, 9-7
ALL_QUEUE_TABLES
  Queue Tables Queue Accessible to the Current User, 9-3
ALL_QUEUES, 9-3
altering
  AQ agents, 8-29
  propagations, 8-26
  queue tables, 8-8
  queues, 8-15
  subscribers, 8-22
  transformations, 8-17
ANYDATA datatype
  apply process, configuring, 23-7
dequing
  examples, 22-8, 23-16
  explicit, configuring, 23-11
  using JMS, 22-4
  using OCI, 22-4
  using PL/SQL, 22-4
enqueuing
examples, 22-6, 23-14
procedures, creating, 23-4
using JMS, 22-3
using OCI, 22-3
using PL/SQL, 22-2
message propagation, 22-5
propagation examples, 22-9, 22-12
queue table, 8-5
queues
  about, 22-1
  creating, 23-2
wrapper for messages, 22-2
ANYDATA.ConvertObject, 22-2
application development
  about, 1-5
  client/server communication, 1-6
  Internet operations, 1-5
  publish/subscribe, 1-10
  third-party messaging, 1-5
  workflows, 1-9
apply process
  configuring, 23-7
query for applied messages, 23-16
AQ agents
  adding to LDAP server, 10-24
  altering, 8-29
  creating, 8-29
  dropping, 8-29
parameters
  agent_name, 8-29
  certificate_location, 8-29
  enable_anyp, 8-29
  enable_http, 8-29
removing from LDAP server, 10-25
AQ Message Properties Type, 2-5
AQ servlet
  deploying, 6-4
  responses using HTTP, 6-3
AQ system privilege
  granting, 8-18
  in JMS, 12-13
  revoking, 8-19
  in JMS, 12-13
AQ$AGENT, 2-2
AQ$AGENT_LIST_T, 2-3
AQ$POST_INFO_LIST, 2-3
AQ$QUEUE_TABLE_NAME_D, 8-5
AQ$QUEUE_TABLE_NAME_E, 8-4
AQ$QUEUE_TABLE_NAME_H, 8-4
AQ$QUEUE_TABLE_NAME_I, 8-4
AQ$QUEUE_TABLE_NAME_P, 8-5
AQ$QUEUE_TABLE_NAME_S, 8-4
AQ$QUEUE_TABLE_NAME_T, 8-4
AQ$RECIPIENT_LIST_T, 2-3
AQ$REG_INFO_LIST, 2-3
AQ$SUBSCRIBER_LIST_T, 2-3
AQ$INTERNET_USERS, 9-8
AQ$QUEUE_TABLE_NAME, 9-4
AQ$QUEUE_TABLE_NAME_R, 9-7
AQ$QUEUE_TABLE_NAME_S, 9-6
AQ_ADMINISTRATOR_ROLE
  and LDAP, 11-3
  and registerConnectionFactory, 12-3
AQ_MsgProperties, 19-26
AQ_TM_PROCESSES parameter, 2-8
AQ_USER_ROLE
  definition, 4-2
  needed for JMS, 3-5
  security, 4-2
AQXmlPublish method, 6-9
AQXmlReceive method, 6-11
AQXmlSend method, 6-9
architecture
  application development, 1-5
  Internet operations, 1-37, 6-1
Messaging Gateway, 17-3
arrays
  dequeuing
    about, 1-22
    buffered messages, 10-21
    demonstration, 1-38
    syntax, 10-20
  enqueuing
    about, 1-19
    demonstration, 1-38
    syntax, 10-10
asynchronous notifications
  about, 1-16
  buffered messages, 1-18
  designated port, 1-17
  purge following, 1-18
  RAW payload delivery, 1-16
  reliability, 1-17
  timeout, 1-17
asynchronous receive in JMS, 11-25

B
batch dequeuing, 10-20
batch enqueuing, 10-10
BFIELD objects
  propagating, 4-9
Boolean message property (JMS)
  getting, 15-14
  setting, 15-5
broadcasting
  definition, 1-11
buffered messages
  about, 1-12
  dequeuing, 1-14
  options, 1-14
  enqueuing, 1-13
exception handling, 1-16
flow control, 1-15
listen_delivery_mode, 10-12
MSG_STATE parameter, 9-4
notification, 1-18
ordering, 1-14
propagation, 1-14
queue-to-queue propagation, 1-15
restrictions, 1-16
tuning, 5-4
types supported, 1-14
views, 1-14
visibility, 1-14
with Messaging Gateway, 17-5
with Oracle JMS, 11-15
with Real Application Clusters, 1-15
buffered queues, 8-5
byte message property (JMS)
  getting, 15-15
  setting, 15-6
BytesMessage
  about, 11-12
  creating, 15-3
  example, 16-5

C
catx1cr.sql, 22-3
CLASSPATH
  Messaging Gateway, 18-10
closing
  JMS Connection, 15-16
  JMS Session, 15-16
  message consumer, 15-16
  MessageProducer, 15-16
commits
  all operations in JMS Session, 15-2
  transaction, 6-13
commit-time ordering
  about, 1-19
  example, 8-7
  requirements, 8-4
compatibility
  about, 4-1
  and Real Application Clusters, 1-5
  migrating queue tables, 8-11
  nonpersistent queues, A-2
  security, 4-3
concurrent processes
  tuning for Oracle Streams AQ, 5-3
Connection (JMS)
  creating
    with default ConnectionFactory
    parameters, 13-2, 14-2
    with username/password, 13-1, 14-2
  getting JDBC connection from JMS Session, 15-2
ConnectionFactory
  getting
    in LDAP, 12-8
  objects, 11-2
  registering
    through database using JDBC connection
    parameters, 12-1
through database using JDBC URL, 12-2
through LDAP using JDBC connection
  parameters, 12-3
through LDAP using JNDI, 12-4
unregistering
  in LDAP through LDAP, 12-5
  in LDAP through the database, 12-5
  through database, 12-5
  through LDAP, 12-5
using JNDI to look up, 11-2
correlation identifier
  about, 1-19, 1-21
  and transaction grouping, 1-23
  and Virtual Private Database, 4-8
  as dequeue condition, 10-14
  as message property, 10-4
  as MessageSelector, 11-17
  dequeuing by specifying, 1-21
  getting in JMS, 15-13
  setting in JMS, 15-4
  with queue table indexes, 5-3
creating
  AQ agents, 8-29
  AQ.jms agent, 15-8
  DurableSubscriber, 14-8, 14-9, 14-10, 14-11
  JMS AdtMessage, 15-4
  JMS BytesMessage, 15-3
  JMS Connection, 13-1, 13-2, 14-2
  JMS MapMessage, 15-3
  JMS Message, 15-4
  JMS ObjectMessage, 15-3
  JMS Session, 13-3, 14-3
  JMS StreamMessage, 15-3
  JMS TextMessage, 15-3
  Messaging Gateway administration user, 18-4
  Messaging Gateway agent user, 18-4
  Messaging Gateway propagation subscriber, 19-16
nonpersistent queues, A-1
point-to-point queue in JMS, 12-10
propagations, 8-24
publish/subscribe Topic in JMS, 12-11
queue tables, 8-1
  in JMS, 12-9
  QueueBrowser, 13-6, 13-7
  QueueConnection, 13-2, 13-3
  QueueReceiver, 13-9
  queues, 8-12
  in JMS, 12-10
  QueueSender, 13-4
  QueueSession, 13-4
  subscribers, 8-20
  TIB/Rendezvous link, 19-11
  TopicConnection, 14-2, 14-3
Index-4

