Oracle® Streams

Advanced Queuing User's Guide and Reference

Release 10.1

Part No. B10785-01

December 2003

ORACLE®

Oracle Streams Advanced Queuing User's Guide and Reference, Release 10.1

Part No. B10785-01

Primary Authors: Craig B. Foch, Shelley Higgins

Contributing Authors: Bhagat Nainani, Brajesh Goyal, Deanna Bradshaw, Denis Raphaely, Phil Locke, Randy Urbano, Shailendra Mishra, Toliver Jue

Contributors: Bob Thome, Charles Hall, Janet Stern, John Lang, Kapil Surlaker, Kathryn Greunefeldt, Kevin Zewe, Kirk Bittler, Krishnan Meiyyapan, Nancy Ikeda, Neerja Bhatt, Qiang Liu, Shengsong Ni, Sugu Venkatasamy, Vivekananda Maganty

Contents

Send Us Your Comments xxv

Preface xxvii

Intended Audience xxviii
Structure xxviii
Related Documents xxxi
Conventions xxxii
Documentation Accessibility xxxvii

What's New in Oracle Streams AQ? xxxix

Oracle Streams AQ Release 10.1 New Features xl
Oracle9*i* Release 2 (9.2.0) New Features xlii
Oracle9*i* Release 1 (9.0.1) New Features in Oracle Streams AQ xliii
Oracle8*i* New Features in Oracle Streams AQ xlvi

Volume 1

Part I Introducing Oracle Streams AQ

1 Introducing Oracle Streams AQ

Overview of Oracle Streams AQ 1-2
Oracle Streams AQ in Integrated Application Environments 1-4
Oracle Streams AQ Client/Server Communication 1-5
Multiconsumer Dequeuing of the Same Message 1-6

Oracle Streams AQ Implementation of Workflows 1-9
Oracle Streams AQ Implementation of Publish/Subscribe 1-11
Message Propagation 1-13
Message Format Transformation 1-16
Internet Integration and Internet Data Access Presentation 1-16
Internet Message Payloads 1-17
Propagation over the Internet Using HTTP 1-18
Internet Data Access Presentation (IDAP) 1-19
Interfaces to Oracle Streams AQ 1-20
Oracle Streams AQ Features 1-20
Enqueue Features 1-20
Dequeue Features 1-24
Propagation Features 1-26
Other Oracle Streams AQ Features 1-28
Oracle Streams AQ Demos 1-34

2 Getting Started with Oracle Streams AQ

Oracle Streams AQ Prerequisites 2-2
Oracle Streams AQ by Example 2-2
Creating Oracle Streams AQ Queues and Queue Tables 2-3
Enqueuing and Dequeuing Oracle Streams AQ Messages 2-9
Oracle Streams AQ Propagation 2-55
Dropping Oracle Streams AQ Objects 2-59
Revoking Roles and Privileges 2-60
Deploying Oracle Streams AQ with XA 2-60
Oracle Streams AQ and Memory Usage 2-65
Frequently Asked Questions 2-81
Oracle Streams AQ Installation Questions 2-81
General Oracle Streams AQ Questions 2-83
Transformation Questions 2-87

3 Basic Components

Object Name (object_name) 3-2
Type Name (type_name) 3-2
AQ Agent Type (aq$_agent) 3-3

AQ Recipient List Type (aq$_recipient_list_t) 3-4
AQ Agent List Type (aq$_agent_list_t) 3-4
AQ Subscriber List Type (aq$_subscriber_list_t) 3-4
AQ Registration Information List Type (aq$_reg_info_list) 3-5
AQ Post Information List Type (aq$_post_info_list) 3-5
AQ Registration Information Type (aq$_reg_info) 3-5
AQ Notification Descriptor Type 3-7
AQ Post Information Type 3-7
Enumerated Constants in the Oracle Streams AQ Administrative Interface 3-8
Enumerated Constants in the Oracle Streams AQ Operational Interface 3-8
INIT.ORA Parameter File Considerations 3-9
AQ_TM_PROCESSES Parameter No Longer Needed in init.ora 3-9
JOB_QUEUE_PROCESSES Parameter 3-10

4 Oracle Streams AQ: Programmatic Environments

Programmatic Environments for Accessing Oracle Streams AQ 4-2
Using PL/SQL to Access Oracle Streams AQ 4-2
Using OCI to Access Oracle Streams AQ 4-3
Using OCCI to Access Oracle Streams AQ 4-4
Using Visual Basic (OO4O) to Access Oracle Streams AQ 4-4
Using Oracle Java Message Service (OJMS) to Access Oracle Streams AQ 4-5
Accessing Standard and Oracle JMS Applications 4-6
Using Oracle Streams AQ XML Servlet to Access Oracle Streams AQ 4-7
Comparing Oracle Streams AQ Programmatic Environments 4-9
Oracle Streams AQ Administrative Interfaces 4-9
Oracle Streams AQ Operational Interfaces 4-11

Part II Managing and Tuning Oracle Streams AQ

5 Managing Oracle Streams AQ

Oracle Streams AQ Compatibility Parameters 5-2
Queue Security and Access Control 5-2
Oracle Streams AQ Security 5-3
Administrator Role 5-3

User Role 5-3
Access to Oracle Streams AQ Object Types 5-4
Queue Security 5-4
Queue Privileges and Access Control 5-4
OCI Applications and Queue Access 5-5
Security Required for Propagation 5-5
Queue Table Export-Import 5-6
Exporting Queue Table Data 5-6
Importing Queue Table Data 5-7
Data Pump Export and Import 5-8
Creating Oracle Streams AQ Administrators and Users 5-9
Oracle Enterprise Manager Support 5-10
Using Oracle Streams AQ with XA 5-10
Restrictions on Queue Management 5-11
Remote Subscribers 5-12
DML Not Supported on Queue Tables or Associated IOTs 5-12
Propagation from Object Queues with REF Payload Attributes 5-12
Collection Types in Message Payloads 5-12
Synonyms on Queue Tables and Queues 5-12
Tablespace Point-in-Time Recovery 5-13
Nonpersistent Queues 5-13
Managing Propagation 5-13
EXECUTE Privileges Required for Propagation 5-14
The Number of Job Queue Processes 5-14
Optimizing Propagation 5-14
Message States During Client Requests for Enqueue 5-16
Propagation from Object Queues 5-16
Debugging Oracle Streams AQ Propagation Problems 5-16
8.0-Compatible Queues 5-17
Migrating To and From 8.0 5-17
Importing and Exporting with 8.0-Style Queues 5-18
Roles in 8.0 5-18
Security with 8.0-Style Queues 5-19
Access to Oracle Streams AQ Object Types 5-19
OCI Application Access to 8.0-Style Queues 5-19

Pluggable Tablespaces and 8.0-Style Multiconsumer Queues 5-19
Autocommit Features in the DBMS_AQADM Package 5-20

6 Oracle Streams AQ Performance and Scalability

Performance Overview 6-2
Oracle Streams AQ and Oracle Real Application Clusters 6-2
Oracle Streams AQ in a Shared Server Environment 6-2
Basic Tuning Tips 6-2
Using Storage Parameters 6-3
I/O Configuration 6-3
Running Enqueue and Dequeue Processes Concurrently in a Single Queue Table 6-3
Running Enqueue and Dequeue Processes Serially in a Single Queue Table 6-3
Creating Indexes on a Queue Table 6-4
Propagation Tuning Tips 6-4

Part III Oracle Streams AQ: Sample Application

7 Oracle Streams AQ Sample Application

A Sample Application 7-2
General Features of Oracle Streams AQ 7-2
System-Level Access Control 7-3
Queue-Level Access Control 7-5
Message Format Transformation 7-7
Creating Transformations 7-9
Structured Payloads 7-12
Creating Queues with XMLType Payloads 7-15
Nonpersistent Queues 7-18
Retention and Message History 7-28
Publish/Subscribe Support 7-29
Oracle Real Application Clusters Support 7-32
Statistics Views and Oracle Streams AQ 7-37
Internet Access for Oracle Streams AQ 7-37
Enqueue Features 7-38
Subscriptions and Recipient Lists 7-38

Priority and Ordering of Messages 7-40
Time Specification: Delay 7-48
Time Specification: Expiration 7-50
Message Grouping 7-53
Message Transformation During Enqueue 7-56
Enqueue Using the Oracle Streams AQ XML Servlet 7-58
Dequeue Features 7-60
Dequeue Methods 7-61
Multiple Recipients 7-66
Local and Remote Recipients 7-67
Message Navigation in Dequeue 7-68
Modes of Dequeuing 7-73
Optimization of Waiting for Arrival of Messages 7-79
Retry with Delay Interval 7-80
Exception Handling 7-84
Rule-Based Subscription 7-90
Listen Capability 7-94
Message Transformation During Dequeue 7-100
Dequeue Using the Oracle Streams AQ XML Servlet 7-101
Asynchronous Notifications 7-101
Registering for Notifications Using the Oracle Streams AQ XML Servlet 7-110
Propagation Features 7-111
Propagation Overview 7-111
Propagation Scheduling 7-112
Propagation of Messages with LOB Attributes 7-116
Enhanced Propagation Scheduling Capabilities 7-118
Exception Handling During Propagation 7-121
Message Format Transformation During Propagation 7-122
Propagation Using HTTP 7-123

Part IV Oracle Streams AQ Administrative and Operational Interface

8 Oracle Streams AQ Administrative Interface

Managing Queue Tables 8-2
Creating a Queue Table 8-2

Altering a Queue Table 8-7
Dropping a Queue Table 8-8
Purging a Queue Table 8-9
Migrating a Queue Table 8-12
Managing Queues 8-13
Creating a Queue 8-13
Creating a Nonpersistent Queue 8-16
Altering a Queue 8-17
Dropping a Queue 8-18
Starting a Queue 8-19
Stopping a Queue 8-20
Managing Transformations 8-20
Creating a Transformation 8-21
Modifying a Transformation 8-22
Dropping a Transformation 8-22
Granting and Revoking Privileges 8-23
Granting System Oracle Streams AQ Privileges 8-23
Revoking Oracle Streams AQ System Privileges 8-24
Granting Queue Privileges 8-25
Revoking Queue Privileges 8-25
Managing Subscribers 8-26
Adding a Subscriber 8-26
Altering a Subscriber 8-29
Removing a Subscriber 8-30
Managing Propagations 8-31
Scheduling a Queue Propagation 8-32
Unscheduling a Queue Propagation 8-33
Verifying Propagation Queue Type 8-33
Altering a Propagation Schedule 8-35
Enabling a Propagation Schedule 8-35
Disabling a Propagation Schedule 8-36
Managing Oracle Streams AQ Agents 8-37
Creating an Oracle Streams AQ Agent 8-37
Altering an Oracle Streams AQ Agent 8-38
Dropping an Oracle Streams AQ Agent 8-38

Enabling Database Access 8-39
Disabling Database Access 8-39
Adding an Alias to the LDAP Server 8-40
Deleting an Alias from the LDAP Server 8-40

9 Oracle Streams AQ Administrative Interface: Views

All Queue Tables in Database View 9-2
User Queue Tables View 9-3
All Queues in Database View 9-4
All Propagation Schedules View 9-5
Queues for Which User Has Any Privilege View 9-7
Queues for Which User Has Queue Privilege View 9-8
Messages in Queue Table View 9-9
Queue Tables in User Schema View 9-12
Queues In User Schema View 9-13
Propagation Schedules in User Schema View 9-14
Queue Subscribers View 9-16
Queue Subscribers and Their Rules View 9-17
Number of Messages in Different States for the Whole Database View 9-17
Number of Messages in Different States for Specific Instances View 9-18
Oracle Streams AQ Agents Registered for Internet Access View 9-19
All Transformations View 9-19
All Transformation Functions View 9-20
User Transformations View 9-21
User Transformation Functions View 9-21

10 Oracle Streams AQ Operational Interface: Basic Operations

Enqueuing a Message 10-2
Enqueuing a Message and Specifying Options 10-3
Using Secure Queues 10-4
Enqueuing a Message and Specifying Message Properties 10-5
Enqueuing a Message and Specifying Sender ID 10-5
Enqueuing a Message and Adding Payload 10-6
Enqueuing an Array of Messages 10-12
Listening to One or More Queues 10-17

Dequeuing a Message 10-28
Dequeuing a Message from a Single-Consumer Queue and Specifying Options 10-29
Dequeuing a Message from a Multiconsumer Queue and Specifying Options 10-33
Dequeuing an Array of Messages 10-34
Registering for Notification 10-39
Posting for Subscriber Notification 10-46
Adding an Agent to the LDAP Server 10-47
Removing an Agent from the LDAP Server 10-48

Volume 2

Part V Using Oracle JMS and Oracle Streams AQ

11 Creating Oracle Streams AQ Applications Using JMS

General Features of JMS and Oracle JMS 11-2
J2EE Compliance 11-2
JMSPriority 11-3
JMSExpiration 11-3
Durable Subscribers 11-4
JMS Connection and Session 11-4
ConnectionFactory Objects 11-5
Using AQjmsFactory to Obtain ConnectionFactory Objects 11-5
Using JNDI to Look Up ConnectionFactory Objects 11-6
JMS Connection 11-9
JMS Session 11-10
JMS Connection Examples 11-12
JMS Destination 11-14
Using a JMS Session to Obtain Destination Objects 11-14
Using JNDI to Look Up Destination Objects 11-14
JMS Destination Methods 11-14
JMS Destination Examples 11-15
System-Level Access Control in JMS 11-17
Destination-Level Access Control in JMS 11-17
Retention and Message History in JMS 11-18
Supporting Oracle Real Application Clusters in JMS 11-19

Supporting Statistics Views in JMS 11-19
Structured Payload/Message Types in JMS 11-20
JMS Message Headers 11-20
JMS Message Properties 11-22
JMS Message Body 11-24
The AQ$_JMS_MESSAGE Type 11-24
JMS Message Body: Stream Message 11-25
JMS Message Body: Bytes Message 11-25
JMS Message Body: Map Message 11-26
JMS Message Body: Text Message 11-26
JMS Message Body: Object Message 11-27
JMS Message Body: AdtMessage 11-27
JMS Point-to-Point Model Features 11-30
Queues 11-30
QueueSender 11-30
QueueReceiver 11-31
QueueBrowser 11-33
JMS Publish/Subscribe Model Features 11-34
Topic 11-34
Durable Subscriber 11-36
TopicPublisher 11-38
Recipient Lists 11-38
TopicReceiver 11-39
TopicBrowser 11-42
JMS MessageProducer Features 11-43
Priority and Ordering of Messages 11-43
Time Specification - Delay 11-44
Time Specification - Expiration 11-44
Message Grouping 11-45
JMS Message Consumer Features 11-45
Receiving Messages 11-46
Message Navigation in Receive 11-48
Browsing Messages 11-51
Retry with Delay Interval 11-52
Asynchronously Receiving Messages Using Message Listener 11-54

Oracle Streams AQ Exception Handling 11-56
JMS Propagation 11-57
Remote Subscribers 11-57
Scheduling Propagation 11-62
Enhanced Propagation Scheduling Capabilities 11-64
Exception Handling During Propagation 11-65
Message Transformation with JMS AQ 11-66
Defining Message Transformations 11-66
Sending Messages to a Destination Using a Transformation 11-66
Receiving Messages from a Destination Using a Transformation 11-67
Specifying Transformations for Topic Subscribers 11-68
Specifying Transformations for Remote Subscribers 11-69

12 Oracle Streams AQ JMS Interface: Basic Operations

EXECUTE Privilege on DBMS_AQIN 12-2
Registering a Queue/Topic Connection Factory 12-2
Registering Through the Database Using JDBC Connection Parameters 12-2
Registering Through the Database Using a JDBC URL 12-4
Registering Through LDAP Using JDBC Connection Parameters 12-5
Registering Through LDAP Using a JDBC URL 12-7
Unregistering a Queue/Topic Connection Factory 12-8
Unregistering Through the Database 12-8
Unregistering Through LDAP 12-9
Getting a Queue/Topic Connection Factory 12-10
Getting a Queue Connection Factory with JDBC URL 12-11
Getting a Queue Connection Factory with JDBC Connection Parameters 12-11
Getting a Topic Connection Factory with JDBC URL 12-12
Getting a Topic Connection Factory with JDBC Connection Parameters 12-13
Getting a Queue/Topic Connection Factory in LDAP 12-14
Getting a Queue/Topic in LDAP 12-15
Creating a Queue Table 12-16
Getting a Queue Table 12-17
Creating a Queue 12-17
Creating a Point-to-Point Queue 12-18
Creating a Publish/Subscribe Topic 12-18

Granting and Revoking Privileges 12-19
Granting Oracle Streams AQ System Privileges 12-20
Revoking Oracle Streams AQ System Privileges 12-21
Granting Publish/Subscribe Topic Privileges 12-21
Revoking Publish/Subscribe Topic Privileges 12-22
Granting Point-to-Point Queue Privileges 12-23
Revoking Point-to-Point Queue Privileges 12-24
Managing Destinations 12-25
Starting a Destination 12-25
Stopping a Destination 12-26
Altering a Destination 12-27
Dropping a Destination 12-28
Propagation Schedules 12-29
Scheduling a Propagation 12-29
Enabling a Propagation Schedule 12-30
Altering a Propagation Schedule 12-31
Disabling a Propagation Schedule 12-32
Unscheduling a Propagation 12-33

13 Oracle Streams AQ JMS Operational Interface: Point-to-Point

Creating a Connection 13-2
Creating a Connection with Username/Password 13-2
Creating a Connection with Default Connection Factory Parameters 13-2
Creating a Queue Connection 13-3
Creating a Queue Connection with Username/Password 13-3
Creating a Queue Connection with an Open JDBC Connection 13-4
Creating a Queue Connection with Default Connection Factory Parameters 13-5
Creating a Queue Connection with an Open OracleOCIConnection Pool 13-5
Creating a Session 13-6
Creating a QueueSession 13-7
Creating a QueueSender 13-7
Sending Messages 13-8
Sending Messages Using a QueueSender with Default Send Options 13-8
Sending Messages Using a QueueSender by Specifying Send Options 13-9
Creating a QueueBrowser 13-11

Queues with Text, Stream, Objects, Bytes or Map Messages 13-11
Queues with Text, Stream, Objects, Bytes, Map Messages, Locking Messages 13-12
Queues of Oracle Object Type Messages 13-13
Queues of Oracle Object Type Messages, Locking Messages 13-15
Creating a QueueReceiver 13-16
Queues of Standard JMS Type Messages 13-16
Queues of Oracle Object Type Messages 13-17

14 Oracle Streams AQ JMS Operational Interface: Publish/Subscribe

Creating a Connection 14-2
Creating a Connection with Username/Password 14-2
Creating a Connection with Default Connection Factory Parameters 14-2
Creating a TopicConnection 14-3
Creating a TopicConnection with Username/Password 14-3
Creating a TopicConnection with Open JDBC Connection 14-4
Creating a TopicConnection with Default Connection Factory Parameters 14-4
Creating a TopicConnection with an Open OracleOCIConnectionPool 14-5
Creating a Session 14-5
Creating a TopicSession 14-6
Creating a TopicPublisher 14-7
Publishing a Message 14-7
TopicPublisher with Minimal Specification 14-7
TopicPublisher and Specifying Correlation and Delay 14-9
TopicPublisher and Specifying Priority and TimeToLive 14-10
TopicPublisher and Specifying a Recipient List Overriding Topic Subscribers 14-12
Creating a Durable Subscriber 14-13
Creating a Durable Subscriber for a JMS Topic Without Selector 14-14
Creating a Durable Subscriber for a JMS Topic With Selector 14-15
Creating a Durable Subscriber for an Oracle Object Type Topic Without Selector 14-18
Creating a Durable Subscriber for an Oracle Object Type Topic With Selector 14-19
Creating a Remote Subscriber 14-22
Creating a Remote Subscriber for Topics of JMS Messages 14-22
Creating a Remote Subscriber for Topics of Oracle Object Type Messages 14-24
Unsubscribing a Durable Subscription 14-26
Unsubscribing a Durable Subscription for a Local Subscriber 14-26

Unsubscribing a Durable Subscription for a Remote Subscriber 14-27
Creating a TopicReceiver 14-28
Creating a TopicReceiver for a Topic of Standard JMS Type Messages 14-28
Creating a TopicReceiver for a Topic of Oracle Object Type Messages 14-30
Creating a TopicBrowser 14-32
Topics with Text, Stream, Objects, Bytes or Map Messages 14-32
Topics with Text, Stream, Objects, Bytes, Map Messages, Locking Messages 14-33
Topics of Oracle Object Type Messages 14-35
Topics of Oracle Object Type Messages, Locking Messages 14-36
Browsing Messages Using a TopicBrowser 14-38

15 Oracle Streams AQ JMS Operational Interface: Shared Interfaces

Oracle Streams AQ JMS Operational Interface: Shared Interfaces 15-2
AQjmsConnection.start.. 15-2
AQjmsSession.getJmsConnection 15-3
AQjmsSession.commit 15-3
AQjmsSession.rollback 15-3
AQjmsSession.getDBConnection 15-4
AQjmsConnection.getOCIConnectionPool 15-4
AQjmsSession.createBytesMessage 15-5
AQjmsSession.createMapMessage 15-5
AQjmsSession.createStreamMessage.. 15-6
AQjmsSession.createObjectMessage 15-6
AQjmsSession.createTextMessage 15-6
AQjmsSession.createMessage 15-7
AQjmsSession.createAdtMessage 15-7
AQjmsMessage.setJMSCorrelationID... 15-8
Specifying JMS Message Property 15-8
AQjmsMessage.setBooleanProperty 15-9
AQjmsMessage.setStringProperty 15-9
AQjmsMessage.setIntProperty 15-10
AQjmsMessage.setDoubleProperty 15-10
AQjmsMessage.setFloatProperty 15-11
AQjmsMessage.setByteProperty 15-11
AQjmsMessage.setLongProperty 15-12

AQjmsMessage.setShortProperty 15-12
AQjmsMessage.setObjectProperty 15-13
Setting Default TimeToLive for All Messages Sent by a MessageProducer 15-14
Setting Default Priority for All Messages Sent by a MessageProducer 15-14
Creating an AQjms Agent 15-15
Receiving a Message Synchronously 15-16
Using a Message Consumer by Specifying Timeout 15-16
Using a Message Consumer Without Waiting 15-17
Specifying the Navigation Mode for Receiving Messages 15-17
Receiving a Message Asynchronously 15-18
Specifying a Message Listener at the Message Consumer 15-18
Specifying a Message Listener at the Session 15-20
Getting Message ID 15-20
AQjmsMessage.getJMSCorrelationID 15-21
AQjmsMessage.getJMSMessageID 15-21
Getting the JMS Message Properties 15-21
AQjmsMessage.getBooleanProperty 15-22
AQjmsMessage.getStringProperty 15-22
AQjmsMessage.getIntProperty 15-22
AQjmsMessage.getDoubleProperty 15-23
AQjmsMessage.getFloatProperty 15-23
AQjmsMessage.getByteProperty 15-23
AQjmsMessage.getLongProperty 15-24
AQjmsMessage.getShortProperty 15-24
AQjmsMessage.getObjectProperty 15-25
Closing and Shutting Down 15-25
AQjmsProducer.close 15-25
AQjmsConsumer.close 15-26
AQjmsConnection.stop 15-26
AQjmsSession.close 15-26
AQjmsConnection.close 15-26
Troubleshooting 15-27
AQjmsException.getErrorCode 15-27
AQjmsException.getErrorNumber 15-27
AQjmsException.getLinkString 15-28

AQjmsException.printStackTrace 15-28
AQjmsConnection.setExceptionListener 15-28
AQjmsConnection.getExceptionListener 15-29

16 Oracle Streams AQ JMS Types Examples

How to Run the Oracle Streams AQ JMS Type Examples 16-2
Setting Up the Examples 16-2
JMS Bytes Message Examples 16-7
JMS Stream Message Examples 16-12
JMS Map Message Examples 16-19
More Oracle Streams AQ JMS Examples 16-26

Part VI Internet Access to Oracle Streams AQ

17 Internet Access to Oracle Streams AQ

Overview of Oracle Streams AQ Operations over the Internet 17-2
Internet Data Access Presentation (IDAP) 17-2
SOAP Message Structure 17-3
The SOAP Envelope 17-3
SOAP Headers 17-3
The SOAP Body 17-4
SOAP Method Invocation 17-4
HTTP Headers 17-4
Method Invocation Body 17-4
Results from a Method Request 17-5
IDAP Documents 17-6
IDAP Client Requests for Enqueue 17-6
IDAP Client Requests for Dequeue 17-17
IDAP Client Requests for Registration 17-21
IDAP Client Requests to Commit a Transaction 17-22
IDAP Client Requests to Rollback a Transaction 17-23
IDAP Server Response to Enqueue 17-23
IDAP Server Response to a Dequeue Request 17-25
IDAP Server Response to a Register Request 17-28

IDAP Commit Response 17-28
IDAP Rollback Response 17-28
IDAP Notification 17-28
IDAP Response in Case of Error 17-29
SOAP and Oracle Streams AQ XML Schemas 17-30
SOAP Schema 17-30
IDAP Schema 17-32
Deploying the Oracle Streams AQ XML Servlet 17-45
Creating the Oracle Streams AQ XML Servlet Class 17-45
Compiling the Oracle Streams AQ XML Servlet 17-47
Configuring the Web server to Authenticate Users Sending POST Requests 17-48
Using HTTP 17-49
Authorizing Users to Perform Operations with Oracle Streams AQ Servlet 17-49
Registering the Oracle Streams AQ Agent 17-49
Mapping the Oracle Streams AQ Agent to Database Users 17-50
Database Sessions 17-50
Using an LDAP Server with an Oracle Streams AQ XML Servlet 17-52
Using HTTP to Access the Oracle Streams AQ XML Servlet 17-53
User Sessions and Transactions 17-56
Using HTTP and HTTPS for Oracle Streams AQ Propagation 17-57
High-Level Architecture 17-57
Setting Up for HTTP Propagation 17-58
Setting Up for Oracle Streams AQ Propagation over HTTP 17-58
Customizing the Oracle Streams AQ Servlet 17-60
Setting the Connection Pool Size 17-60
Setting the Session Timeout 17-60
Specifying the Style Sheet for All Responses from the Servlet 17-61
Callbacks Before and After Oracle Streams AQ Operations 17-62
Frequently Asked Questions: Using Oracle Streams AQ and the Internet 17-66
Internet Access Questions 17-66
Oracle Internet Directory Questions 17-67

Part VII Using Messaging Gateway

18 Introducing Oracle Messaging Gateway

Introducing Oracle Messaging Gateway 18-2
Oracle Messaging Gateway Features 18-2
Oracle Messaging Gateway Architecture 18-3
Administration Package DBMS_MGWADM 18-4
Oracle Messaging Gateway Agent 18-5
Oracle Database 18-5
Non-Oracle Messaging Systems 18-5
Propagation Processing Overview 18-6

19 Getting Started with Oracle Messaging Gateway

Oracle Messaging Gateway Prerequisites 19-2
Loading and Setting Up Oracle Messaging Gateway 19-2
Loading Database Objects into the Database 19-2
Modifying listener.ora for the External Procedure (Solaris Operating System 32-Bit Only) 19-3
Modifying tnsnames.ora for the External Procedure (Solaris Operating System 32-Bit Only).. 19-4
Setting Up a mgw.ora Initialization File 19-5
Creating an Oracle Messaging Gateway Administration User 19-6
Creating an Oracle Messaging Gateway Agent User 19-6
Configuring Oracle Messaging Gateway Connection Information 19-7
Configuring Oracle Messaging Gateway in a RAC Environment 19-7
Setting Up Non-Oracle Messaging Systems 19-7
Setting Up for TIB/Rendezvous 19-8
Setting Up for WebSphere MQ Base Java or JMS 19-9
Verifying the Oracle Messaging Gateway Setup 19-9
Unloading Oracle Messaging Gateway 19-10
Understanding the mgw.ora Initialization File 19-10
mgw.ora Initialization Parameters 19-11
mgw.ora Environment Variables 19-12
mgw.ora Java Properties 19-13
mgw.ora Comment Lines 19-15

20 Working with Oracle Messaging Gateway

Configuring the Oracle Messaging Gateway Agent 20-2
Database Connection 20-2
Resource Limits 20-2
Starting and Shutting Down the Oracle Messaging Gateway Agent 20-3
Starting the Oracle Messaging Gateway Agent 20-3
Shutting Down the Oracle Messaging Gateway Agent 20-4
Oracle Messaging Gateway Agent Job Queue Job 20-4
Running the Oracle Messaging Gateway Agent on RAC 20-5
Configuring Messaging System Links 20-5
Creating a WebSphere MQ Base Java Link 20-6
Creating a WebSphere MQ JMS Link 20-8
Creating a TIB/Rendezvous Link 20-10
Altering a Messaging System Link 20-10
Removing a Messaging System Link 20-11
Views for Messaging System Links 20-11
Configuring Non-Oracle Messaging System Queues 20-12
Registering a Non-Oracle Queue 20-12
Registering a WebSphere MQ Base Java Queue 20-13
Registering a WebSphere MQ JMS Queue or Topic 20-13
Registering a TIB/Rendezvous Subject 20-14
Unregistering a Non-Oracle Queue 20-14
View for Registered Non-Oracle Queues 20-14
Configuring Oracle Messaging Gateway Propagation Jobs 20-15
Propagation Subscriber Overview 20-15
Creating an Oracle Messaging Gateway Propagation Subscriber 20-16
Creating an Oracle Messaging Gateway Propagation Schedule 20-17
Enabling and Disabling a Propagation Job 20-18
Resetting a Propagation Job 20-19
Altering a Propagation Subscriber and Schedule 20-19
Removing a Propagation Subscriber and Schedule 20-20

21 Oracle Messaging Gateway Message Conversion

Converting Oracle Messaging Gateway Non-JMS Messages 21-2
Overview of the Non-JMS Message Conversion Process 21-2

Oracle Messaging Gateway Canonical Types 21-3
Message Header Conversion 21-3
Handling Arbitrary Payload Types Using Message Transformations 21-3
Handling Logical Change Records 21-6
Message Conversion for WebSphere MQ 21-8
WebSphere MQ Message Header Mappings 21-9
WebSphere MQ Outbound Propagation 21-12
WebSphere MQ Inbound Propagation 21-13
Message Conversion for TIB/Rendezvous 21-14
TIB/Rendezvous Outbound Propagation 21-16
TIB/Rendezvous Inbound Propagation 21-17
JMS Messages 21-18
JMS Outbound Propagation 21-20
JMS Inbound Propagation 21-20

22 Monitoring Oracle Messaging Gateway

The Oracle Messaging Gateway Log File 22-2
Sample Oracle Messaging Gateway Log File 22-2
Interpreting Exception Messages in an Oracle Messaging Gateway Log File 22-3
Monitoring the Oracle Messaging Gateway Agent Status 22-4
The MGW_GATEWAY view 22-4
Oracle Messaging Gateway Irrecoverable Error Messages 22-5
Other Oracle Messaging Gateway Error Conditions 22-9
Monitoring Oracle Messaging Gateway Propagation 22-10
Oracle Messaging Gateway Agent Error Messages 22-11

Part VIII Using Oracle Streams with Oracle Streams AQ

23 Staging and Propagating with Oracle Streams AQ

Oracle Streams Event Staging and Propagation Overview 23-2
SYS.AnyData Queues and User Messages 23-2
SYS.AnyData Wrapper for User Messages Payloads 23-3
Programmatic Environments for Enqueue and Dequeue of User Messages 23-3
Enqueuing User Messages Using PL/SQL 23-4

Enqueuing User Messages Using OCI or JMS 23-4
Dequeuing User Messages Using PL/SQL 23-5
Dequeuing User Messages Using OCI or JMS 23-5
Message Propagation and SYS.AnyData Queues 23-7
User-Defined Type Messages 23-8
Managing an Oracle Streams Messaging Environment 23-9
Wrapping User Message Payloads in a SYS.AnyData Wrapper 23-9
Propagating Messages Between a SYS.AnyData Queue and a Typed Queue 23-14

24 Oracle Streams Messaging Example

Overview of Messaging Example 24-2
Prerequisites 24-3
Set Up Users and Create a SYS.AnyData Queue 24-4
Create the Enqueue Procedures 24-9
Configure an Apply Process 24-13
Configure Explicit Dequeue 24-20
Enqueue Events 24-24
Dequeue Events Explicitly and Query for Applied Events 24-30
Enqueue and Dequeue Events Using JMS 24-31

Part IX Troubleshooting Oracle Streams AQ

25 Troubleshooting Oracle Streams AQ

Debugging Oracle Streams AQ Propagation Problems 25-2
Oracle Streams AQ Error Messages 25-4

A Scripts for Implementing BooksOnLine

tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues, and Subscribers A-2
tkaqdocd.sql: Examples of Administrative and Operational Interfaces A-15
tkaqdoce.sql: Operational Examples A-20
tkaqdocp.sql: Examples of Operational Interfaces A-21
tkaqdocc.sql: Clean-Up Script A-36

Glossary

Index

Send Us Your Comments

Oracle Streams Advanced Queuing User's Guide and Reference, Release 10.1

Part No. B10785-01

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

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

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227. Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

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

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

Preface

This reference 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.

The Preface contains these topics:

- Intended Audience
- Structure
- Related Documents
- Conventions
- Documentation Accessibility

Intended Audience

Oracle Streams Advanced Queuing User's Guide and Reference is intended for programmers who develop applications that use Oracle Streams AQ.

Structure

Part I: Introducing Oracle Streams AQ

Chapter 1, "Introducing Oracle Streams AQ"

This chapter introduces you to Oracle Streams AQ and describes the requirements for optimal messaging systems.

Chapter 2, "Getting Started with Oracle Streams AQ"

This chapter describes the prerequisites for Oracle Streams AQ. It also provides examples of operations using different programmatic environments and answers to several frequently asked questions about Oracle Streams AQ in general.

Chapter 3, "Basic Components"

This chapter describes Oracle Streams AQ features including general, enqueue, and dequeue features.

Chapter 4, "Oracle Streams AQ: Programmatic Environments"

This chapter describes the elements you must work with and issues to consider in preparing your Oracle Streams AQ application environment for different languages.

Part II: Planning, Managing, and Tuning Oracle Streams AQ

Chapter 5, "Managing Oracle Streams AQ"

This chapter discusses issues related to managing Oracle Streams AQ, such as migrating queue tables (import-export), security, Oracle Enterprise Manager support, protocols, sample DBA actions to prepare for working with Oracle Streams AQ, and current restrictions.

Chapter 6, "Oracle Streams AQ Performance and Scalability"

This chapter discusses performance and scalability issues. It included frequently asked questions.

Part III. Oracle Streams AQ: Sample Application

Chapter 7, "Oracle Streams AQ Sample Application"

Part IV. Oracle Streams AQ Administrative and Operational Interface

Chapter 8, "Oracle Streams AQ Administrative Interface"

This chapter describes the administrative interface to Oracle Streams AQ.

Chapter 9, "Oracle Streams AQ Administrative Interface: Views"

This chapter describes how to use Oracle Streams AQ views administrative interface. It includes syntax and examples.

Chapter 10, "Oracle Streams AQ Operational Interface: Basic Operations"

This chapter describes how to use the Oracle Streams AQ operational interface. It includes syntax and examples.

Part V. Using Oracle JMS and Oracle Streams AQ

Chapter 11, "Creating Oracle Streams AQ Applications Using JMS"

This chapter describes how to create application using Oracle JMS interface with Oracle Streams AQ.

Chapter 12, "Oracle Streams AQ JMS Interface: Basic Operations"

This chapter describes how to use the Oracle Streams AQ administrative interface for JMS.

Chapter 13, "Oracle Streams AQ JMS Operational Interface: Point-to-Point"

This chapter describes how to use Oracle JMS interface with Oracle Streams AQ for point-to-point operations.

Chapter 14, "Oracle Streams AQ JMS Operational Interface: Publish/Subscribe"

This chapter describes how to use Oracle JMS interface with Oracle Streams AQ for publish/subscribe operations.

Chapter 15, "Oracle Streams AQ JMS Operational Interface: Shared Interfaces"

This chapter describes how to use Oracle JMS interface with Oracle Streams AQ for shared interface operations.

Chapter 16, "Oracle Streams AQ JMS Types Examples"

This chapter provides JMS type enqueuing and dequeuing examples for bytes, streams, and map message types. The examples illustrate how you can use JMS and `DBMS_AQ` for enqueuing and dequeuing.

Part VI. Internet Access with Oracle Streams AQ

Chapter 17, "Internet Access to Oracle Streams AQ"

This chapter describes how to perform Oracle Streams AQ operations over the Internet using its Internet Data Access Presentation (IDAP) and Simple Object Access Protocol (SOAP). It also shows how to transmit messages over the Internet using HTTP.

Part VII. Using Messaging Gateway

Chapter 18, "Introducing Oracle Messaging Gateway"

This chapter introduces Messaging Gateway's features, functions, and architecture. It describes how applications based on Oracle Streams AQ can communicate with non-Oracle messaging systems using Messaging Gateway.

Chapter 19, "Getting Started with Oracle Messaging Gateway"

This chapter describes the prerequisites for running Messaging Gateway, how to load and unload Messaging Gateway, and how to set it up for use.

Chapter 20, "Working with Oracle Messaging Gateway"

This chapter describes how to use Messaging Gateway: how to configure, start, and stop it, and how to configure Messaging Gateway Agent.

Chapter 21, "Oracle Messaging Gateway Message Conversion"

This chapter shows how to transform messages between Oracle Streams AQ formats and those used by supported third-party messaging systems.

Chapter 22, "Monitoring Oracle Messaging Gateway"

This chapter discusses abnormal situations you may experience, several sources of information about Messaging Gateway errors and exceptions, and suggested remedies.