TopicPublisher, 14-4
TopicSession, 14-4
transformations, 8-17
WebSphere MQ base Java link, 19-6
WebSphere MQ JMS link, 19-7

data pump, 4-6
database
  disabling access, 8-30
  enabling access, 8-29
database connection
  configuring Messaging Gateway connection information, 18-5
  Messaging Gateway, 19-2
DBA_ATTRIBUTE_TRANSFORMATIONS, 9-7
DBA_HIST_QUEUEMETRIC
  Queue Metric History, 9-12
DBA_HIST_STREAMSMETRIC
  Streams Metric History, 9-11
DBA_QUEUE_SCHEDULES, 9-3
DBA_QUEUE_SUBSCRIBERS, 9-7
DBA_QUEUE_TABLES
  All Queue Tables in Database, 9-2
DBA_QUEUES, 9-3
DBA_SUBSCR_REGISTRATIONS
  All Subscription Registrations, 9-8
DBA_TRANSFORMATIONS, 9-7
DBMS_AQ procedures
  BIND_AGENT, 10-24
  DEQUEUE, 10-13
  DEQUEUE_ARRAY, 10-20
  ENQUEUE, 10-2
  ENQUEUE_ARRAY, 10-10
  LISTEN, 10-11
  POST, 10-23
  REGISTER, 10-22
  UNBIND_AGENT, 10-25
  UNREGISTER, 10-23
DBMS_AQADM procedures
  ADD_ALIAS_TO_LDAP, 8-30
  ADD_SUBSCRIBER, 8-20
  ALTER_AQ_AGENT, 8-29
  ALTER_PROPAGATION_SCHEDULE, 8-26
  ALTER_QUEUE, 8-15
  ALTER_QUEUE_TABLE, 8-8
  ALTER_SUBSCRIBER, 8-22
  CREATE_AQ_AGENT, 8-29
  CREATE_NP_QUEUE, A-1
  CREATE_QUEUE, 8-12
  CREATE_QUEUE_TABLE, 8-1
  CREATE_TRANSFORMATION, 8-17
  DEL_ALIAS_FROM_LDAP, 8-30
  DISABLE_DB_ACCESS, 8-30
  DISABLE_PROPAGATION_SCHEDULE, 8-27
  DROP_AQ_AGENT, 8-29
  DROP_QUEUE, 8-16
  DROP_QUEUE_TABLE, 8-9
  DROP_TRANSFORMATION, 8-18
  ENABLE_DB_ACCESS, 8-29
  ENABLE_JMS_TYPES, 22-3
  ENABLE_PROPAGATION_SCHEDULE, 8-27
  GRANT_QUEUE_PRIVILEGE, 8-19
  GRANT_SYSTEM_PRIVILEGE, 8-18
  MIGRATE_QUEUE_TABLE, 8-11
  MODIFY_TRANSFORMATION, 8-17
  PURGE_QUEUE_TABLE, 8-9
  REMOVE_SUBSCRIBER, 8-23
  REVOKE_QUEUE_PRIVILEGE, 8-20
  REVOKE_SYSTEM_PRIVILEGE, 8-19
  SCHEDULE_PROPAGATION, 8-24
  START_QUEUE, 8-15
  STOP_QUEUE, 8-16
  UNSCHEDULE_PROPAGATION, 8-28
  VERIFY_QUEUE_TYPES, 1-32, 8-26
DBMS_AQ.BUFFERED, 10-12
DBMS_AQIN, 12-1
DBMS_AQ.PERSISTENT, 10-12
DBMS_AQ.PERSISTENT OR BUFFERED, 10-12
DBMS_MGWADM package
  about, 17-3
  ADD_SUBSCRIBER, 19-16, 19-26
  ALTER_AGENT, 19-3
  ALTER_MSGSYSTEM_LINK, 19-11, 19-23
  ALTER_SUBSCRIBER, 19-26
  CREATE_MSGSYSTEM_LINK, 19-6, 19-7, 19-11, 19-23
  DISABLE_PROPAGATION_SCHEDULE, 19-16
  DOMAIN_QUEUE, 19-13
  DOMAIN_TOPIC, 19-13
  ENABLE_PROPAGATION_SCHEDULE, 19-16
  JMS_CONNECTION, 19-7
  JMS_QUEUE_CONNECTION, 19-7
  JMS_TOPIC_CONNECTION, 19-7
  MQSERIES_BASE_JAVA_INTERFACE, 19-6
  REGISTER_FOREIGN_QUEUE, 19-13, 19-25
  REMOVE_MSGSYSTEM_LINK, 19-12
  RESET_SUBSCRIBER, 19-17
  SHUTDOWN, 19-3
  STARTUP, 19-3
  UNREGISTER_FOREIGN_QUEUE, 19-14
  DB_CONNECT_INFO, 18-5, 19-2
  DBMS_MGWMSG.LCR_TO_XML, 20-4
  DBMS_MGWMSG.LCR_TO_XML, 20-4
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE, 23-2
  DBMS_STREAMS_ADM.SET_UP_QUEUE, 22-3, 23-2
  DBMS_STREAMS_ADM.SET_UP_QUEUE, 22-3, 23-2
  DBMS_TRANSFORM.CREATE_TRANSFORMATION, 22-11
delays
during dequeuing, 1-24
specifying in JMS, 11-22
demonstrations
  about, 1-38
  Oracle Streams AQ, 1-38
  Oracle Streams AQ JMS, 1-39
  Oracle Streams AQ XML, 1-39
dequeue condition
  and Virtual Private Database, 4-8