Part VIII. Using Oracle Streams and Oracle Streams AQ

Chapter 23, "Staging and Propagating with Oracle Streams AQ"

This chapter describes how to use Oracle Streams for staging and propagation of queues and `SYS_AnyData`.

Chapter 24, "Oracle Streams Messaging Example"

This chapter includes a detailed example that illustrates how to use Oracle Streams for messaging.

Part IX. Troubleshooting Oracle Streams AQ

Chapter 25, "Troubleshooting Oracle Streams AQ"

This chapter describes ways you can troubleshoot Oracle Streams AQ.

Appendix A, "Scripts for Implementing BooksOnLine"

This appendix provides scripts for implementing the BooksOnLine example.

Glossary

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Application Developer's Guide - Fundamentals*
- *PL/SQL User's Guide and Reference*
- *Oracle Streams Advanced Queuing Java API Reference*
- *PL/SQL Packages and Types Reference*
- *Oracle Streams Concepts and Administration*
- *Oracle XML DB Developer's Guide*

For Oracle APIs for JMS see:

```
http://otn.oracle.com/docs/products/aq/doc_library/ojms/index.html
```

Many examples in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

```
http://oraclestore.oracle.com/
```

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

```
http://otn.oracle.com/membership/
```

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

```
http://otn.oracle.com/documentation/
```

Conventions

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

- Conventions in Text
- Conventions in Code Examples
- Conventions for Windows Operating Systems

Conventions in Text

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

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

Conventions in Code Examples

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

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

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

<table>
<tr><th>Convention</th><th>Meaning</th><th>Example</th></tr>
<tr><td><code>[ ]</code></td><td>Brackets enclose one or more optional items. Do not enter the brackets.</td><td><code>DECIMAL (digits [, precision])</code></td></tr>
<tr><td><code>{ }</code></td><td>Braces enclose two or more items, one of which is required. Do not enter the braces.</td><td><code>{ENABLE | DISABLE}</code></td></tr>
<tr><td><code>|</code></td><td>A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.</td><td><code>{ENABLE | DISABLE}</code>
<code>[COMPRESS | NOCOMPRESS]</code></td></tr>
<tr><td><code>. . .</code></td><td>Horizontal ellipsis points indicate either:
■ That we have omitted parts of the code that are not directly related to the example
■ That you can repeat a portion of the code</td><td><code>CREATE TABLE ... AS subquery;</code>

<code>SELECT col1, col2, ... , coln FROM employees;</code></td></tr>
<tr><td>.
.
.</td><td>Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.</td><td><code>SQL> SELECT NAME FROM V$DATAFILE;</code>
<code>NAME</code>
<code>------------------------------------</code>
<code>/fsl/dbs/tbs_01.dbf</code>
<code>/fs1/dbs/tbs_02.dbf</code>
<code>.</code>
<code>.</code>
<code>.</code>
<code>/fsl/dbs/tbs_09.dbf</code>
<code>9 rows selected.</code></td></tr>
<tr><td>Other notation</td><td>You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.</td><td><code>acctbal NUMBER(11,2);</code>
<code>acct   CONSTANT NUMBER(4) := 3;</code></td></tr>
</table>

Convention	Meaning	Example
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Windows Operating Systems

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

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32 is the same as C:\WINNT\SYSTEM32`

Convention	Meaning	Example
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`
Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"` `C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`
`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_ BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*nn*, where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle Database Platform Guide for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE**ORACLE_ HOME*\rdbms\admin directory.

Documentation Accessibility

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

`http://www.oracle.com/accessibility/`

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

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

What's New in Oracle Streams AQ?

This section describes new features of Oracle Streams Advanced Queuing (AQ) and provides pointers to additional information. New features information from previous releases is also retained to help those users migrating to the current release.

The following sections describe new features:

- Oracle Streams AQ Release 10.1 New Features
- Oracle9i Release 2 (9.2.0) New Features
- Oracle9i Release 1 (9.0.1) New Features in Oracle Streams AQ
- Oracle8i New Features in Oracle Streams AQ

Oracle Streams AQ Release 10.1 New Features

Advanced Queuing has been integrated into Oracle Streams, and is now called Oracle Streams AQ. It supports all existing functionality and more, including:

- New `AQ Types`

  `Message_Properties_T_Array` Type

  `Message_Properties_T` Type has an additional attribute, `transaction_group`.

- Oracle JMS now supports JMS version 1.1 specifications.

  In earlier versions of JMS, point-to-point and publish/subscribe operations could not be used in the same transaction. JMS version 1.1 includes methods that overcome this restriction, and Oracle JMS supports these new methods.

  > **See Also:** "J2EE Compliance" on page 11-2

- New JMS Types and added functionality to existing types

  > **See Also:** "Oracle Streams AQ Types" and "JMS Types" in *PL/SQL Packages and Types Reference*

- New `DBMS_AQ` packages
- New `DBMS_AQADM` packages

  A new Purge **API** allows for purging data in persistent queues. A limited subset of the Purge API is available for **buffered queues**.

  > **See Also:** "Purging a Queue Table" on page 8-9

- New `V$` views for investigating the status of buffered queues:
  - `V$BUFFERED_QUEUES`
  - `V$BUFFERED_SUBSCRIBERS`
  - `V$BUFFERED_PUBLISHERS`

  > **See Also:** "Dynamic Performance (V$) Views" in *Oracle Database Reference* for more information on these views

- `AQ$`*`Queue_table_name`* has been expanded to show buffered messages.

- The **rules engine** has been enhanced for higher performance and workload.
- New Streams messaging high level API
- You can now enqueue and dequeue multiple messages with a single command.

  See Also:

  - "Enqueue an Array of Messages" on page 1-21
  - "Dequeue an Array of Messages" on page 1-24

- Propagation from object queues with BFILEs is now supported.

  See Also: "Propagation from Object Queues" on page 5-16

- New C++ interface to Oracle Streams AQ

  OCCI AQ is a set of interfaces in C++ that enable messaging clients to access Oracle Streams AQ for enterprise messaging applications. OCCI AQ makes use of the OCI interface to Oracle Streams AQ and encapsulates the queuing functionality supported by OCI.

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

- Parameter `AQ_TM_PROCESSES` is no longer needed in `init.ora`.

  See Also: "AQ_TM_PROCESSES Parameter No Longer Needed in init.ora" on page 3-9

- New Oracle Streams features related to Advanced Queuing include auto capture and apply message handlers.

  See Also: See *Oracle Streams Concepts and Administration*

Oracle Messaging Gateway

In this release Oracle Messaging Gateway has the following new functionality:

- Message propagation between Oracle Java Message Service (OJMS) and IBM WebSphere MQ JMS. Propagation is supported for JMS queues and topics.
- Message propagation between Oracle Streams AQ and TIBCO TIB/Rendezvous for application integration.

See Also: *PL/SQL Packages and Types Reference*, chapters:

- AQ Types
- JMS Types
- DBMS_AQ
- DBMS_AQADM
- DBMS_MGWADM
- DBMS_MGWMSG

Deprecated Features

Java AQ API is deprecated in favor of a unified, industry-standard JMS interface. The Java AQ API is still being supported for legacy applications. However, Oracle recommends that you migrate your Java AQ API application to JMS and that new applications use JMS.

Also deprecated in this release are 8.0-style queues. All new functionality and performance improvements are confined to the newer style queues. 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.

Oracle9*i* Release 2 (9.2.0) New Features

- Oracle Messaging Gateway

  The interaction between different messaging systems is a common integration requirement. Messaging Gateway allows Advanced Queuing to propagate messages to and from non-Oracle messaging systems. It allows secure, transactional, and guaranteed one-time-only delivery of messages between Oracle Advanced Queuing and IBM Websphere MQ v5.1 and v5.2. See Chapter 18, "Introducing Oracle Messaging Gateway" for more information.

- Standard JMS Support

  The JMS implementation in Oracle9*i* release 2 (9.2.0) conforms to Sun Microsystems JMS 1.0.2b standard.

- XMLType Payload Support

  You are no longer required to embed an XMLType attribute in an Oracle object type. You can directly use an XMLType message as the message payload.

Oracle9*i* Release 1 (9.0.1) New Features in Oracle Streams AQ

Oracle9*i* introduces the following new Oracle Streams AQ features to improve e-business integration and use standard Internet transport protocols:

- Internet Integration

  To perform queuing operations over the Internet, Oracle Streams AQ takes advantage of the Internet Data Access Presentation (IDAP), which defines message structure using XML. Using IDAP, Oracle Streams AQ operations such as enqueue, dequeue, notification, and propagation can be executed using HTTP(S). Third-party clients, including third-party messaging vendors, can also interoperate with Oracle Streams AQ over the Internet using Messaging Gateway.

  IDAP messages may be requests, responses, or an error response. An IDAP document sent from an Oracle Streams AQ client contains an attribute for designating the remote operation; that is, enqueue, dequeue, or register accompanied by operational data. The Oracle Streams AQ implementation of IDAP can also be used to process batched enqueue and dequeue of messages.

  The HTTP support in Oracle Streams AQ is implemented by using the Oracle Streams AQ servlet which is bundled with the Oracle Database server. A client invokes the servlet through an HTTP post request that is sent to the Web server. The Web server invokes the servlet mentioned in the post method if one is not already invoked. The servlet parses the content of the IDAP document and uses the Java AQ API to perform the designated operation. On completion of the call, the servlet formats either a response or an error response as indicated by IDAP and sends it back to the client.

  IDAP is transport independent and therefore can work with other transport protocols transparently. Oracle Database supports HTTP; other proprietary protocols can also be supported using the callout mechanism through transformations.

- Oracle Streams AQ Security over the Internet

  Oracle Streams AQ functionality allows only authorized Internet users to perform operations on queues. An Internet user connects to a Web server, which in turn connects to the database using an application server. The Internet user doing the operation is typically not the database user connected to the database. Also, the Oracle Streams AQ queues cannot reside in the same schema as the connected database user. Oracle Streams AQ uses proxy authentication so that only authorized Internet users can perform operations on queues.

- LDAP Integration

  Oracle Internet Directory Integration: To leverage **Lightweight Directory Access Protocol** (LDAP) as the single point for managing generic information, Oracle Streams AQ is integrated with the Oracle Internet Directory server. This addresses the following requirements:

  - Global topics (queues): Oracle Streams AQ queue information can be stored in an Oracle Internet Directory server. Oracle Internet Directory provides a single point of contact to locate the required topic or queue. Business applications (users) looking for specific information need not know in which database the queue is located. Using the industry standard Java Message Service (JMS) API, users can directly connect to the queue without explicitly specifying the database or the location of the topic or queue.
  - Global events: Oracle Internet Directory can be used as the repository for event registration. Clients can register for database events even when the database is down. This allows clients to register for events such as "Database Open," which would not have been possible earlier. Clients can register for events in multiple databases in a single request.

  XML Integration: XML has emerged as a standard for e-business data representations. The XMLType datatype has been added to the Oracle server to support operations on XML data. Oracle Streams AQ not only supports XMLType data type payloads, but also allows definitions of subscriptions based on the contents of an XML message. This is powerful functionality for online market places where multiple vendors may define their subscriptions based on the contents of the orders.

- Transformation Infrastructure

  Applications are designed independent of each other. So, the messages they understand are different from each other. To integrate these applications, messages must be transformed. There are various existing solutions to handle these transformations. Oracle Streams AQ provides a transformation infrastructure that can be used to plug in transformation functionality from Oracle Application Interconnect or other third-party solutions such as Mercator without losing Oracle Streams AQ functionality. Transformations can be specified as PL/SQL call back functions, which are applied at enqueue, dequeue, or propagation of messages. These PL/SQL callback functions can call third-party functions implemented in C, Java, or PL/SQL. XSLT transformations can also be specified for XML messages.

- Oracle Streams AQ Management

  You can use new and enhanced Oracle Enterprise Manager to manage Oracle Streams AQ, as follows:

  - Improved UI task flow and administration of queues, including a topology display at the database level and at the queue level, error and propagation schedules for all the queues in the database, and relevant initialization parameters (init.ora)
  - Ability to view the message queue

  Oracle diagnostics and tuning pack supports alerts and monitoring of Oracle Streams AQ queues. Alerts can be sent when the number of messages for a particular subscriber exceeds a threshold. Alerts can be sent when there is an error in propagation. In addition, queues can be monitored for the number of messages in ready state or the number of messages for each subscriber.

- Additional Enhancements

  PL/SQL notifications and e-mail notifications: Oracle9*i* allows notifications on the queues to be PL/SQL functions. Using this functionality, users can register PL/SQL functions that are called when a message of interest is enqueued. Using e-mail notification functionality, an e-mail address can be registered to provide notifications. E-mail is sent if the message of interest arrives in the queue. Presentation of the e-mail message can also be specified while registering for e-mail notification. Users can also specify an HTTP URL to which notifications can be sent.

  Dequeue enhancements: Using the dequeue with a condition functionality, subscribers can select messages that satisfy a specified condition from the messages meant for them.

  Overall performance improvements: Oracle Streams AQ exhibits overall performance improvements as a result of code optimization and other changes.

  Propagation enhancements: The maximum number of job queue processes has been increased from 36 to 1000 in Oracle9*i*. With Internet propagation, you can set up propagation between queues over HTTP. Overall performance improvements have been made in propagation due to design changes in the propagation algorithm.

- JMS Enhancements

  All the new Oracle9*i* features are supported through JMS, as well as the following:

  - Connection pooling: Using this feature, a pool of connection can be established with the Oracle Database server. Later, at the time of establishing a **JMS session**, a connection from the pool can be picked up.
  - Global topics: This is the result of the integration with Oracle Internet Directory. Oracle Streams AQ queue information can be stored and looked up from it.
  - Topic browsing: Allows durable subscribers to browse through the messages in a **publish/subscribe** (topic) destination, and optionally allows these subscribers to purge the browsed messages (so that they are no longer retained by Oracle Streams AQ for that subscriber).
  - Exception listener support: This allows a client to be asynchronously notified of a problem. Some connections only consume messages, so they have no other way to learn that their connection has failed.

Oracle8*i* New Features in Oracle Streams AQ

The Oracle8*i* release included the following Advanced Queuing features:

- Queue-level access control
- Nonpersistent queues
- Support for Real Application Clusters
- Rule-based subscribers for publish/subscribe
- Asynchronous notification
- Sender identification
- Listen capability (wait on multiple queues)
- Propagation of messages with LOBs
- Enhanced propagation scheduling
- Dequeuing message headers only
- Support for statistics views
- Java API (native AQ)

- Java Message Service (JMS) API
- Separate storage of history management information

Part I

Introducing Oracle Streams AQ

Part I introduces Oracle Streams Advanced Queuing (AQ) and tells you how to get started with it. It also describes its main components and supported programing languages.

This part contains the following chapters:

- Chapter 1, "Introducing Oracle Streams AQ"
- Chapter 2, "Getting Started with Oracle Streams AQ"
- Chapter 3, "Basic Components"
- Chapter 4, "Oracle Streams AQ: Programmatic Environments"

1

Introducing Oracle Streams AQ

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:

- Overview of Oracle Streams AQ
- Oracle Streams AQ in Integrated Application Environments
- Oracle Streams AQ Client/Server Communication
- Multiconsumer Dequeuing of the Same Message
- Oracle Streams AQ Implementation of Workflows
- Oracle Streams AQ Implementation of Publish/Subscribe
- Message Propagation
- Message Format Transformation
- Internet Integration and Internet Data Access Presentation
- Interfaces to Oracle Streams AQ
- Oracle Streams AQ Features
- Oracle Streams AQ Demos

Note: For helpful examples on using Oracle Streams AQ, search for the "Oracle By Example Series" at the OTN Web site:

`http://otn.oracle.com/index.html`

Overview of Oracle Streams AQ

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.

Oracle Streams AQ 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 Streams AQ 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 Streams AQ. Also queue tables can be imported and exported. You can use database development and management tools such as Oracle Enterprise Manager to monitor queues.

See Also: Chapter 5, "Managing Oracle Streams AQ"

Performance

Requests for service must be decoupled from supply of services to increase efficiency and provide the infrastructure for complex scheduling. Oracle Streams AQ exhibits high performance characteristics as measured by the following metrics:

- Number of messages enqueued/dequeued each second
- Time to evaluate a complex query on a message warehouse
- Time to recover/restart the messaging process after a failure

Scalability

Queuing systems must be scalable. Oracle Streams AQ 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 Streams AQ is able to meet requirements in the following situations:

- Applications that 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 that 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 that are not ready to receive messages that have been processed.

Persistence for Scheduling

Queuing systems need message persistence so they can deal with priorities: messages arriving later can be of higher priority than messages arriving earlier; messages arriving earlier may wait for messages arriving later before actions are executed; the same message may be accessed by different processes; and so on. Priorities also change. Messages in a specific queue can become more important, and so must be processed with less delay or interference from messages in other queues. Similarly, messages sent to some destinations can have a higher priority than others.

Persistence for Accessing and Analyzing Metadata

Message persistence is needed to 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 Streams AQ, you can analyze periods of greatest demand or evaluate the lag between receiving and completing an order.

See Also: Chapter 6, "Oracle Streams AQ Performance and Scalability"

Oracle Streams AQ in Integrated Application Environments

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

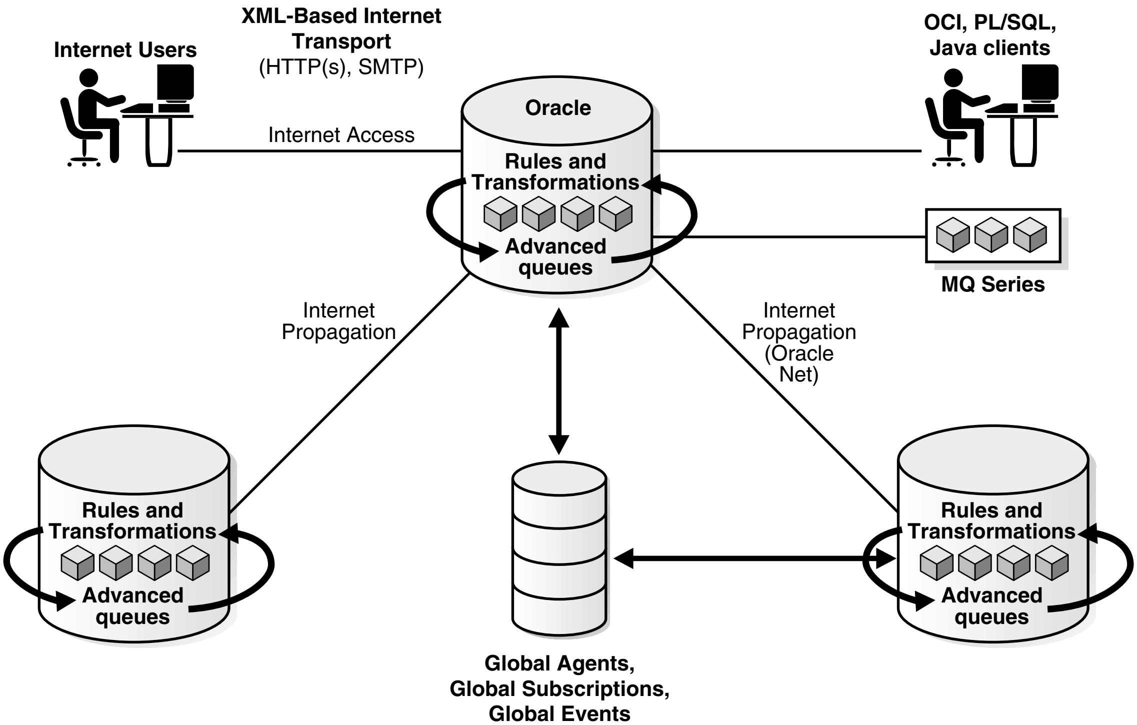
Using Oracle Net Services, messages are exchanged between a client and the Oracle Database server or between two Oracle Database servers. Oracle Net Services also propagates messages from one Oracle Database queue to another. Or, as shown in Figure 1–1, you can perform Oracle Streams AQ operations over 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.