Index-4
with queue table indexes, 5-3
dequeuing
ANYDATA queues
   examples, 22-8, 23-16, 23-18
   using JMS, 22-4
   using OCI, 22-4
   using PL/SQL, 22-4
buffered messages, 1-14
by multiple consumers, 1-7
concurrent processes, 1-21
demonstration, 1-38
features, 1-21
IDAP client request, 6-11
IDAP server response to request, 6-14
message arrays, 1-22, 10-20
message states, 1-23
messages, 10-13
methods, 1-21
modes
   about, 1-22
   navigation of messages, 1-23
options, 10-13
   buffered messages, 1-14
parameters
   array_size, 10-20
   dequeue_options, 10-13, 10-20
retries with delays, 1-24
transaction protection, 1-25
waiting for messages, 1-24
destination (JMS)
   altering, 12-17
   dropping, 12-17
   starting, 12-16
   stopping, 12-16
disabling
   database access, 8-30
   propagations, 8-27
double message property (JMS)
   getting, 15-14
   setting, 15-6
dropping
   AQ agents, 8-29
   propagations, 8-28
   queue tables, 8-9
   queues, 8-16
   transformations, 8-18
DurableSubscriber
   about, 11-18
   creating
      for JMS Topic, 14-8, 14-9
      for Oracle object type Topic, 14-10, 14-11
   unsubscribing
      for a local subscriber, 14-16
      for a remote subscriber, 14-17
database access, 8-29
propagations, 8-27
enqueueing
ANYDATA queues
   examples, 22-6-23-14, 23-18
   procedures, creating, 23-4
   using JMS, 22-3
   using OCI, 22-3
   using PL/SQL, 22-2
buffered messages, 1-13
client request for, 6-9
correlation identifier, 1-13
demonstration, 1-38
features, 1-19
IDAP client request, 6-9
IDAP server response to request, 6-14
message array, 1-19, 10-20
message expiration, 1-21
message grouping, 1-20
message properties, 10-3
messages, 10-2
options, 10-2
parameters
   array_size, 10-10
   dequeue_options, 10-2
   message_properties, 10-2
   message_properties_array, 10-10
   payload, 10-2
   payload_array, 10-10
   priority and ordering of messages, 1-19
   sender identification, 1-20
enumerated constants
   about, 2-6
delay, 2-7
delivery_mode, 2-7
dequeue mode, 2-7
expiration, 2-7
message_grouping, 2-7
namespace, 2-7
navigation, 2-7
operational interface, 2-7
queue_type, 2-7
retention, 2-7
state, 2-7
visibility, 2-7
wait, 2-7
environment variables
   CLASSPATH, 18-10
   Messaging Gateway, 18-10
   MGW_PRE_PATH, 18-10
   ORACLE_SID, 18-10
error conditions
   Messaging Gateway, 21-7
error handling
   error messages, 7-2
   IDAP, 6-15
   propagations, 1-32
error messages, 7-2
   Messaging Gateway, 21-4
   Messaging Gateway agent, 21-9
E
   e-mail notification, 6-16
demonstration, 1-38
enabling
errors (JMS)
  getting codes, 15-17
  getting number, 15-17
exception (JMS)
  exception linked to a JMS exception, getting, 15-17
  exception listener
  getting, 15-18
  setting, 15-17
  printing stack trace, 15-17
exception handling
  buffered messages, 1-16
  exception queues, 1-25, 11-25
  Messaging Gateway, 21-3
  propagations in JMS, 11-28
exception queues
  about, 1-25
  in JMS, 11-25
expiration
  setting during enqueuing, 1-21
  specifying in JMS, 11-22
exporting
  queue tables
  about, 4-4
  data pump, 4-6
  modes, 4-5
  multiple recipients, 4-4
flow control
  about, 1-15

F

float message property (JMS)
  getting, 15-15
  setting, 15-6

G

getting (JMS)
  ConnectionFactory, 12-6
  correlation identifier, 15-13
  error codes, 15-17
  error numbers, 15-17
  exceptions, 15-17
  JDBC connection, 15-2
  JMS Connection, 15-2
  message identifier, 15-13
  OracleOCIConnectionPool, 15-2
  Queue in LDAP, 12-9
  queue table, 12-10
  QueueConnectionFactory, 12-6
  in LDAP, 12-8
  with JDBC connection parameters, 12-7
  with JDBC URL, 12-6
  Topic in LDAP, 12-9
  TopicConnectionFactory, 12-6
  with JDBC connection parameters, 12-8
  with JDBC URL, 12-7
  GLOBAL_AQ_USER_ROLE
  and registerConnectionFactory, 11-3, 12-3
granting
  AQ system privilege, 8-18
  in JMS, 12-13
  queue privilege, 8-19
  in JMS, 12-14
  Topic privilege in JMS, 12-14
  grouping
  messages, 1-20
  GV$AQ, 11-9

H

HTTP
  AQ operations over, 6-1
  AQ servlet responses, 6-3
  client requests, 6-3
  headers, 6-8
  propagation, 6-4
  propagation using, 1-33
  response, 6-9
  transactions, 6-3
  user sessions, 6-3

I

IDAP
  client request
  commit transaction, 6-13
  dequeue, 6-11
  enqueue, 6-9
  registration, 6-13
  roll back transaction, 6-14
  error handling, 6-15
  message, 6-9
  notification, 6-15
  request and response documents, 6-9
  server response
  commit transaction, 6-15
  dequeue request, 6-14
  enqueue request, 6-14
  register request, 6-15
  roll back transaction, 6-15
importing
  queue tables
  about, 4-5
  data pump, 4-6
  IGNORE parameter, 4-5
  multiple recipients, 4-5
inboxes, 1-29
indexes
  tuning for Oracle Streams AQ, 5-3
initialization parameters
  Messaging Gateway, 18-9
  INIT.ORA parameter, 2-7
integer message property (JMS)
  getting, 15-14
  setting, 15-5
interaces to Oracle Streams AQ
  about, 1-38
  administrative, 3-6
  AQ XML servlet, 3-5
Index-7

comparison, 3-1, 3-6
JMS, 3-4
OCI, 3-3
OCI security, 4-3
OCCI, 3-3
OCI, 3-3
OO4O, 3-3
operational, 3-7
PL/SQL, 3-2
Internet Data Access Presentation
about, 6-7
Internet operations
and application development, 1-5
AQ servlet responses, 6-3
architecture, 1-37, 6-1
client requests, 6-3
deploying AQ servlet, 6-4
IDAP client request
commit transaction, 6-13
dequeue, 6-11
equeque, 6-9
registration, 6-13
roll back transaction, 6-14
IDAP errors, 6-15
IDAP notification, 6-15
IDAP request and response documents, 6-9
IDAP server response
commit transaction, 6-15
dequeue request, 6-14
equeque request, 6-14
register request, 6-15
roll back transaction, 6-15
Internet Data Access Presentation, 6-7
JMS types, 6-2
notification by e-mail, 6-16
object type queues, 6-2
payloads, 6-2
propagation, 6-4
RAW queues, 6-2
SOAP
body, 6-7
evelope, 6-7
message structure, 6-7
method invocation, 6-8
transactions, 6-3
user authentication, 6-3
user sessions, 6-3
I/O
configuring for Oracle Streams AQ, 5-3

J

J2EE compliance, 11-29
Java properties
Messaging Gateway, 18-11
oracle.mgw.batch_size, 18-11
oracle.mgw.polling_interval, 18-11
oracle.mgw.tibrv.advMsgCallback, 18-12
oracle.mgw.tibrv.encoding, 18-11
oracle.mgw.tibrv.intraProcAdvSubjects, 18-12
JDBC connection
getting from JMS Session, 15-2
registering ConnectionFactory using JDBC parameters through the database, 12-1
using to register ConnectionFactory through LDAP, 12-3
JDBC OCI driver
needed for JMS, 3-5
JDBC thin driver
needed for JMS, 3-5
JDBC URL
registering ConnectionFactory using JDBC URL through LDAP, 12-4
registering through the database, 12-2
JMS
about, 11-1
and Real Application Clusters, 11-8
ANYDATA messages
dequeueing, 23-18
enqueuing, 23-18
asynchronous receive, 11-25
buffered messages, 11-15
Connection, 11-1
exception queues, 11-25
J2EE compliance, 11-29
JDBC OCI driver needed, 3-5
JDBC thin driver needed, 3-5
message bodies, 11-12
message consumer features, 11-22
message headers, 11-9
message properties, 11-10
message types, 11-9
MessageProducer features, 11-21
point-to-point features, 11-16
propagation schedules, 12-17
publish/subscribe features, 11-17
queue tables
creating, 12-9
going, 12-10
recipient lists, 11-19
Session, 11-1
statistics views support, 11-9
structured payloads, 11-9
troubleshooting, 15-17
JMS Connection
about, 11-3
closing, 15-16
going, 15-2
going OracleOCIConnectionPool from, 15-2
starting, 15-2
stopping, 15-16
JMS correlation identifier
setting, 15-4
JMS Destination
about, 11-6
managing, 12-16
methods, 11-7
using JMS Session to obtain, 11-6
using JNDI to look up, 11-7
JMS examples
BytesMessage, 16-5
MapMessage, 16-15
setting up, 16-1
StreamMessage, 16-10
TextMessage, 16-21
JMS message property
Boolean, 15-5, 15-14
byte, 15-6, 15-15
double, 15-6, 15-14
float, 15-6, 15-15
integer, 15-5, 15-14
long, 15-6, 15-15
object, 15-7, 15-15
short, 15-7, 15-15
string, 15-5, 15-14
JMS messages
browsing, 11-24
with a TopicBrowser, 14-22
correlation identifier, 15-13
creating
AdMessage, 15-4
BytesMessage, 15-3
JMS Message, 15-4
MapMessage, 15-3
ObjectMessage, 15-3
StreamMessage, 15-3
TextMessage, 15-3
delay, specifying, 11-22
expiration, specifying, 11-22
grouping, 11-22
message consumer, closing, 15-16
message identifier, 15-13
message listener
specifying at JMS Session, 15-13
specifying at message consumer, 15-12
message property
getting, 15-13
setting, 15-4
MessageProducer, closing, 15-16
navigating in receive, 11-23
navigation mode for receiving, specifying, 15-11
Priority
setting default, 15-8
priority and ordering, 11-21
propagation with Messaging Gateway
inbound, 20-15
outbound, 20-15
publishing
specifying a recipient list, 14-7
specifying delivery mode, priority, and time to live, 14-6
specifying Topic, 14-5
with minimal specification, 14-4
QueueBrowser for, creating, 13-6, 13-7
QueueReceiver for, creating, 13-9
receiving
about, 11-23
asynchronously, 15-12, 15-13
from a destination using a transformation, 15-10
synchronize, 15-8, 15-10
with a message consumer, 15-8, 15-10
remote subscribers for, creating, 14-13
remove no data, 11-24
retry with delay, 11-24
sending using a QueueSender, 13-4, 13-5
TimeToLive
setting default, 15-7
TopicBrowser for, creating, 14-19, 14-20
TopicReceiver for, creating, 14-17, 14-18
transformation with JMS AQ, 11-29
JMS propagations
about, 11-26
altering, 12-19
disabling, 12-19
enabling, 12-18
exception handling, 11-28
RemoteSubscriber, 11-26
scheduling, 11-26, 12-18
unscheduling, 12-20
JMS publish/subscribe
setting up, 11-20
JMS Session
about, 11-5
closing, 15-16
committing all operations, 15-2
creating, 13-3, 14-3
getting JDBC connection from, 15-2
rolling back all operations, 15-2
specifying message listener, 15-13
using to obtain Destination object, 11-6
JMS type queues/topics, 6-2
JMS types
ANYDATA queues, 22-3
Internet operations, 6-2
JMS_DeliveryMode, 19-26
JMS_NoLocal, 19-27
JNDI
using to look up ConnectionFactory objects, 11-2
using to look up Destination object, 11-7
JOB_QUEUE_PROCESSES, 4-9

L

LDAP
and AQ_ADMINISTRATOR_ROLE, 11-3
queue/topic connection factory, 12-8
registering ConnectionFactory, 12-4
unregistering ConnectionFactory, 12-5
LDAP server
adding alias, 8-30
adding AQ agents, 10-24
deleting alias, 8-30
removing AQ agents, 10-25
links
altering, 19-11
configuring Messaging Gateway links, 19-5
MGW_LINKS view, 19-12
MGW_MQSERIES_LINK view, 19-12
MGW_TIBRV_LINKS view, 19-12
Index-9

removing, 19-12
TIB/Rendezvous, creating, 19-11
WebSphere MQ base Java, creating, 19-6
WebSphere MQ JMS, creating, 19-7

listener.ora
modifying for Messaging Gateway, 18-2, 18-3
modifying for TIB/Rendezvous, 18-7
modifying for WebSphere MQ, 18-7

listening
about, 1-24
application development, 1-11
demonstration, 1-38
parameters
agent_list, 10-11
listen_delivery_mode, 10-11
wait, 10-11
syntax, 10-11

LOBs
propagation, 1-31

log file
Messaging Gateway, 21-1
log_directory, 18-9
log_level, 18-9

logical change records
Messaging Gateway, 20-4

long message property (JMS)
getting, 15-15
setting, 15-6

M

managing
nonpersistent queues, A-2
propagations, 4-8, 8-23
queue tables, 8-1
queues, 8-12
subscribers, 8-20
transformations, 8-16

MapMessage
about, 11-13
creating, 15-3
eaxmple, 16-15

message headers
conversion with Messaging Gateway, 20-2
WebSphere MQ mappings, 20-6

message identifier
about, 1-21
and transaction grouping, 1-23
getting in JMS, 15-13

message properties
TIB/Rendezvous, 20-12
using with message types in JMS, 11-14

message types in JMS
about, 11-9
AdtMessage, 11-13
BytesMessage, 11-12
MapMessage, 11-13
ObjectMessage, 11-13
StreamMessage, 11-12
TextMessage, 11-13