See Also: Chapter 17, "Internet Access to Oracle Streams AQ" for more information on Internet integration with Oracle Streams AQ

Application integration also involves the integration of heterogeneous messaging systems. Oracle Streams AQ 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 Streams AQ environment.

See Also: Chapter 18, "Introducing Oracle Messaging Gateway" for more information on Oracle Streams AQ integration with non-Oracle Database messaging systems

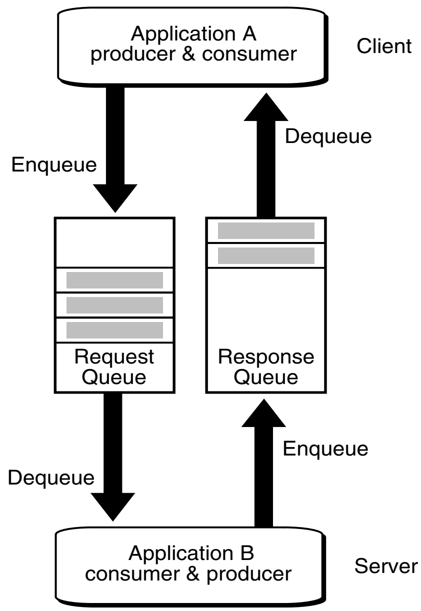
Figure 1–1 Integrated Application Environment Using Oracle Streams AQ



Oracle Streams AQ Client/Server Communication

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

Figure 1–2 Client/Server Communication Using Oracle Streams AQ



Application A enqueues a request into the request queue. Application B dequeues and processes the request. Application B enqueues the result in the response queue, and 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.

Note: The various enqueue and dequeue operations are part of different transactions.

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 Streams AQ 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

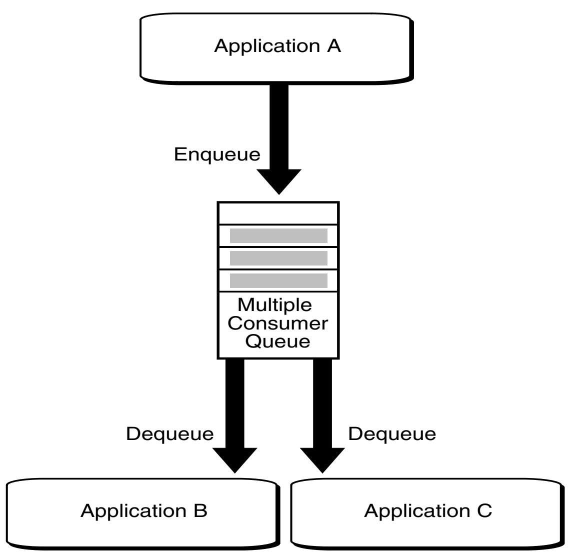
Multiple consumers can be associated with a queue as subscribers. This causes all messages enqueued in the queue to be made available to be consumed by each of the queue subscribers. The subscribers to the queue can be changed dynamically without any change to the messages or message producers. Subscribers to the queue are added and removed by using the Oracle Streams AQ administrative package.

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 undetermined.

Every message will eventually be dequeued by every subscriber.

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



Note: Queue subscribers can be applications or other queues.

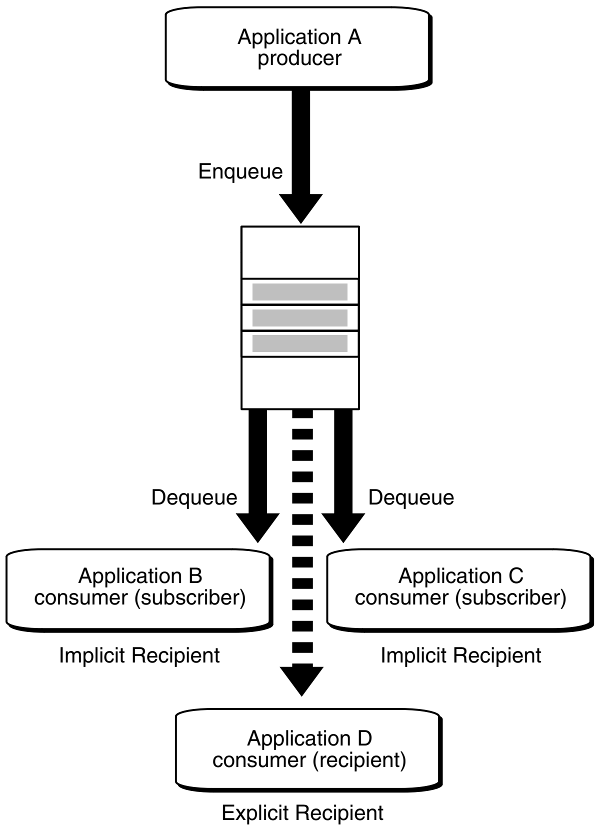
Message Recipients

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.

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 Streams AQ 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.

Figure 1–4 Explicit and Implicit Recipients of Messages

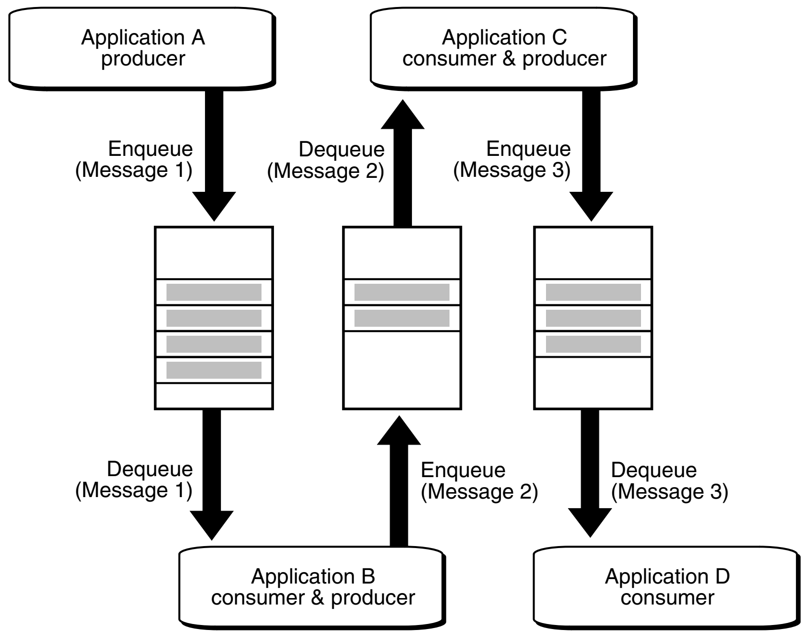


Note: Multiple producers can simultaneously enqueue messages aimed at different targeted recipients.

Oracle Streams AQ Implementation of Workflows

Figure 1–5 illustrates the use of Oracle Streams AQ for implementing a **workflow**, also known as a chained application transaction. Application A begins a workflow by enqueuing Message 1. Application B dequeues it, performs whatever activity is required, and enqueues Message 2. Application C dequeues Message 2 and generates Message 3. Application D, the final step in the workflow, dequeues it.

Figure 1–5 Implementing a Workflow using Oracle Streams AQ



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 of the contents of previous messages.

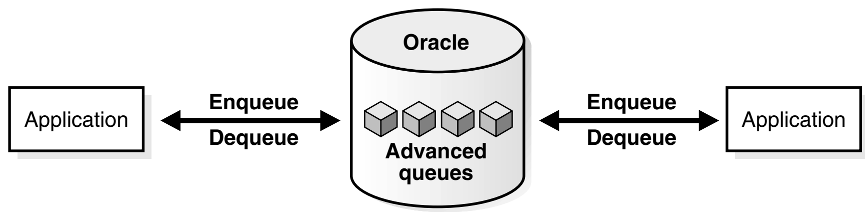
The queues are used to buffer the flow of information between different processing stages of the business process. By specifying delay interval and expiration time for a message, a window of execution can be provided for each of the applications.

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 Streams AQ helps you gain this knowledge by supporting the optional retention of messages for analysis of historical patterns and prediction of future trends.

Oracle Streams AQ 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.

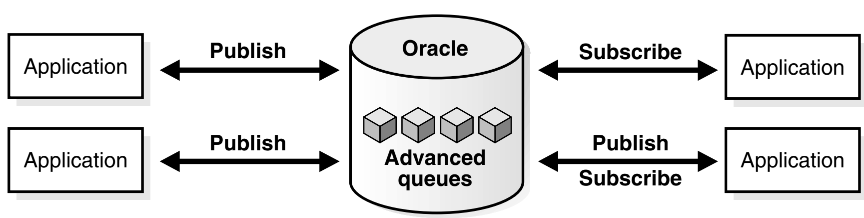
Figure 1–6 Point-to-Point Messaging



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.

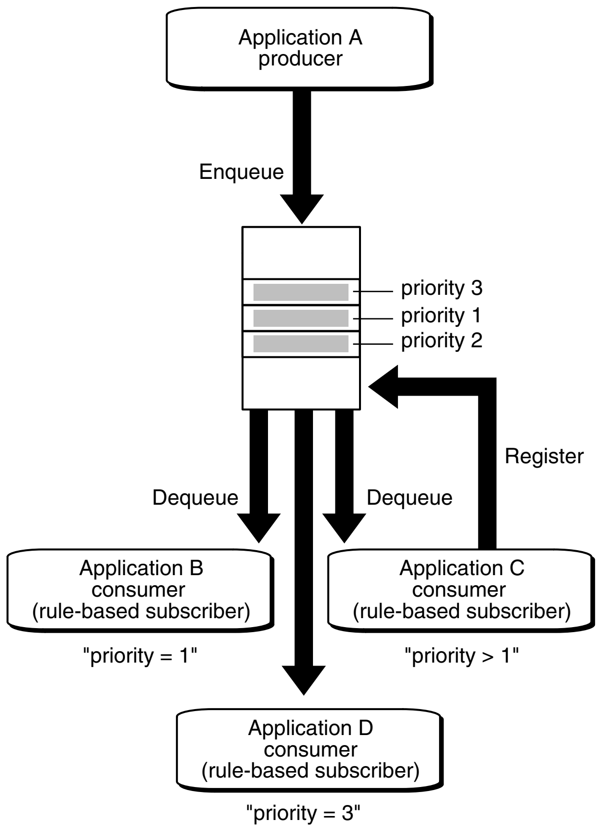
Figure 1–7 Publish/Subscribe Mode



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.

Figure 1–8 illustrates the use of Oracle Streams AQ 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", and application D subscribes with rule "priority = 3".

Figure 1–8 Implementing Publish/Subscribe using Oracle Streams AQ



Application A enqueues 3 messages with differing priorities. Application B receives a single message (priority 1), application C receives two messages (priority 2, 3) and application D receives a single message (priority 3). Message recipients are computed dynamically based on message properties and content.

A combination of Oracle Streams AQ features allows publish/subscribe messaging between applications. These features, described later in this guide, include rule-based subscribers, message propagation, the listen feature, and notification capabilities.

Message Propagation

Enqueued messages are said to be propagated when they are reproduced on another queue.

This section contains these topics:

- Fanning Out Messages
- Compositing Messages
- Inboxes and Outboxes

Fanning Out Messages

In Oracle Streams AQ, message recipients can be either consumers or other queues. If the message recipient is a queue, then message recipients include all subscribers to the queue (one or more of which can be other queues). Thus it is possible to fan out messages to a large number of recipients without requiring them all to dequeue messages from a single queue.

For example, imagine a queue named `Source` with subscriber queues `dispatch1@dest1` and `dispatch2@dest2`. Queue `dispatch1@dest1` has subscriber queues `outerreach1@dest3` and `outerreach2@dest4`, while queue `dispatch2@dest2` has subscriber queues `outerreach3@dest21` and `outerreach4@dest4`. Messages enqueued in `Source` are propagated to all the subscribers of four different queues.

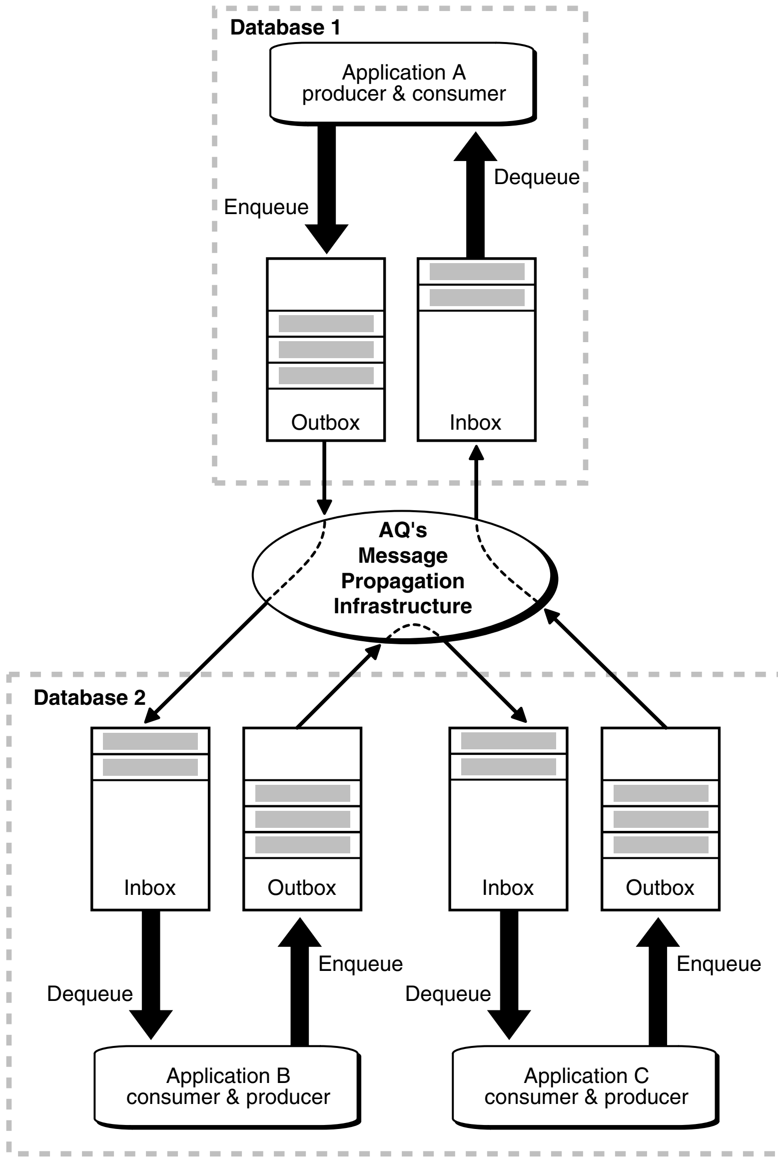
Compositing Messages

Messages from different queues can be combined into a single queue. This is also known as funneling. For example, if queue `composite@endpoint` is a subscriber to both `funnel1@source1` and `funnel2@source2`, then subscribers to `composite@endpoint` get all messages enqueued in those queues as well as messages enqueued directly to `composite@endpoint`.

Inboxes and Outboxes

Figure 1–9 illustrates applications on different databases communicating using Oracle Streams AQ. 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. Messages sent locally (on the same node) and messages sent remotely (on a different node) all go in the outbox. Similarly, an application dequeues messages from its inbox no matter where the message originates. Oracle Streams AQ facilitates such interchanges, treating all messages on the same basis.

Figure 1–9 Message Propagation in Oracle Streams AQ



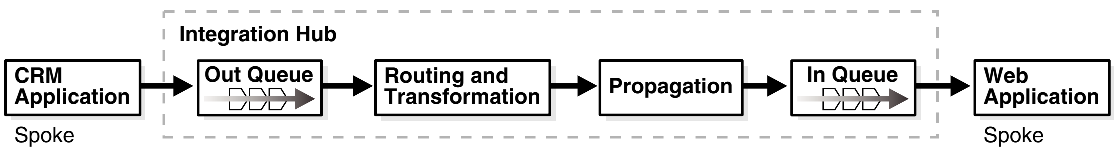
Message Format Transformation

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.

You can arrange transformations to occur when a message is enqueued, when it is dequeued, or when it is propagated to a remote subscriber.

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



Internet Integration and Internet Data Access Presentation

You can access Oracle Streams AQ over the Internet by using **Simple Object Access Protocol** (SOAP). **Internet Data Access Presentation** (IDAP) is the SOAP specification for Oracle Streams AQ 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).

This section contains these topics:

- Internet Message Payloads
- Propagation over the Internet Using HTTP
- Internet Data Access Presentation (IDAP)

See Also: Chapter 17, "Internet Access to Oracle Streams AQ"

Internet Message Payloads

Oracle Streams AQ 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 XML specified here must map to the SQL type of the payload for the queue table.

> **See Also:** *Oracle XML DB Developer's Guide* for details on mapping SQL types to XML

Example 1–1 A Queue Type and its XML Equivalent

Assume the queue is defined to be of type `EMP_TYP`, which has the following structure:

```
CREATE OR REPLACE TYPE emp_typ AS object (
     empno NUMBER(4),
     ename VARCHAR2(10),
     job VARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal   NUMBER(7,2),
     comm  NUMBER(7,2)
     deptno NUMBER(2));
```

The corresponding XML representation is:

```
<EMP_TYP>
  <EMPNO>1111</EMPNO>
  <ENAME>Mary</ENAME>
  <MGR>5000</MGR>
  <HIREDATE>1996-01-01 0:0:0</HIREDATE>
  <SAL>10000</SAL>
  <COMM>100.12</COMM>
  <DEPTNO>60</DEPTNO>
</EMP_TYP>
```

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.