MessageListener, 11-25
MessageProducer
closing, 15-16
features, 11-21, 11-29
setting default Priority, 15-8
setting default TimeToLive, 15-7
messages
array dequeling, 1-22, 10-20
array enqueuing, 1-19, 10-10
bodies in JMS, 11-12
browsing in JMS, 11-24, 14-22
correlation identifier
about, 1-21
correlation identifiers, 1-19
in JMS, 15-13
creating in JMS, 15-3, 15-4
creating remote subscribers in JMS, 14-14
delay, specifying in JMS, 11-22
dequeling
features, 1-21
methods, 1-21
modes, 1-22
syntax, 10-13
using JMS, 23-18
with concurrent processes, 1-21
denqueuing
features, 1-19
options, 10-2
syntax, 10-2
using JMS, 23-18
exception queues, 1-25
expiration
about, 1-21
specifying in JMS, 11-22
format transformations, 1-34
grouping, 1-20
in JMS, 11-22
header conversion with Messaging Gateway, 20-2
headers in JMS, 11-9
history and retention in JMS, 11-8
identifier
about, 1-21
JMS message conversion, 20-14
JMS message property
getting, 15-13
JMS message property, setting, 15-4
message consumer in JMS, closing, 15-16
message identifier in JMS, 15-13
MessageProducer in JMS, closing, 15-16
navigating in JMS, 11-23
navigation during dequeling, 1-23
navigation in receive, 11-23
navigation mode, specifying in JMS, 15-11
non-JMS message conversion, 20-1
nonrepudiation, 1-36
object type support, 1-3
ordering
buffered messages, 1-14
ordering during propagation, 1-29

Index-9
payload restrictions, 4-7
persistence
for security, 1-3
metadata analysis, 1-3
scheduling, 1-3
priority and ordering, 1-19
in JMS, 11-21
priority during propagation, 1-29
Priority, setting in JMS, 14-8
propagation
ANYDATA, 22-5
errors, 1-32
features, 1-27
inboxes and outboxes, 1-29
LOBs, 1-31
remote consumers, 1-28
scheduling, 1-30
statistics, 1-31
using HTTP, 1-33
with RAC, 1-32
properties, 10-3
in JMS, 11-10
publishing in JMS, 14-4, 14-5, 14-6, 14-7
QueueBrowser for, creating, 13-6, 13-7
QueueReceiver for, creating, 13-9
receiving in JMS, 11-23
receiving synchronously in JMS, 15-8, 15-10
recipients
about, 1-8
remote subscribers, creating in JMS, 14-13
remove no data in JMS, 11-24
retention and history, 1-36
retries during dequeueing, 1-24
retry with delay in JMS, 11-24
sender identification, 1-20
sending in JMS, 13-4, 13-5
states during dequeueing, 1-23
third-party propagation support, 1-33
TIB/Rendezvous conversion, 20-10
TimeToLive, setting in JMS, 15-7
TopicBrowser for, creating, 14-19, 14-20, 14-21
TopicReceiver for, creating, 14-17, 14-18
tracking, 1-36
transaction protection, 1-25
transformations, 1-34
in JMS, 11-29
using types with properties in JMS, 11-14
waiting during dequeueing, 1-24
WebSphere MQ conversion, 20-6
XML transformations, 1-34
MessageSelector
about, 11-17
Messaging Gateway
about, 17-1
administration, 17-3
administration user
creating, 18-4
agent
about, 17-4
configuring, 19-1
error messages, 21-9
shutting down, 19-3
starting, 19-3
agent user
creating, 18-4
and JMS, 17-1
and non-Oracle messaging systems, 17-4
architecture, 17-3
buffered messages, 17-5
canonical types, 20-2
database connection, 19-2
database connection information,
configuring, 18-5
environment variables, 18-10
error conditions, 21-7
error messages, 21-4
exception handling, 21-3
features, 17-1
in a RAC environment, 18-5
initialization file, 18-3
about, 18-9
initialization parameters, 18-9
integration with Oracle Database, 17-4
Java properties, 18-11
links
altering, 19-11
loading, 18-1
log file, 21-1
logical change records, 20-4
message conversion (JMS), 20-14
message conversion (non-JMS), 20-1
messaging system links
configuring, 19-5
modifying listener.ora, 18-2, 18-3
monitoring agent status, 21-3
non-Oracle messaging
configuration properties, 19-20
optional link configuration properties, 19-23
non-Oracle messaging queues
configuring, 19-12
non-Oracle queue
unregistering, 19-14
optional foreign queue configuration
properties, 19-25
optional subscriber configuration
properties, 19-26
propagation, 17-4
propagation disabling, 19-16
propagation enabling, 19-16
propagation resetting, 19-17
propagation schedule
removing, 19-17
propagation subscriber
creating, 19-16
removing, 19-17
propagation subscribers, 19-15
propagations, 19-14
monitoring, 21-8
registering non-Oracle queue, 19-13
removing a link, 19-12
Index-10
resource limits, 19-3
running agent on RAC, 19-5
setting up for TIB/Rendezvous, 18-7
setting up for WebSphere MQ, 18-7
setting up third-party messaging, 18-6
setup procedure, 18-1
verifying, 18-8
unloading, 18-8
view for non-Oracle queues, 19-14
views, 21-3
views for links, 19-12
Messaging Gateway user
and MGW_AGENT_ROLE, 18-4
MGW_ADMINISTRATOR_ROLE
and Messaging Gateway administration user, 18-4
creating, 18-2
MGW_AGENT_OPTIONS
Supplemental Options and Properties, 9-14
MGW_AGENT_ROLE, 19-2
and Messaging Gateway user, 18-4
creating, 18-2
MGW_BASIC_MSG_T, 20-2
MGW_FOREIGN_QUEUES, 19-14
Foreign Queues, 9-16
MGW_GATEWAY, 19-3, 21-3
Configuration and Status Information, 9-12
MGW_JOBS
Messaging Gateway Propagation Jobs, 9-16
MGW_LINKS, 19-12
Names and Types of Messaging System Links, 9-14
MGW_MQSERIES_LINK, 19-12
MGW_MQSERIES_LINKS
WebSphere MQ Messaging System Links, 9-14
MGW_PRE_PATH, 18-10
MGW_SCHEDULES
Information about Schedules, 9-18
MGW_SUBSCRIBERS
Information for Subscribers, 9-17
MGW_TIBRV_LINKS, 19-12
TIB/Rendezvous Messaging System Links, 9-15
MGW_TIBRV_MSG_T, 20-2
mgw.ora
about, 18-9
comment lines, 18-12
environment variables, 18-10
Java properties, 18-11
modifying for TIB/Rendezvous, 18-7
modifying for WebSphere MQ, 18-8
parameters, 18-9
setting up, 18-3
migrating
queue tables, 8-11
modifying
listener.ora for Messaging Gateway, 18-2, 18-3
transformations, 8-17
monitoring
Messaging Gateway, 21-1
propagations, 21-8
Messaging Gateway agent status, 21-3
MQ_BrokerControlQueue, 19-23
MQ_BrokerPubQueue, 19-23
MQ_BrokerQueueManager, 19-24
MQ_BrokerVersion, 19-24
MQ_ccsid, 19-24
MQ_CharacterSet, 19-26
MQ_JmsDurSubQueue, 19-24, 19-26
MQ_JmsTargetClient, 19-26
MQ_openOptions, 19-26
MQ_PubAckInterval, 19-24
MQ_ReceiveExit, 19-24
MQ_ReceiveExitInit, 19-24
MQ_SecurityExit, 19-24
MQ_SecurityExitInit, 19-25
MQ_SendExit, 19-25
MQ_SendExitInit, 19-25
MsgBatchSize, 19-27
multicasting
definition, 1-11
multiconsumer dequeuing, 1-7