> **See Also:** "IDAP Documents" on page 17-6 for examples of using different IDAP message payload

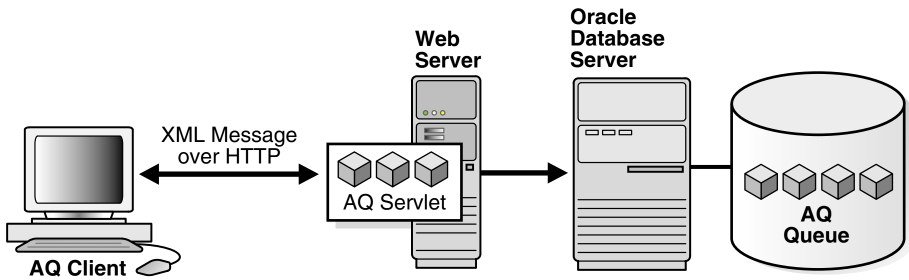
Propagation over the Internet Using HTTP

Figure 1–11 shows the architecture for performing Oracle Streams AQ operations over HTTP. The major components are:

- Oracle Streams AQ client program
- Web server/Servlet Runner hosting the Oracle Streams AQ **servlet**
- Oracle Database server

The Oracle Streams AQ client program sends XML messages (conforming to IDAP) to the Oracle Streams AQ servlet, which understands the XML message and performs Oracle Streams AQ operations. Any HTTP client, for example Web browsers, can be used. The Web server/Servlet Runner hosting the Oracle Streams AQ servlet interprets the incoming XML messages. Examples include Apache/Jserv or Tomcat. The Oracle Streams AQ servlet connects to the Oracle Database server and performs operations on the users' queues.

Figure 1–11 Architecture for Performing Oracle Streams AQ Operations Using HTTP



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 Streams AQ operations are scoped in the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes in the SOAP body.
- The receiver processes IDAP messages that have correct namespaces. For requests with incorrect namespaces, the receiver returns an invalid request error.
- 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`

> **See Also:** Chapter 17, "Internet Access to Oracle Streams AQ" for more information about IDAP

Interfaces to Oracle Streams AQ

You can access Oracle Streams AQ 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 has been deprecated in Oracle Streams AQ release 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:

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

Oracle Streams AQ Features

This section contains these topics:

- Enqueue Features
- Dequeue Features
- Propagation Features
- Other Oracle Streams AQ Features

Enqueue Features

The following features apply to enqueuing messages:

- Enqueue an Array of Messages
- Correlation Identifiers
- Subscription and Recipient Lists
- Priority and Ordering of Messages in Enqueuing

- Message Grouping
- Propagation
- Sender Identification
- Time Specification and Scheduling
- Rule-Based Subscribers
- Asynchronous Notification

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.

> **See Also:** "Enqueuing an Array of Messages" on page 10-12

Correlation Identifiers

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

Subscription and Recipient Lists

A single message can be designed to be consumed by multiple consumers. A queue administrator can specify the list of subscribers who can retrieve messages from a queue. Different queues can have different subscribers, and a consumer program can be a subscriber to more than one queue. Further, 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.

You can design a single message for consumption by multiple consumers in a number of different ways. The consumers who are allowed to retrieve the message are specified as explicit recipients of the message by the user or application that enqueues the message. Every explicit recipient is an agent identified by name, address, and protocol.

A queue administrator can also specify a default list of recipients who can retrieve all the messages from a specific queue. These implicit recipients become subscribers to the queue by being specified in the default list. If a message is enqueued without specifying any explicit recipients, then the message is delivered to all the designated subscribers.

A rule-based subscriber is one that has a rule associated with it in the default recipient list. A rule-based subscriber is sent a message with no explicit recipients specified only if the associated rule evaluated to TRUE for the message. Different queues can have different subscribers, and the same recipient can be a subscriber to more than one queue. Further, 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.

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.

Priority and Ordering of Messages in Enqueuing

It is possible to specify the priority of the enqueued message. An enqueued message can also have its exact position in the queue specified. This means that users have three options to specify the order in which messages are consumed: (a) a sort order specifies which properties are used to order all message in a queue; (b) a priority can be assigned to each message; (c) a sequence deviation positions a message in relation to other messages. Further, if several consumers act on the same queue, then a 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.

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 the header message, followed by messages representing details, followed by the trailer message.

Propagation

This feature enables applications to communicate with each other without having to be connected to the same database or the same queue. Messages can be propagated from one Oracle Streams AQ to another, irrespective of whether the queues are local or remote. Propagation is accomplished using database links and Oracle Net Services.

Sender Identification

Applications can mark the messages they send with a custom identification. Oracle Streams AQ 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

Delay interval or expiration intervals can be specified for an enqueued message, thereby providing windows of execution. A message can be marked as available for processing only after a specified time elapses (a delay time) and must be consumed before a specified time limit expires.

Rule-Based Subscribers

A message can be delivered to multiple recipients based on message properties or message content. Users define a rule-based subscription for a given queue as the mechanism to specify interest in receiving messages of interest. Rules can be specified based on message properties and message data (for object and raw payloads). Subscriber rules are then used to evaluate recipients for message delivery.

Asynchronous Notification

The asynchronous notification feature allows clients to receive notification of a message of interest. The client can use it to monitor multiple subscriptions. The client need not be connected to the database to receive notifications regarding its subscriptions.

Clients can use the **Oracle Call Interface** (OCI) function `OCISubscriptionRegister` or the PL/SQL procedure `DBMS_AQ.REGISTER` to register interest in messages in a queue.

See Also: "Registering for Notification" on page 10-39

Dequeue Features

The following features apply to dequeuing messages:

- Dequeue an Array of Messages
- Recipients
- Navigation of Messages in Dequeuing
- Modes of Dequeuing
- Optimization of Waiting for the Arrival of Messages
- Retries with Delays
- Optional Transaction Protection
- Exception Handling
- Listen Capability (Wait on Multiple Queues)
- Dequeue Message Header with No Payload

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.

See Also: "Dequeuing an Array of Messages" on page 10-34

Recipients

A message can be retrieved by multiple recipients without the need for multiple copies of the same message. Designated recipients can be located locally or at remote sites.

Navigation of Messages in Dequeuing

Users have several options to select a message from a queue. They can select the first message or once they have selected a message and established a position, they can retrieve the next. The selection is influenced by the ordering or can be limited by specifying a correlation identifier. Users can also retrieve a specific message using the message identifier.

Modes of Dequeuing

A dequeue request can either browse or remove a message. If a message is browsed, then it remains available for further processing. If a message is removed, then it is not available more for dequeue requests. Depending on the queue properties, a removed message can be retained in the queue table.

Optimization of Waiting for the Arrival of Messages

A dequeue request can be applied against an empty queue. To avoid polling for the arrival of a new message, a user can specify if and for how long the request is allowed to wait for the arrival of a message.

Retries with Delays

A message must be consumed exactly once. If an attempt to dequeue a message fails and the transaction is rolled back, then the message is made available for reprocessing after some user-specified delay elapses. Reprocessing is attempted up to the user-specified limit.

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.

Exception Handling

A message may not be consumed within given constraints, such as within the window of execution or within the limits of the retries. If such a condition arises, then the message is moved to a user-specified **exception queue**.

Listen Capability (Wait on Multiple Queues)

The listen call is a blocking call that can be used to wait for messages on multiple queues. It can be used by a gateway application to monitor a set of queues. An application can also use it to wait for messages on a list of subscriptions. If the listen returns successfully, then a dequeue must be used to retrieve the message.

Dequeue Message Header with No Payload

The dequeue mode `REMOVE_NODATA` can be used to remove a message from a queue without retrieving the payload. Use this mode to delete a message with a large payload whose content is irrelevant.

Propagation Features

The following features apply to propagating messages:

- Automatic Coordination of Enqueuing and Dequeuing
- Propagation of Messages with LOBs
- Propagation Scheduling
- Enhanced Propagation Scheduling Capabilities
- Third-Party Support

> **See Also:** "Internet Integration and Internet Data Access Presentation" on page 1-16 for information on propagation over the Internet

Automatic Coordination of Enqueuing and Dequeuing

Recipients can be local or remote. Because Oracle Database does not support distributed object types, remote enqueuing or dequeuing using a standard database link does not work. However, you can use Oracle Streams AQ message propagation to enqueue to a remote queue. For example, you can connect to database X and enqueue the message in a queue, `DROPBOX`, located in database X. You can configure Oracle Streams AQ so that all messages enqueued in `DROPBOX` are automatically propagated to another queue in database Y, regardless of whether database Y is local or remote. Oracle Streams AQ automatically checks if the type of the remote queue in database Y is structurally equivalent to the type of the local queue in database X and propagates the message.

Recipients of propagated messages can be applications or queues. If the recipient is a queue, then the actual recipients are determined by the subscription list associated with the recipient queue. If the queues are remote, then messages are propagated using the specified database link. AQ-to-AQ message propagation is directly supported; propagation between Oracle Streams AQ and other message systems, such as WebSphere MQ and TIB/Rendezvous, is supported through Messaging Gateway.

Propagation of Messages with LOBs

Propagation handles payloads with **LOB** attributes.

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

Propagation Scheduling

Messages can be scheduled to propagate from a queue to local or remote destinations. Administrators can specify the start time, the propagation window, and a function to determine the next propagation window (for periodic schedules).

Enhanced Propagation Scheduling Capabilities

Detailed run-time 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. For example, available statistics about the total and average number of message/bytes propagated can be used to tune schedules. 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.

Third-Party Support

Oracle Streams AQ allows messages to be enqueued in queues that can then be propagated to different messaging systems by third-party propagators. If the protocol number for a recipient is in the range 128 - 255, then the address of the recipient is not interpreted by Oracle Streams AQ and so the message is not propagated by the Oracle Streams AQ system. Instead, a third-party propagator can then dequeue the message by specifying a reserved consumer name in the dequeue operation. The reserved consumer names are of the form `AQ$_P#`, where # is the protocol number in the range 128–255. For example, the consumer name `AQ$_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 Streams AQ to propagate messages to and from third-party messaging systems is through Messaging Gateway, an Enterprise Edition feature. Messaging Gateway dequeues messages from an Oracle Streams AQ queue and guarantees delivery to a third-party messaging system such as Websphere MQ (MQSeries). Messaging Gateway can also dequeue messages from third-party messaging systems and enqueue them to an Oracle Streams AQ queue.

See Also: Chapter 18, "Introducing Oracle Messaging Gateway"

Other Oracle Streams AQ Features

This section contains these topics:

- Queue Monitor Coordinator
- Oracle Internet Directory
- Oracle Enterprise Manager Integration
- SQL Access
- Support for Statistics Views
- Structured and XMLType Payloads
- Retention and Message History
- Tracking and Event Journals
- Queue-Level Access Control
- Nonpersistent Queues
- Support for Oracle Real Application Clusters

Queue Monitor Coordinator

Before release 10.1, the Oracle Streams AQ time manager process was called queue monitor (`QMNn`), a background process controlled by setting the dynamic `init.ora` parameter `AQ_TM_PROCESSES`. Beginning with release 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

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 **buffered queues** and other Oracle Streams tasks, however, are not affected by this parameter.

Note: 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.

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 Internet Directory Administrator's Guide* for more information.

Oracle Enterprise Manager Integration

You can use Oracle Enterprise Manager to do the following:

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

See Also: "Oracle Enterprise Manager Support" on page 5-10

SQL Access

Messages are placed in normal rows in a database table, and so 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 Streams AQ 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.

Support for Statistics Views

Basic statistics about queues are available using the `GV$AQ` view.

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 Streams AQ 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 operators `ExistsNode()` and `SchemaMatch()`
- Specify the operators in subscriber rules or dequeue conditions

Retention and Message History

The systems administrator specifies the retention duration to retain messages after consumption. Oracle Streams AQ 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.

Tracking and Event Journals

If messages are retained, then they can be related to each other. For example, if a message `m2` is produced as a result of the consumption of message `m1`, then `m1` is related to `m2`. This allows users to track sequences of related messages. These sequences represent event journals, which are often constructed by applications. Oracle Streams AQ is designed to let applications create event journals automatically.

When an online order is placed, multiple messages are generated by the various applications involved in processing the order. Oracle Streams AQ offers features to track interrelated messages independent of the applications that generated them. You can determine who enqueued and dequeued messages, who the users are, and who did what operations.

With Oracle Streams AQ tracking features, you can use SQL SELECT and JOIN statements to get order information from AQ$*queuetablename* and the views ENQ_TRAN_ID, DEQ_TRAN_ID, USER_DATA (the payload), CORR_ID, and MSG_ID. These views contain the following data used for tracking:

- Transaction IDs from ENQ_TRAN_ID and DEQ_TRAN_ID, captured during enqueuing and dequeuing.
- Correlation IDs from CORR_ID, part of the message properties
- USER_DATA message content that can be used for tracking

Queue-Level Access Control

The owner of an 8.1-compatible queue can grant or revoke queue-level privileges on the queue. Database administrators can grant or revoke new Oracle Streams AQ system-level privileges to any database user. Database administrators can also make any database user an Oracle Streams AQ administrator.

Nonpersistent Queues

Oracle Streams AQ 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). Oracle Streams AQ supports persistent and nonpersistent messages with a common **API**.

Support for Oracle Real Application Clusters

An application can specify the instance affinity for a queue table. When Oracle Streams AQ is used with Real Application Clusters and multiple instances, this information is used to partition the queue tables between instances for queue-monitor scheduling. The queue table is monitored by the queue monitors of the instance specified by the user. If an instance affinity is not specified, then the queue tables are arbitrarily partitioned among the available instances. There can be pinging between the application accessing the queue table and the queue monitor monitoring it. Specifying the instance affinity does not prevent the application from accessing the queue table and its queues from other instances.

This feature prevents pinging between queue monitors and Oracle Streams AQ propagation jobs running in different instances. If compatibility is set to Oracle8*i* release 8.1.5 or higher, then an instance affinity (primary and secondary) can be specified for a queue table. When Oracle Streams AQ 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. At any time, the queue table is affiliated to one instance. In the absence of an explicitly specified affinity, any available instance is made the owner of the queue table. 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.

Nonrepudiation and the AQ$*QueueTableName* View

Oracle Streams AQ maintains the entire history of information about a message along with the message itself. You can look up history information by using the `AQ$`*`QueueTableName`* view. This information serves as the proof of sending and receiving of messages and can be used for nonrepudiation of the sender and nonrepudiation of the receiver.

> **See Also:** Chapter 9, "Oracle Streams AQ Administrative Interface: Views" for more information about the `AQ$`*`QueueTableName`* view

The following information is kept at enqueue for nonrepudiation of the enqueuer:

- Oracle Streams AQ agent doing the enqueue
- Database user doing the enqueue
- Enqueue time
- Transaction ID of the transaction doing the enqueue

The following information is kept at dequeue for nonrepudiation of the dequeuer:

- Oracle Streams AQ 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 propagation corresponds to the message ID of the source message. This field can be used to correlate the propagated messages. This is useful for nonrepudiation of the dequeuer of propagated messages.

Stronger nonrepudiation 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 dequeuer at the time of dequeue.

Oracle Streams AQ Demos

The following demos can be found in the `$ORACLE_HOME/rdbms/demo` directory. Refer to `aqxmlreadme.txt` and `aqjmsreadme.txt` in the demo directory for more information.

Table 1–1 Oracle Streams AQ Demos

Demo and Locations	Topic
`aqjmsdemo01.java`	Enqueue text messages and dequeue based on message properties
`aqjmsdemo02.java`	Message Listener demo
`aqjmsdemo03.java`	Message Listener demo
`aqjmsdemo04.java`	Oracle Type Payload - Dequeue on payload content
`aqjmsdemo05.java`	Example of the `QueueBrowser`
`aqjmsdemo06.java`	Schedule propagation between queues in the database
`aqjmsdmo.sql`	Set up Oracle Streams AQ JMS demos
`aqjmsREADME.txt`	Describes the Oracle Streams AQ Java API and JMS demos
`aqorademo01.java`	Enqueue and dequeue RAW messages
`aqorademo02.java`	Enqueue and dequeue object type messages using the Custom Datum interface
`aqoradmo.sql`	Setup file for Oracle Streams AQ Java API demos
`aqxml01.xml`	AQXmlSend—Enqueue to Oracle object type single- consumer queue with piggyback commit
`aqxml02.xml`	AQXmlReceive—Dequeue from Oracle object type single-consumer queue with piggyback commit
`aqxml03.xml`	AQXmlPublish—Enqueue to Oracle object type (with LOB) multiconsumer queue
`aqxml04.xml`	AQXmlReceive—Dequeue from Oracle object type multi-consumer queue
`aqxml05.xml`	AQXmlCommit—Commit previous operation
`aqxml06.xml`	AQXmlSend—Enqueue to JMS Text single-consumer queue with piggyback commit
`aqxml07.xml`	AQXmlReceive—Dequeue from JMS Text single-consumer queue with piggyback commit

Table 1–1 (Cont.) Oracle Streams AQ Demos

Demo and Locations	Topic
`aqxml08.xml`	AQXmlPublish—Enqueue JMS MAP message with recipient into multiconsumer queue
`aqxml09.xml`	AQXmlReceive—Dequeue JMS MAP message from multiconsumer queue
`aqxml10.xml`	AQXmlRollback—Roll back previous operation
`aqxmlhtp.sql`	HTTP Propagation
`aqxmlREADME.txt`	Describes the Internet access demos
`AQDemoServlet.java`	Servlet to post Oracle Streams AQ XML files (for Jserv)
`AQPropServlet.java`	Servlet for Oracle Streams AQ HTTP propagation
`aqdemo00.sql`	Create users, message types, tables, and so on
`aqdemo01.sql`	Set up queue_tables, queues, and subscribers
`aqdemo02.sql`	Enqueue messages
`aqdemo03.sql`	Install dequeue procedures
`aqdemo04.sql`	Perform blocking dequeue
`aqdemo05.sql`	Perform listen for multiple agents
`aqdemo06.sql`	Clean up users, queue_tables, queues, subscribers (cleanup script)
`aqdemo07.sql`	Enqueue /dequeue to queue of type ADT with XMLType
`aqdemo08.sql`	Notification
`aqdemo09.sql`	Set up queues and subscribers (for OCI array demos also)
`aqdemo10.sql`	Array enqueue 10 messages
`aqdemo11.sql`	Array dequeue 10 messages
`aqdemo12.sql`	Clean up queues and subscribers (for OCI array demos also)
`ociaqdemo00.c`	Enqueue messages
`ociaqdemo01.c`	Perform blocking dequeue
`ociaqdemo02.c`	Perform listen for multiple agents
`ociaqarrayenq.c`	Array enqueue 10 messages
`ociaqarraydeq.c`	Array dequeue 10 messages

2

Getting Started with Oracle Streams AQ

This chapter describes the prerequisites for using Oracle Streams Advanced Queuing (AQ). It discusses planning and design issues and includes several frequently asked questions about Oracle Streams AQ.

This chapter contains the following topics:

- Oracle Streams AQ Prerequisites
- Oracle Streams AQ by Example
- Frequently Asked Questions

Oracle Streams AQ Prerequisites

Oracle Streams AQ prerequisites depend on:

- Your operating environment and programing languages
- How structured your data is
- Messaging Requirements
- What your source and target systems are, in other words where you are sending your messages to and from.

Oracle Streams AQ is provided with Oracle Database 10*g*.

Oracle Streams AQ by Example

This section provides examples of Oracle Streams Advanced Queuing (AQ) operations using different programmatic environments.

This section contains these topics:

- Creating Oracle Streams AQ Queues and Queue Tables
- Enqueuing and Dequeuing Oracle Streams AQ Messages
- Oracle Streams AQ Propagation
- Dropping Oracle Streams AQ Objects
- Revoking Roles and Privileges
- Deploying Oracle Streams AQ with XA
- Oracle Streams AQ and Memory Usage

Creating Oracle Streams AQ Queues and Queue Tables

You must set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

The following examples illustrate how to create Oracle Streams AQ queues and queue tables:

- Creating a Queue Table and Queue of Object Type
- Creating a Queue Table and Queue of Raw Type
- Creating a Prioritized Message Queue Table and Queue
- Creating a Multiconsumer Queue Table and Queue
- Creating a Queue to Demonstrate Propagation
- Setting Up Java Oracle Streams AQ Examples
- Creating a Java Oracle Streams AQ Session for User 'aqjava'
- Creating a Queue Table and Queue Using Java
- Creating a Queue and Starting Enqueue or Dequeue Using Java
- Creating a Multiconsumer Queue and Adding Subscribers Using Java

Example 2–1 Creating a Queue Table and Queue of Object Type

```
/* Creating a message type: */
CREATE type aq.Message_typ as object (
subject     VARCHAR2(30),
text        VARCHAR2(80));

/* Creating a object type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table        => 'aq.objmsgs80_qtab',
queue_payload_type => 'aq.Message_typ');
```

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name          => 'msg_queue',
queue_table         => 'aq.objmsgs80_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name          => 'msg_queue');
```

Example 2–2   Creating a Queue Table and Queue of Raw Type

```
/* Creating a RAW type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table           => 'aq.RawMsgs_qtab',
queue_payload_type    => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name           => 'raw_msg_queue',
queue_table          => 'aq.RawMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name           => 'raw_msg_queue');
```

Example 2–3   Creating a Prioritized Message Queue Table and Queue

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table         => 'aq.priority_msg',
sort_list           => 'PRIORITY,ENQ_TIME',
queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name          => 'priority_msg_queue',
queue_table         => 'aq.priority_msg');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name          => 'priority_msg_queue');
```

Example 2–4   Creating a Multiconsumer Queue Table and Queue

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table         => 'aq.MultiConsumerMsgs_qtab',
multiple_consumers => TRUE,
queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name          => 'msg_queue_multiple',
queue_table         => 'aq.MultiConsumerMsgs_qtab');
```

```
EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'msg_queue_multiple');
```

Example 2–5 Creating a Queue to Demonstrate Propagation

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name        => 'another_msg_queue',
queue_table       => 'aq.MultiConsumerMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'another_msg_queue');
```

Example 2–6 Setting Up Java Oracle Streams AQ Examples

```
CONNECT system/manager

DROP USER aqjava CASCADE;
GRANT CONNECT, RESOURCE, AQ_ADMINISTRATOR_ROLE TO aqjava IDENTIFIED BY aqjava;
GRANT EXECUTE ON DBMS_AQADM TO aqjava;
GRANT EXECUTE ON DBMS_AQ TO aqjava;
CONNECT aqjava/aqjava

/* Set up main class from which we will call subsequent examples and handle
   exceptions: */
import java.sql.*;
import oracle.AQ.*;

public class test_aqjava
{
   public static void main(String args[])
   {
      AQSession  aq_sess = null;
      try
      {
         aq_sess = createSession(args);

        /* now run the test: */
        runTest(aq_sess);
      }
      catch (Exception ex)
      {
         System.out.println("Exception-1: " + ex);
         ex.printStackTrace();
      }
```

```
   }
}
```

Example 2–7 Creating a Java Oracle Streams AQ Session for User 'aqjava'

```
public static AQSession createSession(String args[])
   {
      Connection db_conn;
      AQSession  aq_sess = null;

      try
      {

         Class.forName("oracle.jdbc.driver.OracleDriver");
         /* your actual hostname, port number, and SID will
         vary from what follows. Here we use 'dlsun736,' '5521,'
         and 'test,' respectively: */

         db_conn =
                  DriverManager.getConnection(
                  "jdbc:oracle:thin:@dlsun736:5521:test",
                  "aqjava", "aqjava");

         System.out.println("JDBC Connection opened ");
         db_conn.setAutoCommit(false);

         /* Load the Oracle8i AQ driver: */
         Class.forName("oracle.AQ.AQOracleDriver");

         /* Creating an AQ Session: */
         aq_sess = AQDriverManager.createAQSession(db_conn);
         System.out.println("Successfully created AQSession ");
      }
      catch (Exception ex)
      {
         System.out.println("Exception: " + ex);
         ex.printStackTrace();
      }
      return aq_sess;
   }
```

Example 2–8 Creating a Queue Table and Queue Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty     qtable_prop;
    AQQueueProperty          queue_prop;
    AQQueueTable             q_table;
    AQQueue                  queue;

    /* Creating a AQQueueTableProperty object (payload type - RAW): */
    qtable_prop = new AQQueueTableProperty("RAW");

    /* Creating a queue table called aq_table1 in aqjava schema: */
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table1", qtable_prop);
    System.out.println("Successfully created aq_table1 in aqjava schema");

    /* Creating a new AQQueueProperty object */
    queue_prop = new AQQueueProperty();

    /* Creating a queue called aq_queue1 in aq_table1: */
    queue = aq_sess.createQueue (q_table, "aq_queue1", queue_prop);
    System.out.println("Successfully created aq_queue1 in aq_table1");
}

/* Get a handle to an existing queue table and queue: */
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable            q_table;
    AQQueue                 queue;

    /* Get a handle to queue table - aq_table1 in aqjava schema: */
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table1");
    System.out.println("Successful getQueueTable");

    /* Get a handle to a queue - aq_queue1 in aqjava schema: */
    queue = aq_sess.getQueue ("aqjava", "aq_queue1");
    System.out.println("Successful getQueue");
}
```

Example 2–9 Creating a Queue and Starting Enqueue or Dequeue Using Java

```
{
     AQQueueTableProperty    qtable_prop;
     AQQueueProperty         queue_prop;
     AQQueueTable            q_table;
     AQQueue                 queue;
```

```
    /* Creating a AQQueueTable property object (payload type - RAW): */
    qtable_prop = new AQQueueTableProperty("RAW");
   qtable_prop.setCompatible("8.1");

    /* Creating a queue table called aq_table3 in aqjava schema: */
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table3", qtable_prop);
    System.out.println("Successful createQueueTable");

    /* Creating a new AQQueueProperty object: */
    queue_prop = new AQQueueProperty();

    /* Creating a queue called aq_queue3 in aq_table3: */
    queue = aq_sess.createQueue (q_table, "aq_queue3", queue_prop);
    System.out.println("Successful createQueue");

    /* Enable enqueue/dequeue on this queue: */
    queue.start();
    System.out.println("Successful start queue");

    /* Grant enqueue_any privilege on this queue to user scott: */
    queue.grantQueuePrivilege("ENQUEUE", "scott");
    System.out.println("Successful grantQueuePrivilege");
}
```

Example 2–10 Creating a Multiconsumer Queue and Adding Subscribers Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty    qtable_prop;
    AQQueueProperty         queue_prop;
    AQQueueTable            q_table;
    AQQueue                 queue;
    AQAgent                  subs1, subs2;

    /* Creating a AQQueueTable property object (payload type - RAW): */
    qtable_prop = new AQQueueTableProperty("RAW");
    System.out.println("Successful setCompatible");

    /* Set multiconsumer flag to true: */
    qtable_prop.setMultiConsumer(true);

    /* Creating a queue table called aq_table4 in aqjava schema: */
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table4", qtable_prop);
    System.out.println("Successful createQueueTable");
```

```
        /* Creating a new AQQueueProperty object: */
        queue_prop = new AQQueueProperty();
        /* Creating a queue called aq_queue4 in aq_table4 */
        queue = aq_sess.createQueue (q_table, "aq_queue4", queue_prop);
        System.out.println("Successful createQueue");

        /* Enable enqueue/dequeue on this queue: */
        queue.start();
        System.out.println("Successful start queue");

        /* Add subscribers to this queue: */
        subs1 = new AQAgent("GREEN", null, 0);
        subs2 = new AQAgent("BLUE", null, 0);

        queue.addSubscriber(subs1, null); /* no rule   */
        System.out.println("Successful addSubscriber 1");

        queue.addSubscriber(subs2, "priority < 2"); /* with rule */
        System.out.println("Successful addSubscriber 2");
    }
```

Enqueuing and Dequeuing Oracle Streams AQ Messages

You must set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \  code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=program name \
config=config file SQLCHECK=SEMANTICS userid=aq/aq
```

The following examples illustrate how to enqueue and dequeue Oracle Streams AQ messages:

- Enqueuing and Dequeuing Object Type Messages Using PL/SQL
- Enqueuing and Dequeuing Object Type Messages Using Pro*C/C++
- Enqueuing and Dequeuing Object Type Messages Using OCI

- Enqueuing and Dequeuing Object Type Messages (CustomDatum interface) Using Java
- Enqueuing and Dequeuing Object Type Messages (using SQLData interface) Using Java
- Enqueuing and Dequeuing RAW Type Messages Using PL/SQL
- Enqueuing and Dequeuing RAW Type Messages Using Pro*C/C++
- Enqueuing and Dequeuing RAW Type Messages Using OCI
- Enqueuing RAW Messages Using Java
- Dequeuing Messages Using Java
- Dequeuing Messages in Browse Mode Using Java
- Enqueuing and Dequeuing Messages by Priority Using PL/SQL
- Enqueuing Messages with Priority Using Java
- Dequeuing Messages after Preview by Criterion Using PL/SQL
- Enqueuing and Dequeuing Messages with Time Delay and Expiration Using PL/SQL
- Enqueuing and Dequeuing Messages by Correlation and Message ID Using Pro*C/C++
- Enqueuing and Dequeuing Messages by Correlation and Message ID Using OCI
- Enqueuing and Dequeuing Messages to/from a Multiconsumer Queue Using PL/SQL
- Enqueuing and Dequeuing Messages to/from a Multiconsumer Queue using OCI
- Enqueuing and Dequeuing Messages Using Message Grouping Using PL/SQL
- Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using PL/SQL
- Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using Java

Example 2–11 Enqueuing and Dequeuing Object Type Messages Using PL/SQL

To enqueue a single message without any other parameters specify the queue name and the payload.

```
/* Enqueue to msg_queue: */
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   message := message_typ('NORMAL MESSAGE',
   'enqueued to msg_queue first.');

   dbms_aq.enqueue(queue_name => 'msg_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);

   COMMIT;
END;

/* Dequeue from msg_queue: */
DECLARE
   dequeue_options     dbms_aq.dequeue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
           dequeue_options    => dequeue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                    ' ... ' || message.text );
   COMMIT;
END;
```

Example 2–12 Enqueuing and Dequeuing Object Type Messages Using Pro*C/C++

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>
/* The header file generated by processing
object type 'aq.Message_typ': */
#include  "pceg.h"

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
Message_typ     *message = (Message_typ*)0;  /* payload */
message_type_ind *imsg;              /*payload indicator*/
char             user[60]="aq/AQ";  /* user logon password */
char             subject[30];  /* components of the */
char             txt[80];      /* payload type */

/* ENQUEUE and DEQUEUE to an OBJECT QUEUE */

/*  Connect to database: */
EXEC SQL CONNECT :user;

/* On an oracle error print the error number :*/
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* Allocate memory for the host variable from the object cache : */
EXEC SQL ALLOCATE :message;

/* ENQUEUE */

strcpy(subject, "NORMAL ENQUEUE");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* Initialize the components of message : */
EXEC SQL OBJECT SET subject, text OF  :message TO :subject, :txt;
```

```
/* Embedded PLSQL call to the AQ enqueue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable 'message' to the payload: */
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message:imsg,    /* indicator must be specified */
msgid => msgid);
END;
END-EXEC;
/* Commit work */
EXEC SQL COMMIT;

printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* Dequeue */

/* Embedded PLSQL call to the AQ dequeue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Return the payload into the host variable 'message':  */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
 /* Commit work :*/
EXEC SQL COMMIT;

/* Extract the components of message: */
EXEC SQL OBJECT GET SUBJECT,TEXT FROM :message INTO :subject,:txt;
```

```
printf("Dequeued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);
}
```

Example 2–13 Enqueuing and Dequeuing Object Type Messages Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv       *envhp;
  OCIServer    *srvhp;
  OCIError     *errhp;
  OCISvcCtx    *svchp;
  dvoid        *tmp;
  OCIType      *mesg_tdo = (OCIType *) 0;
  message      msg;
  null_message nmsg;
  message      *mesg     = &msg;
  null_message *nmesg    = &nmsg;
  message      *deqmesg  = (message *)0;
  null_message *ndeqmesg = (null_message *)0;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );
```

```
OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit(&envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
    (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* Obtain TDO of message_typ */
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* Prepare the message payload */
mesg->subject = (OCIString *)0;
mesg->data = (OCIString *)0;
OCIStringAssignText(envhp, errhp,
                    (CONST text *)"NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
                     &mesg->subject);


OCIStringAssignText(envhp, errhp,
                    (CONST text *)"OCI ENQUEUE", strlen("OCI ENQUEUE"),
                    &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* Enqueue into the msg_queue */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* Dequeue from the msg_queue */
```

```
  OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
           mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Example 2–14 Enqueuing and Dequeuing Object Type Messages (CustomDatum interface) Using Java

To enqueue and dequeue **object type** messages follow the lettered steps:

a. Create the SQL type for the Queue Payload

```
connect aquser/aquser
create type ADDRESS as object (street VARCHAR (30), city VARCHAR(30));
create type PERSON as object (name VARCHAR (30), home ADDRESS);
```

b. Generate the java class that maps to the PERSON **ADT** and implements the CustomDatum interface (using Jpublisher tool)

```
jpub -user=aquser/aquser -sql=ADDRESS,PERSON -case=mixed -usertypes=oracle
-methods=false  -compatible=CustomDatum
```

This creates two classes, `PERSON.java` and `ADDRESS.java`, corresponding to the `PERSON` and `ADDRESS` ADT types.

c. Create the queue table and queue with ADT payload

d. Enqueue and dequeue messages containing object payloads

```
public static void AQObjectPayloadTest(AQSession aq_sess)
       throws AQException, SQLException, ClassNotFoundException
  {
    Connection           db_conn   = null;
    AQQueue              queue     = null;
    AQMessage            message   = null;
    AQObjectPayload      payload   = null;
    AQEnqueueOption      eq_option = null;
    AQDequeueOption      dq_option = null;
    PERSON           pers = null;
    PERSON           pers2= null;
    ADDRESS          addr = null;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();


    queue = aq_sess.getQueue("aquser", "test_queue2");
```

```
  /* Enable enqueue/dequeue on this queue */
  queue.start();

  /* Enqueue a message in test_queue2 */
  message = queue.createMessage();

  pers = new PERSON();
  pers.setName("John");
  addr = new ADDRESS();
  addr.setStreet("500 Easy Street");
  addr.setCity("San Francisco");
  pers.setHome(addr);

  payload = message.getObjectPayload();
  payload.setPayloadData(pers);
  eq_option = new AQEnqueueOption();

  /* Enqueue a message into test_queue2 */
  queue.enqueue(eq_option, message);

  db_conn.commit();

  /* Dequeue a message from test_queue2 */
  dq_option = new AQDequeueOption();
  message = ((AQOracleQueue)queue).dequeue(dq_option, PERSON.getFactory());

  payload = message.getObjectPayload();
  pers2 = (PERSON) payload.getPayloadData();

  System.out.println("Object data retrieved:  [PERSON]");
  System.out.println("Name:    " + pers2.getName());
  System.out.println("Address ");
  System.out.println("Street: " + pers2.getHome().getStreet());
  System.out.println("City:    " + pers2.getHome().getCity());

  db_conn.commit();
}
```

Example 2–15 Enqueuing and Dequeuing Object Type Messages (using SQLData interface) Using Java

To enqueue and dequeue object type messages follow the lettered steps:

a. Create the SQL type for the Queue Payload.

```
connect aquser/aquser
create type EMPLOYEE as object (empname VARCHAR (50), empno INTEGER);
```

b. Create a java class that maps to the EMPLOYEE ADT and implements the SQLData interface. This class can also be generated using JPublisher using the following syntax.

```
jpub -user=aquser/aquser -sql=EMPLOYEE -case=mixed -usertypes=jdbc
-methods=false

import java.sql.*;
import oracle.jdbc2.*;

public class Employee implements SQLData
{
  private String sql_type;
  public String empName;
  public int empNo;
  public Employee()
  {}
  public Employee (String sql_type, String empName, int empNo)
  {
    this.sql_type = sql_type;
    this.empName = empName;
    this.empNo = empNo;
  }

  ////// implements SQLData //////
  public String getSQLTypeName() throws SQLException
  { return sql_type;
  }
  public void readSQL(SQLInput stream, String typeName)
    throws SQLException
  {
    sql_type = typeName;
    empName = stream.readString();
    empNo = stream.readInt();
  }

  public void writeSQL(SQLOutput stream)
```

```
    throws SQLException
  {
    stream.writeString(empName);
    stream.writeInt(empNo);
  }

  public String toString()
  {
String ret_str = "";
    ret_str += "[Employee]\n";
    ret_str += "Name: " + empName + "\n";
    ret_str += "Number: " + empNo + "\n";

    return ret_str;
  }
}
```

c. Create the queue table and queue with ADT payload.

```
public static void createEmployeeObjQueue(AQSession aq_sess)
       throws AQException
  {
    AQQueueTableProperty qt_prop   = null;
    AQQueueProperty      q_prop    = null;
    AQQueueTable         q_table   = null;
    AQQueue              queue     = null;

    /* Message payload type is aquser.EMPLOYEE */
    qt_prop = new AQQueueTableProperty("AQUSER.EMPLOYEE");
    qt_prop.setComment("queue-table1");

    /* Creating aQTable1 */
    System.out.println("\nCreate QueueTable: [aqtable1]");
    q_table = aq_sess.createQueueTable("aquser", "aqtable1", qt_prop);

    /* Create test_queue1 */
    q_prop = new AQQueueProperty();
    queue = q_table.createQueue("test_queue1", q_prop);

    /* Enable enqueue/dequeue on this queue */
    queue.start();
  }
```

d. Enqueue and dequeue messages containing object payloads.

```
public static void AQObjectPayloadTest2(AQSession aq_sess)
       throws AQException, SQLException, ClassNotFoundException
  {
    Connection          db_conn   = null;
    AQQueue             queue     = null;
    AQMessage           message   = null;
    AQObjectPayload     payload   = null;
    AQEnqueueOption     eq_option = null;
    AQDequeueOption     dq_option = null;
    Employee            emp  = null;
    Employee            emp2 = null;
    Hashtable           map;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* Get the Queue object */
    queue = aq_sess.getQueue("aquser", "test_queue1");

    /* Register Employee class (corresponding to EMPLOYEE Adt)
     * in the connection type map
     */
    try
    {
      map  = (java.util.Hashtable)(((OracleConnection)db_conn).getTypeMap());
      map.put("AQUSER.EMPLOYEE", Class.forName("Employee"));
    }
    catch(Exception ex)
    {
      System.out.println("Error registering type: " + ex);
    }

    /* Enqueue a message in test_queue1 */
    message = queue.createMessage();
    emp = new Employee("AQUSER.EMPLOYEE", "Mark", 1007);

    /* Set the object payload */
    payload = message.getObjectPayload();
    payload.setPayloadData(emp);

    /* Enqueue a message into test_queue1*/
    eq_option = new AQEnqueueOption();
    queue.enqueue(eq_option, message);
    db_conn.commit();
```

```
    /* Dequeue a message from test_queue1 */
    dq_option = new AQDequeueOption();

    message = queue.dequeue(dq_option, Class.forName("Employee"));
    payload = message.getObjectPayload();
    emp2 = (Employee) payload.getPayloadData();
    System.out.println("\nObject data retrieved:  [EMPLOYEE]");
    System.out.println("Name  : " + emp2.empName);
    System.out.println("EmpId : " + emp2.empNo);

    db_conn.commit();
  }
```