N

names
queue tables
length, 8-2
mixed case, 8-2
queues
length, 8-13
mixed case, 8-13
navigation
during dequeuing, 1-23
modes
FIRST_MESSAGE, 1-23
NEXT_MESSAGE, 1-23
NEXT_TRANSACTION, 1-23
specifying mode in JMS, 15-11
nonpersistent queues
compatibility, A-2
creating, A-1
managing, A-2
notifications, A-2
restrictions, A-2
nonrepudiation
about, 1-36
notifications
about, 1-16
buffered messages, 1-18
designated port, 1-17
e-mail, 6-16
IDAP, 6-15
nonpersistent queues, A-2
parameters
post_count, 10-23
post_list, 10-23
reg_count, 10-22
reg_list, 10-22
posting, 10-23
purge following, 1-18
RAW payload delivery, 1-16
registering, 10-22
reliability, 1-17
timeout, 1-17
unregistering, 10-23

O
object message property (JMS)
  getting, 15-15
  setting, 15-7
object types
  access, 4-3
  support for, 1-3
  synonyms, 4-8
object_name, 2-1
ObjectMessage
  about, 11-13
  creating, 15-3
OCCI
  interface to Oracle Streams AQ, 3-3
  Oracle type translator, 3-3
OCI
  interface to Oracle Streams AQ, 3-3
  Oracle type translator, 3-3
OO4O
  interface to Oracle Streams AQ, 3-3
operational interfaces to Oracle Streams AQ
  options
    deqing, 10-13
    enqueuing, 10-2
  Oracle AQ Views, 9-1
  Oracle Enterprise Manager
    and Oracle Streams AQ, 1-35
    support for, 4-6
  Oracle Internet Directory
    and Oracle Streams AQ, 1-35
    Oracle Streams AQ integration, 1-4
Oracle JMS
  about, 11-1
  Oracle Messaging Gateway Views, 9-2
  Oracle object (ADT) type queues
    Internet operations, 6-2
Oracle Real Application Clusters
  and JMS, 11-8
  message propagation, 1-32
Oracle type translator, 3-3
ORACLE_SID
  Messaging Gateway, 18-10
  oracle.mgw.batch_size, 18-11
  oracle.mgw.polling_interval, 18-11
  oracle.mgw.tibrv.advMsgCallback, 18-12
  oracle.mgw.tibrv.encoding, 18-11
  oracle.mgw.tibrv.intraProcAdvSubjects, 18-12
OracleOCIConectionPool
  getting from JMS Connection, 15-2
ordering
  commit-time, 1-19
  during propagation, 1-29
messages in JMS, 11-21
specifying during enqueuing, 1-19
outboxes, 1-29

P
parameters
  admin_option, 8-18
  agent_list, 10-11
  agent_name, 8-29
  alias, 8-30
  AQ_TM_PROCESSES, 2-8
  array_size, 10-10, 10-20
  attempts, 10-4
  attribute_number, 8-17
  certificate, 10-24
  certificate_location, 8-29
  comment, 8-2, 8-13
  compatibility, 4-1
  compatible, 8-3
  consumer_name, 10-13
  correlation, 10-4, 10-14
  db_username, 8-29
  delay, 10-3
  delivery_mode, 8-20, 10-3, 10-4, 10-14
  deq_condition, 10-14
  dequeue, 8-15, 8-16
  dequeue_mode, 10-13
  dequeue_options, 10-13, 10-20
  dest_queue_name, 8-26
  destination, 8-24, 8-26
  destination_queue, 8-24
  duration, 8-24
  enable_anyp, 8-29
  enable_http, 8-29
  enqueue, 8-15, 8-16
  enqueue_options, 10-2
  enqueue_time, 10-4
  exception_queue, 10-4
  expiration, 10-4
  from_schema, 8-17
  from_type, 8-17
  grant_option, 8-19
  grantees, 8-18
  latency, 8-24
  listen_delivery_mode, 10-11, 10-12
  log_directory, 18-9
  log_level, 18-9
  max_retries, 8-13
  message_grouping, 8-2
  message_properties, 10-2
  message_properties_array, 10-10, 10-11
  MSG_STATE, 9-4
  msgid, 10-14
  multiple_consumers, 8-2
  name, 10-24
  namespace, 10-24
  navigation, 10-13
  next_time, 8-24
  obj_location, 8-30
payloads
  ANYDATA wrappers for, 22-2
  Internet operations, 6-2
  restrictions, 4-7
  structured, 1-4
  transformations with Messaging Gateway, 20-2
  XMLType, 1-4

performance
  about, 1-2
  buffered messages, 5-4
  concurrent processes, 5-3
  configuring I/O, 5-3
  Oracle Streams AQ and RAC, 5-1
  persistent messaging, 5-1
  propagation tuning, 5-4
  queue table indexes, 5-3
  serial processes, 5-3
  shared servers, 5-2
  storage parameters, 5-2
  persistent messaging

  compared to buffered, 1-12
  performance, 5-1
  tuning, 5-2

point-to-point messages
  about, 11-16
  port
    designated for notification, 1-17
    posting for notification, 10-23
    PreserveMessageID, 19-27
  priority
    during propagation, 1-29
    specifying during enqueuing, 1-19
  Priority (JMS)
    about, 11-21
    setting for all messages from a MessageProducer, 15-8

privileges
  AQ system privilege
    granting, 8-18
    granting in JMS, 12-13
    revoking, 8-19
    revoking in JMS, 12-13
  DBMS_AQIN, 12-1
  parameters
    admin_option, 8-18
    grant_option, 8-19
    grantee, 8-18
  queue privilege
    granting, 8-19
    granting in JMS, 12-14
    revoking, 8-20
    revoking in JMS, 12-15
  required for propagation, 4-8
  security, 4-3
  SELECT_ANY_DICTIONARY, 23-2
  Topic privileges
    granting in JMS, 12-14
    revoking in JMS, 12-14

programmatic interfaces
  about, 1-38
  ANYDATA queues, 22-2
  AQ XML servlet, 3-5
  comparison, 3-1, 3-6
  JMS, 3-4
  OCCI, 3-3
  OCI, 3-3
  OCI security, 4-3
  OO4O, 3-3
  PL/SQL, 3-2

propagations
  about, 1-27
  in JMS, 11-26
  altering, 8-26
  in JMS, 12-19
  ANYDATA queues
    about, 22-5
    examples, 22-9, 22-12
  BFILE objects, 4-9
  buffered messages, 1-14
  creating, 8-24
Index-14