Example 2–16   Enqueuing and Dequeuing RAW Type Messages Using PL/SQL

```
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             RAW(4096);

BEGIN
   message :=  HEXTORAW(RPAD('FF',4095,'FF'));
   DBMS_AQ.ENQUEUE(queue_name => 'raw_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
                     payload  => message,
                      msgid   => message_handle);

   COMMIT;
END;

/* Dequeue from raw_msg_queue: */
/* Dequeue from raw_msg_queue: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             RAW(4096);

BEGIN
   DBMS_AQ.DEQUEUE(queue_name => 'raw_msg_queue',
           dequeue_options    => dequeue_options,
```

```
            message_properties => message_properties,
            payload            => message,
            msgid              => message_handle);

   COMMIT;
END;
```

You must set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \  code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=program name \
config=config file SQLCHECK=SEMANTICS userid=aq/aq
```

Example 2–17 Enqueuing and Dequeuing RAW Type Messages Using Pro*C/C++

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
OCIEnv          *oeh;   /* OCI Env handle */
OCIError        *err;   /* OCI Err handle */
OCIRaw          *message= (OCIRaw*)0;   /* payload */
ub1             message_txt[100];   /* data for payload */
char            user[60]="aq/AQ";   /* user logon password */
int             status;   /* returns status of the OCI call */
```

```
/* Enqueue and dequeue to a RAW queue */

/* Connect to database: */
EXEC SQL CONNECT :user;

/* On an oracle error print the error number: */
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* Get the OCI Env handle: */
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
printf(" error in SQLEnvGet \n");
exit(1);
}
/* Get the OCI Error handle: */
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0))
{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}

/* Enqueue */
/* The bytes to be put into the raw payload:*/
strcpy(message_txt, "Enqueue to a Raw payload queue ");

/* Assign bytes to the OCIRaw pointer :
Memory must be allocated explicitly to OCIRaw*: */
if (status=OCIRawAssignBytes(oeh, err, message_txt, 100,
 &message))
{
printf(" error in  OCIRawAssignBytes  %d \n", status);
exit(1);
}

/*  Embedded PLSQL call to the AQ enqueue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable message to the raw payload: */
dbms_aq.enqueue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
```

```
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;

/* Dequeue */
/*  Embedded PLSQL call to the AQ dequeue procedure :*/
EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Return the raw payload into the host variable 'message':*/
dbms_aq.dequeue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
}
```

Example 2–18   Enqueuing and Dequeuing RAW Type Messages Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

int main()
{
  OCIEnv       *envhp;
  OCIServer    *srvhp;
  OCIError     *errhp;
  OCISvcCtx    *svchp;
  dvoid        *tmp;
  OCIType      *mesg_tdo = (OCIType *) 0;
  char         msg_text[100];
  OCIRaw       *mesg = (OCIRaw *)0;
```

```
OCIRaw       *deqmesg = (OCIRaw *)0;
OCIInd       ind = 0;
dvoid        *indptr = (dvoid *)&ind;
int          i;

OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* Obtain the TDO of the RAW data type */
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQADM", strlen("AQADM"),
              (CONST text *)"RAW", strlen("RAW"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* Prepare the message payload */
strcpy(msg_text, "Enqueue to a RAW queue");
OCIRawAssignBytes(envhp, errhp, msg_text, strlen(msg_text), &mesg);

/* Enqueue the message into raw_msg_queue */
OCIAQEnq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
        mesg_tdo, (dvoid **)&mesg, (dvoid **)&indptr, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* Dequeue the same message into C variable deqmesg */
OCIAQDeq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&indptr, 0, 0);
```

```
  for (i = 0; i < OCIRawSize(envhp, deqmesg); i++)
    printf("%c", *(OCIRawPtr(envhp, deqmesg) + i));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Example 2–19 Enqueuing RAW Messages Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTable             q_table;
     AQQueue                  queue;
     AQMessage                message;
     AQRawPayload             raw_payload;
     AQEnqueueOption          enq_option;
     String                   test_data = "new message";
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

     /* Get a handle to queue table - aq_table4 in aqjava schema: */
     q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
     System.out.println("Successful getQueueTable");

     /* Get a handle to a queue - aq_queue4 in aquser schema: */
     queue = aq_sess.getQueue ("aqjava", "aq_queue4");
     System.out.println("Successful getQueue");

     /* Creating a message to contain raw payload: */
     message = queue.createMessage();

     /* Get handle to the AQRawPayload object and populate it with raw data: */
     b_array = test_data.getBytes();

     raw_payload = message.getRawPayload();

     raw_payload.setStream(b_array, b_array.length);

     /* Creating a AQEnqueueOption object with default options: */
     enq_option = new AQEnqueueOption();
     /* Enqueue the message: */
     queue.enqueue(enq_option, message);

     db_conn.commit();
}
```

Example 2–20 Dequeuing Messages Using Java

```
   public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTable             q_table;
     AQQueue                  queue;
     AQMessage                message;
     AQRawPayload             raw_payload;
     AQEnqueueOption          enq_option;
     String                   test_data = "new message";
     AQDequeueOption          deq_option;
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

     /* Get a handle to queue table - aq_table4 in aqjava schema: */
     q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
     System.out.println("Successful getQueueTable");

     /* Get a handle to a queue - aq_queue4 in aquser schema: */
     queue = aq_sess.getQueue ("aqjava", "aq_queue4");
     System.out.println("Successful getQueue");

     /* Creating a message to contain raw payload: */
     message = queue.createMessage();

     /* Get handle to the AQRawPayload object and populate it with raw data: */
     b_array = test_data.getBytes();

     raw_payload = message.getRawPayload();

     raw_payload.setStream(b_array, b_array.length);

     /* Creating a AQEnqueueOption object with default options: */
     enq_option = new AQEnqueueOption();

     /* Enqueue the message: */
     queue.enqueue(enq_option, message);
     System.out.println("Successful enqueue");

     db_conn.commit();

     /* Creating a AQDequeueOption object with default options: */
     deq_option = new AQDequeueOption();
```

```
     /* Dequeue a message: */
     message = queue.dequeue(deq_option);
     System.out.println("Successful dequeue");

     /* Retrieve raw data from the message: */
     raw_payload = message.getRawPayload();

     b_array = raw_payload.getBytes();

     db_conn.commit();
}
```

Example 2–21 Dequeuing Messages in Browse Mode Using Java

```
   public static void runTest(AQSession aq_sess) throws AQException
{
      AQQueueTable             q_table;
      AQQueueTable             q_table;
      AQQueue                  queue;
      AQMessage                message;
      AQRawPayload             raw_payload;
      AQEnqueueOption          enq_option;
      String                   test_data = "new message";
      AQDequeueOption          deq_option;
      byte[]                    b_array;
      Connection               db_conn;

      db_conn = ((AQOracleSession)aq_sess).getDBConnection();

      /* Get a handle to queue table - aq_table4 in aqjava schema: */
      q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
      System.out.println("Successful getQueueTable");

      /* Get a handle to a queue - aq_queue4 in aquser schema: */
      queue = aq_sess.getQueue ("aqjava", "aq_queue4");
      System.out.println("Successful getQueue");

      /* Creating a message to contain raw payload: */
      message = queue.createMessage();

      /* Get handle to the AQRawPayload object and populate it with raw data: */
      b_array = test_data.getBytes();

      raw_payload = message.getRawPayload();
```

```
        raw_payload.setStream(b_array, b_array.length);

        /* Creating a AQEnqueueOption object with default options: */
        enq_option = new AQEnqueueOption();

        /* Enqueue the message: */
        queue.enqueue(enq_option, message);
        System.out.println("Successful enqueue");

        db_conn.commit();

        /* Creating a AQDequeueOption object with default options: */
        deq_option = new AQDequeueOption();

        /* Set dequeue mode to BROWSE: */
        deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_BROWSE);

        /* Set wait time to 10 seconds: */
        deq_option.setWaitTime(10);

        /* Dequeue a message: */
        message = queue.dequeue(deq_option);

        /* Retrieve raw data from the message: */
        raw_payload = message.getRawPayload();
        b_array = raw_payload.getBytes();

        String ret_value = new String(b_array);
        System.out.println("Dequeued message: " + ret_value);

        db_conn.commit();
}
```

Example 2–22 Enqueuing and Dequeuing Messages by Priority Using PL/SQL

When two messages are enqueued with the same priority, the message which was enqueued earlier is dequeued first. However, if two messages are of different priorities, then the message with the lower value (higher priority) is dequeued first.

```
/* Enqueue two messages with priority 30 and 5: */
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
```

```
   message             aq.message_typ;

BEGIN
   message := message_typ('PRIORITY MESSAGE',
   'enqued at priority 30.');

   message_properties.priority := 30;

   DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   message := message_typ('PRIORITY MESSAGE',
   'Enqueued at priority 5.');

   message_properties.priority := 5;

   DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);
END;

/* Dequeue from priority queue: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   DBMS_AQ.DEQUEUE(queue_name  => 'priority_msg_queue',
         dequeue_options       => dequeue_options,
         message_properties    => message_properties,
         payload               => message,
         msgid                 => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );

   COMMIT;
```

```
   DBMS_AQ.DEQUEUE(queue_name => 'priority_msg_queue',
         dequeue_options      => dequeue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );
   COMMIT;
END;

/* On return, the second message with priority set to 5 is retrieved before the
message with priority set to 30 because priority takes precedence over enqueue
time. */
```

Example 2–23 Enqueuing Messages with Priority Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTable             q_table;
     AQQueue                  queue;
     AQMessage                message;
     AQMessageProperty        m_property;
     AQRawPayload             raw_payload;
     AQEnqueueOption          enq_option;
     String                   test_data;
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

     /* Get a handle to queue table - aq_table4 in aqjava schema: */
     qtable = aq_sess.getQueueTable ("aqjava", "aq_table4");
     System.out.println("Successful getQueueTable");

     /* Get a handle to a queue - aq_queue4 in aqjava schema: */
     queue = aq_sess.getQueue ("aqjava", "aq_queue4");
     System.out.println("Successful getQueue");

     /* Enqueue 5 messages with priorities with different priorities: */
     for (int i = 0; i < 5; i++ )
       {
          /* Creating a message to contain raw payload: */
          message = queue.createMessage();
```

```
          test_data = "Small_message_" + (i+1);       /* some test data */

          /* Get a handle to the AQRawPayload object and
             populate it with raw data: */
          b_array = test_data.getBytes();

          raw_payload = message.getRawPayload();

          raw_payload.setStream(b_array, b_array.length);

          /* Set message priority: */
         m_property = message.getMessageProperty();

         if( i < 2)
            m_property.setPriority(2);
          else
             m_property.setPriority(3);

          /* Creating a AQEnqueueOption object with default options: */
          enq_option = new AQEnqueueOption();

          /* Enqueue the message: */
         queue.enqueue(enq_option, message);
         System.out.println("Successful enqueue");
     }

     db_conn.commit();
}
```

Example 2–24 Dequeuing Messages after Preview by Criterion Using PL/SQL

An application can preview messages in browse mode or locked mode without deleting the message. The message of interest can then be removed from the queue.

```
/* Enqueue 6 messages to msg_queue
— GREEN, GREEN, YELLOW, VIOLET, BLUE, RED */

DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   message := message_typ('GREEN',
```

```
'GREEN enqueued to msg_queue first.');

DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);

message := message_typ('GREEN',
'GREEN also enqueued to msg_queue second.');

DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);

message := message_typ('YELLOW',
'YELLOW enqueued to msg_queue third.');

DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);

DBMS_OUTPUT.PUT_LINE ('Message handle: ' || message_handle);

message := message_typ('VIOLET',
'VIOLET enqueued to msg_queue fourth.');

DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);

message := message_typ('BLUE',
'BLUE enqueued to msg_queue fifth.');

DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);
```

```
   message := message_typ('RED',
   'RED enqueued to msg_queue sixth.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);

   COMMIT;
END;

/* Dequeue in BROWSE mode until RED is found,
and remove RED from queue: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;

   LOOP
      DBMS_AQ.DEQUEUE(queue_name          => 'msg_queue',
                      dequeue_options    => dequeue_options,
                      message_properties => message_properties,
                      payload            => message,
                      msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                   ' ... ' || message.text );

      EXIT WHEN message.subject = 'RED';

   END LOOP;

   dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
   dequeue_options.msgid        := message_handle;

   DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
           dequeue_options    => dequeue_options,
           message_properties => message_properties,
           payload            => message,
```

```
           msgid              => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );

   COMMIT;
END;

/* Dequeue in LOCKED mode until BLUE is found,
and remove BLUE from queue: */
DECLARE
dequeue_options     dbms_aq.dequeue_options_t;
message_properties  dbms_aq.message_properties_t;
message_handle      RAW(16);
message             aq.message_typ;

BEGIN
dequeue_options.dequeue_mode := dbms_aq.LOCKED;

      LOOP

dbms_aq.dequeue(queue_name => 'msg_queue',
                 dequeue_options    => dequeue_options,
                 message_properties => message_properties,
                 payload            => message,
                 msgid              => message_handle);

dbms_output.put_line ('Message: ' || message.subject ||
          ' ... ' || message.text );

EXIT WHEN message.subject = 'BLUE';
      END LOOP;

dequeue_options.dequeue_mode := dbms_aq.REMOVE;
dequeue_options.msgid        := message_handle;

dbms_aq.dequeue(queue_name => 'msg_queue',
dequeue_options      => dequeue_options,
message_properties   => message_properties,
payload              => message,
msgid => message_handle);

DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
' ... ' || message.text );
```

```
      COMMIT;
END;
```

Expiration is calculated from the earliest dequeue time. So, if an application wants a message to be dequeued no earlier than a week from now, but no later than 3 weeks from now, then this requires setting the expiration time for 2 weeks. This scenario is described in the following code segment.

Example 2–25 Enqueuing and Dequeuing Messages with Time Delay and Expiration Using PL/SQL

```
/* Enqueue message for delayed availability: */
DECLARE
enqueue_options     dbms_aq.enqueue_options_t;
message_properties  dbms_aq.message_properties_t;
message_handle      RAW(16);
message             aq.Message_typ;

BEGIN
message := Message_typ('DELAYED',
'This 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 => 'msg_queue',
enqueue_options      => enqueue_options,
message_properties   => message_properties,
payload            => message,
msgid              => message_handle);

      COMMIT;
END;
```

You must set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \  code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=program name \
config=config file SQLCHECK=SEMANTICS userid=aq/aq
```

Example 2–26 Enqueuing and Dequeuing Messages by Correlation and Message ID Using Pro*C/C++

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>
/* The header file generated by processing
object type 'aq.Message_typ': */
#include  "pceg.h"

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
OCIEnv            *oeh;  /* OCI Env Handle */
OCIError          *err;  /* OCI Error Handle */
Message_typ      *message = (Message_typ*)0; /* queue  payload */
message_type_ind *imsg;              /*payload indicator*/
OCIRaw            *msgid = (OCIRaw*)0;  /* message id */
ub1               msgmem[16]="";  /* memory for msgid */
char              user[60]="aq/AQ";  /* user login password */
char              subject[30];  /* components of */
char              txt[80];  /* Message_typ */
char              correlation1[30];  /* message correlation  */
char              correlation2[30];
int               status;  /* code returned by the OCI calls */

/* Dequeue by correlation and msgid */

/* Connect to the database: */
EXEC SQL CONNECT :user;
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* Allocate space in the object cache for the host variable: */
EXEC SQL ALLOCATE :message;
```

```
/* Get the OCI Env handle: */
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
 printf(" error in SQLEnvGet \n");
 exit(1);
}
/* Get the OCI Error handle: */
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0))
{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}


/* Assign memory for msgid:
Memory must be allocated explicitly to OCIRaw*: */
if (status=OCIRawAssignBytes(oeh, err, msgmem, 16, &msgid))
{
printf(" error in  OCIRawAssignBytes  %d \n", status);
exit(1);
}

/* First enqueue */

strcpy(correlation1, "1st message");
strcpy(subject, "NORMAL ENQUEUE1");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* Initialize the components of message: */
EXEC SQL OJECT SET subject, text OF :message TO :subject, :txt;

/* Embedded PLSQL call to the AQ enqueue procedure: */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
BEGIN
/* Bind the host variable 'correlation1': to message correlation*/
message_properties.correlation := :correlation1;

/* Bind the host variable 'message' to payload and
 return message ID into host variable 'msgid': */
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
```

```
enqueue_options => enqueue_options,
payload => :message:imsg,    /* indicator must be specified */
msgid => :msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;

printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* Second enqueue */

strcpy(correlation2, "2nd message");
strcpy(subject, "NORMAL ENQUEUE2");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* Initialize the components of message: */
EXEC SQL OBJECT SET subject, text OF :messsage TO :subject,:txt;

/* Embedded PLSQL call to the AQ enqueue procedure: */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable 'correlation2':  to message correlaiton */
message_properties.correlation := :correlation2;

/* Bind the host variable 'message': to payload */
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);
```

```
/* First dequeue - by  correlation */

EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Dequeue by  correlation in host variable 'correlation2': */
dequeue_options.correlation := :correlation2;

/* Return the payload into host variable 'message': */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work : */
EXEC SQL COMMIT;

/* Extract the values of the components of message: */
EXEC SQL OBJECT GET subject, text FROM :message INTO :subject,:txt;

printf("Dequeued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* SECOND DEQUEUE - by MSGID  */

EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Dequeue by msgid in host variable 'msgid': */
dequeue_options.msgid := :msgid;

/* Return the payload into host variable 'message':  */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
```

```
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
}
```