debugging, 7-1
disabling, 8-27
   in JMS, 12-19
   with Messaging Gateway, 19-16
dropping, 8-28
enabling, 8-27
   in JMS, 12-18
   with Messaging Gateway, 19-16
error handling, 1-32
   in JMS, 11-28
features, 1-27
inboxes and outboxes, 1-29
JMS messages with Messaging Gateway
   inbound, 20-15
   outbound, 20-15
managing, 4-8, 8-23
messages with LOBs, 1-31
Messaging Gateway
   configuring for, 19-14
   monitoring, 21-8
   resetting with, 19-17
   subscribers, about, 19-15
   subscribers, creating, 19-16
   subscribers, removing, 19-17
optimizing, 4-9
parameters
   destination, 8-24
   destination_queue, 8-24
duration, 8-24
   latency, 8-24
   next_time, 8-24
   start_time, 8-24
priority and ordering of messages, 1-29
privileges required, 4-8
publishing JMS messages
   specifying a recipient list, 14-7
   specifying delivery mode, priority, and time to live, 14-6
   specifying Topic, 14-5
   with minimal specification, 14-4
publish/subscribe
   about, 1-10
   setting up, 1-11, 11-20
purge
   following notification, 1-18
   purging
   queue tables, 8-9

Queue (JMS)
   getting in LDAP, 12-9
queue monitor coordinator, 1-35
queue privilege
   granting, 8-19
   in JMS, 12-14
   revoking, 8-20
   in JMS, 12-15
queue tables
   altering, 8-8
   creating, 8-1
   in JMS, 12-9
data pump, 4-6
dropping, 8-9
export
   modes, 4-5
   exporting
   about, 4-4
   getting in JMS, 12-10
importing
   about, 4-4, 4-5
   IGNORE parameter, 4-5
   multiple recipients, 4-5
managing, 8-1
migrating, 8-11
multiple recipients
   exporting, 4-4
names
   length, 8-2
   mixed case, 4-1, 8-2
parameters
   comment, 8-2
   compatible, 8-3
message_grouping, 8-2
multiple_consumers, 8-2
primary_instance, 8-2
queue_payload_type, 8-2
queue_table, 8-2
secondary_instance, 8-2

TIB/Rendezvous, 20-12, 20-13
tuning, 5-4
unscheduling, 8-28
   in JMS, 12-20
using HTTP, 1-33, 6-4
using HTTP and HTTPS, 6-4
WebSphere MQ, 20-9
   with Messaging Gateway, 17-4
   with RAC, 1-32
publishing JMS messages
   specifying a recipient list, 14-7
   specifying delivery mode, priority, and time to live, 14-6
   specifying Topic, 14-5
   with minimal specification, 14-4
publish/subscribe
   about, 1-10
   setting up, 1-11, 11-20
purge
   following notification, 1-18
   purging
   queue tables, 8-9

Q

Index-14
recovery
  restrictions, 4-8
REF payloads
  restrictions, 4-7
registerConnectionFactory
  and AQ_ADMINISTRATOR_ROLE, 12-3
  and GLOBAL_AQ_USER_ROLE, 12-3
using JDBC connection parameters through
  LDAP, 12-3
using JDBC connection parameters through the
database, 12-1
using JDBC URL through LDAP, 12-4
registering
  for notification, 10-22
  through the database, JDBC URL, 12-2
registration
  client request for, 6-9
  IDAP client request, 6-13
  IDAP server response to request, 6-15
reliability
  notifications, 1-17
remote consumers
  propagation, 1-28
remote subscribers
  restrictions, 4-7
RemoteSubscriber, 11-19, 11-26
resource limits
  Messaging Gateway, 19-3
restrictions
  buffered messages, 1-16
  message payloads, 4-7
  nonpersistent queues, A-2
  point-in-time recovery, 4-8
  queue management, 4-7
  REF payloads, 4-7
  remote subscribers, 4-7
  subscribers, 4-7
  synonyms, 4-8
  virtual private database, 4-8
retention
  of messages, 1-36
    in JMS, 11-8
retries
  during dequeuing, 1-24
  multiple sessions dequeuing, 1-25
revoking
  AQ system privilege, 8-19
    in JMS, 12-13
  queue privilege, 8-20
    in JMS, 12-15
roles
  AQ_ADMINISTRATOR_ROLE, 3-5, 4-2, 12-3
  AQ_USER_ROLE, 3-5, 4-2
  GLOBAL_AQ_USER_ROLE, 11-3, 12-3
  MGW_ADMINISTRATOR_ROLE, 18-2, 18-4
  MGW_AGENT_ROLE, 18-2, 18-4
  SELECT_CATALOG_ROLE, 23-2
rollback
  all operations in JMS Session, 15-2
  RV_discardAmount, 19-27
RV_limitPolicy, 19-27
RV_maxEvents, 19-27
S
scalability
  about, 1-3
schedules
  enabling and disabling propagation with
    Messaging Gateway, 19-16
scheduling
  about propagation scheduling, 1-30
  propagations using SCHEDULE_PROPAGATION, 8-24
secure queues, 10-1
security, 4-2
  at destination level in JMS, 11-8
  at system level in JMS, 11-7
  compatibility parameter, 4-3
  message persistence, 1-3
  OCI applications, 4-3
  propagations, 4-4
  queue privileges, 4-3
  queue tables
    secure parameter, 8-3
  roles, 4-2
sender identification
  during enqueuing, 1-20
serial processes
  tuning for Oracle Streams AQ, 5-3
Session (JMS)
  creating, 13-3, 14-3
  shared servers
    performance with Oracle Streams AQ, 5-2
short message property (JMS)
  getting, 15-15
  setting, 15-7
SOAP
  ANYDATA queues, 22-5
  body, 6-7
  envelope, 6-7
  header, 6-7
  message structure, 6-7
  method invocation, 6-8
stack trace
  printing in JMS, 15-17
starting
  JMS Connection, 15-2
  Messaging Gateway agent, 19-3
  queues, 8-15
statistics
  propagation, 1-31
stopping
  JMS Connection, 15-16
  queues, 8-16
storage parameters
  tuning Oracle Streams AQ, 5-2
StreamMessage
  about, 11-12
  creating, 15-3
example, 16-10
string message property (JMS)
  getting, 15-14
  setting, 15-5
structured payloads, 1-4
  about, 1-4
  in JMS, 11-9
subscribers
  about, 1-7
  adding, 8-20
  altering, 8-22
  creating, 8-20
creating JMS remote subscriber for Oracle object
  type messages, 14-14
creating remote subscriber for JMS
  messages, 14-13
creating with Messaging Gateway, 19-16
in Messaging Gateway propagations, 19-15
managing, 8-20
names
  mixed case, 4-1
ordering, 1-7
parameters
  delivery_mode, 8-20
  queue_to_queue, 8-20
  rule, 8-20
  transformation, 8-20
removing, 8-23
restrictions, 4-7
specifying transformations for in JMS, 14-12, 14-15
unsubscribing DurableSubscribers, 14-16, 14-17
synonyms
  restrictions, 4-8
  SYS.AQ$_DESCRIPTOR, 2-5
  SYS.AQ$_POST_INFO, 2-6
  SYS.AQ$_REG_INFO, 2-3
  SYS.MGW_MQSERIES_PROPERTIES, 19-20
  SYS.MGW_TIBRV_PROPERTIES, 19-22
system privilege
  granting, 8-18
  in JMS, 12-13
  revoking, 8-19
  in JMS, 12-13

T

TextMessage
  about, 11-13
  creating, 15-3, 15-4
  example, 16-21
third-party messaging
  and application development, 1-5
  and Messaging Gateway, 17-4
configuration properties, 19-20
optional foreign queue configuration properties, 19-25
optional link configuration properties, 19-23
optional subscriber configuration properties, 19-26
queues
  configuring, 19-12
  registering, 19-13
  unregistering, 19-14
  setting up, 18-6
  view for registered queues, 19-14
TIB/Rendezvous
  AQ_MsgProperties, 19-26
  links
    creating, 19-11
    listener.ora, modifying, 18-7
    message conversion, 20-10
    message property mapping, 20-12
    Messaging Gateway. setting up for, 18-7
    mgw.ora, modifying, 18-7
    MsgBatchSize, 19-27
    PreserveMessageID, 19-27
  propagation
    inbound, 20-13
    outbound, 20-12
    RV_discardAmount, 19-27
    RV_limitPolicy, 19-27
    RV_maxEvents, 19-27
  Subject
    registering, 19-14
    unregistering, 19-14
    system properties, 19-22
  time specification
    during enqueuing, 1-21
  timeout
    notifications, 1-17
  TimeToLive
    setting for all messages from a
      MessageProducer, 15-7
Topic
  creating DurableSubscriber for, 14-8, 14-9, 14-10, 14-11
  creating in JMS, 12-11
  getting in LDAP, 12-9
  granting Topic privilege in JMS, 12-14
  revoking Topic privilege in JMS, 12-14
  specifying transformations for subscribers, 14-12
  TopicBrowser, 11-20
    browsing messages using, 14-22
    creating for Topic of Oracle type messages, 14-21
    creating for Topic of standard JMS type messages, 14-19, 14-20
  TopicConnectionFactory
    getting
      in LDAP, 12-8
      with JDBC connection parameters, 12-8
      with JDBC URL, 12-7
    registering
      through database using JDBC connection
      parameters, 12-1
Index-18

through database using JDBC URL, 12-2
through LDAP using JDBC connection parameters, 12-3
through LDAP using JDBC URL, 12-4
unregistering
through database, 12-5
through LDAP, 12-5

TopicPublisher
about, 11-19
creating, 14-4

TopicReceiver
about, 11-19
creating for Topic of Oracle object type messages, 14-18
creating for Topic of standard JMS type messages, 14-17

TopicSession
creating, 14-4

transaction
IDAP client request
commit, 6-13
roll back, 6-14
protection during dequeuing, 1-25

transformations
about, 1-34
altering, 8-17
creating, 8-17
dropping, 8-18
for remote subscribers, specifying in JMS, 14-15
for Topic subscribers, specifying in JMS, 14-17
managing, 8-16
Messaging Gateway, 20-2
modifying, 8-17

parameters
attribute_number, 8-17
from_schema, 8-17
from_type, 8-17
to_schema, 8-17
to_type, 8-17
transformation, 8-17

XML, 1-34

troubleshooting
in JMS, 15-17

buffered messaging support, 1-14
Messaging Gateway, 20-2
MGW_BASIC_MSG_T, 20-2
MGW_TIBRV_MSG_T, 20-2
oracle.xdb.XMLType, 22-4
support for, 1-3
SYS.LCR$_DDL_RECORD, 22-2
SYS.LCR$_ROW_RECORD, 22-2

unregistering
ConnectionFactory in LDAP, 12-5
notification, 10-23
unscheduling
propagations, 8-28
in JMS, 12-20

user authentication
Internet operations, 6-3

USER_ATTRIBUTE_TRANSFORMATIONS
User Transformation Functions, 9-8
USER_QUEUE_SCHEDULES, 9-3
USER_QUEUE_SUBSCRIBERS, 9-7
USER_QUEUE_TABLES, 9-3
USER_QUEUES, 9-3
USER_SUBSCR_REGISTRATIONS
User Subscription Registrations, 9-8
USER_TRANSFORMATIONS, 9-8
users
Messaging Gateway agent, 18-4

views
V$METRICGROUP
Information about the Metric Group, 9-10
verifying
Messaging Gateway setup, 18-8
queue type, 8-26

users
Messaging Gateway agent, 21-3
Messaging Gateway links, 19-12
propagation schedules in user schema, 9-3
queue subscribers, 9-6
queue subscribers and their rules, 9-7
queue subscribers for queues where user has
queue privileges, 9-7
queue subscribers in user schema, 9-7
queue tables in user schema, 9-3
queues for which user has any privilege, 9-3
queues for which user has queue privilege, 9-3
queues in user schema, 9-3
registered non-Oracle queues, 19-14
user transformations, 9-8

Index-19
virtual private database
restrictions, 4-8
visibility
about, 6-10, 6-12
buffered messages, 1-14
dequeue options, 10-13
enqueue options, 10-2
rollback operations, 6-14

W
waiting
during dequeuing, 1-24
WebSphere MQ
base Java link, creating, 19-6
base Java queue
registering, 19-13
unregistering, 19-14
JMS link, creating, 19-7
JMS Queue or Topic
registering, 19-13
unregistering, 19-14
JMS_DeliveryMode, 19-26
JMS_NoLocal, 19-27
listener .ora, modifying, 18-7
message conversion, 20-6
message header mappings, 20-6
Messaging Gateway, setting up for, 18-7
mgw.ora, modifying, 18-8
MQ_BrokerControlQueue, 19-23
MQ_BrokerPubQueue, 19-23
MQ_BrokerQueueManager, 19-24
MQ_BrokerVersion, 19-24
MQ_csid, 19-24
MQ_CharacterSet, 19-26
MQ_JmsDurSubQueue, 19-24, 19-26
MQ_JmsTargetClient, 19-26
MQ_openOptions, 19-26
MQ_PubAckInterval, 19-24
MQ_ReceiveExit, 19-24
MQ_ReceiveExitInit, 19-24
MQ_SecurityExit, 19-24
MQ_SecurityExitInit, 19-25
MQ_SendExit, 19-25
MQ_SendExitInit, 19-25
MsgBatchSize, 19-27
optional link configuration properties, 19-23
PreserveMessageID, 19-27
propagation
inbound, 20-9
outbound, 20-9
system properties, 19-20

X
XA
using with Oracle Streams AQ, 4-6
XML, 6-1
deploying AQ servlet, 6-4
message format transformations, 1-34