Example 2–27 Enqueuing and Dequeuing Messages by Correlation and Message ID Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv      *envhp;
  OCIServer   *srvhp;
  OCIError    *errhp;
  OCISvcCtx   *svchp;
  dvoid       *tmp;
  OCIType     *mesg_tdo = (OCIType *) 0;
  message     msg;
  null_message nmsg;
  message     *mesg     = &msg;
  null_message *nmesg    = &nmsg;
  message     *deqmesg  = (message *)0;
  null_message *ndeqmesg = (null_message *)0;
```

```
OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* Obtain TDO of message_typ */
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* Prepare the message payload */
mesg->subject = (OCIString *)0;
mesg->data = (OCIString *)0;
OCIStringAssignText(envhp, errhp,
                    (CONST text *)"NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
                    &mesg->subject);
OCIStringAssignText(envhp, errhp,
                    (CONST text *)"OCI ENQUEUE", strlen("OCI ENQUEUE"),
                    &mesg->data);
nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* Enqueue into the msg_queue */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
OCITransCommit(svchp, errhp, (ub4) 0);

/* Dequeue from the msg_queue */
```

```
  OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
           mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Example 2–28   Enqueuing and Dequeuing Messages to/from a Multiconsumer Queue Using PL/SQL

```
/* Create subscriber list: */
DECLARE
   subscriber aq$_agent;

   /* Add subscribers RED and GREEN to the suscriber list: */
BEGIN
   subscriber := aq$_agent('RED', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
   subscriber => subscriber);

   subscriber := aq$_agent('GREEN', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
   subscriber => subscriber);
END;

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             aq.message_typ;

   /* Enqueue MESSAGE 1 for subscribers to the queue.
   BEGIN
   message := message_typ('MESSAGE 1',
   'This message is queued for queue subscribers.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
   enqueue_options    => enqueue_options,
   message_properties => message_properties,
   payload            => message,
   msgid              => message_handle);

   /* Enqueue MESSAGE 2 for specified recipients.*/
   message := message_typ('MESSAGE 2',
```

```
   'This message is queued for two recipients.');
   recipients(1) := aq$_agent('RED', NULL, NULL);
   recipients(2) := aq$_agent('BLUE', NULL, NULL);
   message_properties.recipient_list := recipients;

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   COMMIT;
END;
```

RED is both a subscriber to the queue, as well as being a specified recipient of MESSAGE 2. By contrast, GREEN is only a subscriber to those messages in the queue (in this case, MESSAGE) for which no recipients have been specified. BLUE, while not a subscriber to the queue, is nevertheless specified to receive MESSAGE 2.

```
/* Dequeue messages from msg_queue_multiple: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;
   no_messages         exception;
   pragma exception_init (no_messages, -25228);

BEGIN

   dequeue_options.wait := DBMS_AQ.NO_WAIT;
   BEGIN
   /* Consumer BLUE will get MESSAGE 2: */
   dequeue_options.consumer_name := 'BLUE';
   dequeue_options.navigation := FIRST_MESSAGE;

   LOOP

   DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
             dequeue_options    => dequeue_options,
             message_properties => message_properties,
             payload            => message,
             msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
```

```
            ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;

   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
   DBMS_OUTPUT.PUT_LINE ('No more messages for BLUE');
   COMMIT;
END;

BEGIN
/* Consumer RED will get MESSAGE 1 and MESSAGE 2: */
   dequeue_options.consumer_name := 'RED';
dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE
   LOOP
      DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
                dequeue_options    => dequeue_options,
                message_properties => message_properties,
                payload            => message,
                msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                    ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;
   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
      DBMS_OUTPUT.PUT_LINE ('No more messages for RED');
   COMMIT;
END;

BEGIN
   /* Consumer GREEN will get MESSAGE 1: */
   dequeue_options.consumer_name := 'GREEN';
   dequeue_options.navigation := FIRST_MESSAGE;
   LOOP
      DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
                dequeue_options    => dequeue_options,
                message_properties => message_properties,
                payload            => message,
                msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
          ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;
```

```
   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
      DBMS_OUTPUT.PUT_LINE ('No more messages for GREEN');
   COMMIT;
END;
```

You must set up the following data structures for certain examples to work:

```
CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
 queue_table        => 'aq.qtable_multi',
   multiple_consumers => true,
   queue_payload_type => 'aq.message_typ');
EXECUTE DBMS_AQADM.START_QUEUE('aq.msg_queue_multiple');
CONNECT aq/aq
```

Example 2–29 Enqueuing and Dequeuing Messages to/from a Multiconsumer Queue using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv             *envhp;
  OCIServer          *srvhp;
  OCIError           *errhp;
  OCISvcCtx          *svchp;
```

```
dvoid                  *tmp;
OCIType                *mesg_tdo = (OCIType *) 0;
message                msg;
null_message           nmsg;
message                *mesg = &msg;
null_message           *nmesg = &nmsg;
message                *deqmesg = (message *)0;
null_message           *ndeqmesg = (null_message *)0;
OCIAQMsgProperties     *msgprop = (OCIAQMsgProperties *)0;
OCIAQAgent             *agents[2];
OCIAQDeqOptions        *deqopt = (OCIAQDeqOptions *)0;
ub4                    wait = OCI_DEQ_NO_WAIT;
ub4                    navigation = OCI_DEQ_FIRST_MSG;

OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );



OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* Obtain TDO of message_typ */
OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
             (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
             (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* Prepare the message payload */
```

```
 mesg->subject = (OCIString *)0;
 mesg->data = (OCIString *)0;
 OCIStringAssignText(envhp, errhp,
                     (CONST text *)"MESSAGE 1", strlen("MESSAGE 1"),
                     &mesg->subject);
 OCIStringAssignText(envhp, errhp,
                    (CONST text *)"mesg for queue subscribers",
                    strlen("mesg for queue subscribers"), &mesg->data);
 nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

 /* Enqueue MESSAGE 1 for subscribers to the queue. */
 OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, 0,
          mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

 /* Enqueue MESSAGE 2 for specified recipients. */
 /* prepare message payload */
OCIStringAssignText(envhp, errhp,
                     (CONST text *)"MESSAGE 2", strlen("MESSAGE 2"),
                     &mesg->subject);
 OCIStringAssignText(envhp, errhp,
      (CONST text *)"mesg for two recipients",
      strlen("mesg for two recipients"), &mesg->data);

 /* Allocate AQ message properties and agent descriptors */
 OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
                    OCI_DTYPE_AQMSG_PROPERTIES, 0, (dvoid **)0);
 OCIDescriptorAlloc(envhp, (dvoid **)&agents[0],
                    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
 OCIDescriptorAlloc(envhp, (dvoid **)&agents[1],
                    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);

 /* Prepare the recipient list, RED and BLUE */
 OCIAttrSet(agents[0], OCI_DTYPE_AQAGENT, "RED", strlen("RED"),
            OCI_ATTR_AGENT_NAME, errhp);
 OCIAttrSet(agents[1], OCI_DTYPE_AQAGENT, "BLUE", strlen("BLUE"),
            OCI_ATTR_AGENT_NAME, errhp);
 OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES, (dvoid *)agents, 2,
            OCI_ATTR_RECIPIENT_LIST, errhp);

 OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, msgprop,
          mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

 OCITransCommit(svchp, errhp, (ub4) 0);

 /* Now dequeue the messages using different consumer names */
```

```
/* Allocate dequeue options descriptor to set the dequeue options */
OCIDescriptorAlloc(envhp, (dvoid **)&deqopt, OCI_DTYPE_AQDEQ_OPTIONS, 0,
                   (dvoid **)0);

/* Set wait parameter to NO_WAIT so that the dequeue returns immediately */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&wait, 0,
           OCI_ATTR_WAIT, errhp);

/* Set navigation to FIRST_MESSAGE so that the dequeue resets the position */
/* after a new consumer_name is set in the dequeue options          */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp);

/* Dequeue from the msg_queue_multiple as consumer BLUE */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"BLUE", strlen("BLUE"),
           OCI_ATTR_CONSUMER_NAME, errhp);

while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
                mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
                == OCI_SUCCESS)
{
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* Dequeue from the msg_queue_multiple as consumer RED */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"RED", strlen("RED"),
      OCI_ATTR_CONSUMER_NAME, errhp);
while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
  mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
  == OCI_SUCCESS)
{
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
}
OCITransCommit(svchp, errhp, (ub4) 0);

/* Dequeue from the msg_queue_multiple as consumer GREEN */
OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,(dvoid *)"GREEN",strlen("GREEN"),
      OCI_ATTR_CONSUMER_NAME, errhp);
while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
  mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
  == OCI_SUCCESS)
{
```

```
    printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  }
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Example 2–30 Enqueuing and Dequeuing Messages Using Message Grouping Using PL/SQL

```
CONNECT aq/aq

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table          => 'aq.msggroup',
   queue_payload_type   => 'aq.message_typ',
   message_grouping     => DBMS_AQADM.TRANSACTIONAL);

EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name      => 'msggroup_queue',
   queue_table     => 'aq.msggroup');

EXECUTE DBMS_AQADM.START_QUEUE(
   queue_name => 'msggroup_queue');

/* Enqueue three messages in each transaction */
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN

  /* Loop through three times, committing after every iteration */
  FOR txnno in 1..3 LOOP

    /* Loop through three times, enqueuing each iteration */
    FOR mesgno in 1..3 LOOP
      message := message_typ('GROUP#' || txnno,
                 'Message#' || mesgno ||  ' in group' || txnno);

      DBMS_AQ.ENQUEUE(queue_name         => 'msggroup_queue',
                  enqueue_options        => enqueue_options,
                  message_properties     => message_properties,
                  payload                => message,
                  msgid                  => message_handle);
```

```
    END LOOP;
    /* Commit the transaction */
    COMMIT;
  END LOOP;
END;

/* Now dequeue the messages as groups */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.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   => 'msggroup_queue',
                dequeue_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;

     EXCEPTION
       WHEN end_of_group THEN
         DBMS_OUTPUT.PUT_LINE ('Finished processing a group of messages');
         COMMIT;
         dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
     END;
   END LOOP;
   EXCEPTION
     WHEN no_messages THEN
```

```
      DBMS_OUTPUT.PUT_LINE ('No more messages');
END;
```

Example 2–31 Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using PL/SQL

```
/* Create the message payload object type with one or more LOB attributes. On
   enqueue, set the LOB attribute to EMPTY_BLOB. After the enqueue completes,
   before you commit your transaction. Select the LOB attribute from the
   user_data column of the queue table or queue table view. You can now
   use the LOB interfaces (which are available through both OCI and PL/SQL) to
   write the LOB data to the queue.  On dequeue, the message payload
   will contain the LOB locator. You can use this LOB locator after
   the dequeue, but before you commit your transaction, to read the LOB data.
 */
/* Setup the accounts: */

connect system/manager

CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT aq_administrator_role TO aqadm;

CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON DBMS_AQ TO aq;
CREATE TYPE aq.message AS OBJECT(id      NUMBER,
                                 subject VARCHAR2(100),
                                 data    BLOB,
                                 trailer NUMBER);
CREATE TABLESPACE aq_tbs DATAFILE 'aq.dbs' SIZE 2M REUSE;


/* create the queue table, queues and start the queue: */

CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'aq.qt1',
   queue_payload_type => 'aq.message');
EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name  => 'aq.queue1',
   queue_table =>  'aq.qt1');
EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'aq.queue1');

/* End set up: */
```

```
/* Enqueue Large data types: */

CONNECT aq/aq
CREATE OR REPLACE PROCEDURE blobenqueue(msgno IN NUMBER) AS
enq_userdata aq.message;
enq_msgid    RAW(16);
enqopt       DBMS_AQ.enqueue_options_t;
msgprop      DBMS_AQ.message_properties_t;
lob_loc      BLOB;
buffer       RAW(4096);

BEGIN

   buffer := HEXTORAW(RPAD('FF', 4096, 'FF'));
   enq_userdata := aq.message(msgno, 'Large Lob data', EMPTY_BLOB(), msgno);
   DBMS_AQ.ENQUEUE('aq.queue1', enqopt, msgprop, enq_userdata, enq_msgid);

   --select the lob locator for the queue table
   SELECT t.user_data.data INTO lob_loc
      FROM qt1 t
      WHERE t.msgid = enq_msgid;

   DBMS_LOB.WRITE(lob_loc, 2000, 1, buffer );
   COMMIT;
END;

/* Dequeue lob data: */

CREATE OR REPLACE PROCEDURE blobdequeue AS
   dequeue_options    DBMS_AQ.dequeue_options_t;
   message_properties DBMS_AQ.message_properties_t;
   mid                RAW(16);
   pload              aq.message;
   lob_loc            BLOB;
   amount             BINARY_INTEGER;
   buffer             RAW(4096);

BEGIN
   DBMS_AQ.DEQUEUE('aq.queue1', dequeue_options, message_properties,
                   pload, mid);
   lob_loc := pload.data;

   -- read the lob data info buffer
   amount := 2000;
```

```
   DBMS_LOB.READ(lob_loc, amount, 1, buffer);
   DBMS_OUTPUT.PUT_LINE('Amount of data read: '||amount);
   COMMIT;
END;

/* Do the enqueues and dequeues: */
 SET SERVEROUTPUT ON

BEGIN
   FOR i IN 1..5 LOOP
      blobenqueue(i);
   END LOOP;
END;

BEGIN
   FOR i IN 1..5 LOOP
     blobdequeue();
   END LOOP;
END;
```

Example 2–32 Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using Java

1. Create the message type (ADT with **CLOB** and **BLOB**).

```
connect aquser/aquser

create type LobMessage as object(id        NUMBER,
                              subject   varchar2(100),
                              data      blob,
                              cdata     clob,
                              trailer   number);
```

2. Create the queue table and queue.

```
connect aquser/aquser
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
          queue_table => 'qt_adt',
          queue_payload_type => 'LOBMESSAGE',
          comment => 'single-consumer, default sort ordering, ADT Message',
          compatible => '8.1.0'
        );

EXECUTE DBMS_AQADM.CREATE_QUEUE(
          queue_name  => 'q1_adt',
          queue_table => 'qt_adt'
```

```
        );

EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'q1_adt');
```

3. Run jpublisher to generate the java class that maps to the LobMessage.

```
   Oracle object type

   jpub -user=aquser/aquser -sql=LobMessage -case=mixed -methods=false
-usertypes=oracle -compatible=CustomDatum
```

4. Enqueue and dequeue messages.

Oracle Streams AQ Propagation

The following examples illustrate Oracle Streams AQ propagation:

- Enqueuing Messages for Remote Subscribers or Recipients to a Multiconsumer Queue and Propagation Scheduling Using PL/SQL
- Managing Propagation From One Queue To Other Queues In the Same Database Using PL/SQL
- Managing Propagation from One Queue to Other Queues In Another Database Using PL/SQL
- Unscheduling Propagation Using PL/SQL

Caution: You must create queues or queue tables, or start or enable queues, for certain examples to work.

Example 2–33 Enqueuing Messages for Remote Subscribers or Recipients to a Multiconsumer Queue and Propagation Scheduling Using PL/SQL

```
/* Create subscriber list: */
DECLARE
   subscriber aq$_agent;

   /* Add subscribers RED and GREEN with different addresses  to the suscriber
   list: */
BEGIN
   BEGIN
      /* Add subscriber RED that will dequeue messages from another_msg_queue
      queue in the same datatbase */
      subscriber := aq$_agent('RED', 'another_msg_queue', NULL);
```

```
      DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
      subscriber => subscriber);

      /* Schedule propagation from msg_queue_multiple to other queues in the
      same database: */
      DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple');

      /* Add subscriber GREEN that will dequeue messages from the msg_queue
      queue in another database reached by the database link another_db.world */
      subscriber := aq$_agent('GREEN', 'msg_queue@another_db.world', NULL);
      DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
      subscriber => subscriber);

   /* Schedule propagation from msg_queue_multiple to other queues in the
      database "another_database": */
   END;
   BEGIN
      DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple',
      destination  => 'another_db.world');
   END;
END;

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             aq.message_typ;

/* Enqueue MESSAGE 1 for subscribers to the queue. */
BEGIN
   message := message_typ('MESSAGE 1',
   'This message is queued for queue subscribers.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   /* Enqueue MESSAGE 2 for specified recipients.*/
   message := message_typ('MESSAGE 2',
   'This message is queued for two recipients.');
   recipients(1) := aq$_agent('RED', 'another_msg_queue', NULL);
   recipients(2) := aq$_agent('BLUE', NULL, NULL);
```

```
   message_properties.recipient_list := recipients;

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   COMMIT;
END;
```

> **Note:** `RED` at address `another_msg_queue` is both a subscriber to the queue, as well as being a specified recipient of `MESSAGE 2`. By contrast, `GREEN` at address `msg_queue@another_db.world` is only a subscriber to those messages in the queue (in this case, `MESSAGE 1`) for which no recipients have been specified. `BLUE`, while not a subscriber to the queue, is nevertheless specified to receive `MESSAGE 2`.

Example 2–34 Managing Propagation From One Queue To Other Queues In the Same Database Using PL/SQL

```
/* Schedule propagation from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'q1def');

/* Disable propagation from queue q1def to other queues in the same
database */
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   queue_name => 'q1def');

/* Alter schedule from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name => 'q1def',
   duration   => '2000',
   next_time  => 'SYSDATE + 3600/86400',
   latency    => '32');

/* Enable propagation from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   queue_name => 'q1def');

/* Unschedule propagation from queue q1def to other queues in the same database
```

```
*/
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name => 'q1def');
```

Example 2–35 Managing Propagation from One Queue to Other Queues In Another Database Using PL/SQL

```
/* Schedule propagation from queue q1def to other queues in another database
reached by the database link another_db.world  */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
   queue_name   => 'q1def',
   destination  => 'another_db.world');

/* Disable propagation from queue q1def to other queues in another database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world');

/* Alter schedule from queue q1def to other queues in another database reached
by the database link another_db.world  */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world',
   duration    => '2000',
   next_time   => 'SYSDATE +  3600/86400',
   latency     => '32');

/* Enable propagation from queue q1def to other queues in another database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world');

/* Unschedule propagation from queue q1def to other queues in another database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name  => 'q1def',
   destination => 'another_db.world');
```

Example 2–36 Unscheduling Propagation Using PL/SQL

```
/* Unschedule propagation from msg_queue_multiple to the destination
another_db.world */
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name => 'msg_queue_multiple',
   destination => 'another_db.world');
```

Dropping Oracle Streams AQ Objects

The following example illustrates how to drop Oracle Streams AQ objects.

Caution: You must create queues or queue tables, or start, stop, or enable queues, for certain examples to work.

Example 2–37 Dropping Oracle Streams AQ Objects

```
/* Cleans up all objects related to the object type: */
CONNECT aq/aq

EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.objmsgs80_qtab');

/* Cleans up all objects related to the RAW type: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.RawMsgs_qtab');

/* Cleans up all objects related to the priority queue: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name     => 'priority_msg_queue');
```

```
EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name     => 'priority_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table   => 'aq.priority_msg');

/* Cleans up all objects related to the multiple-consumer queue: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name  => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name  => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.MultiConsumerMsgs_qtab');

DROP TYPE aq.message_typ;
```

Revoking Roles and Privileges

The following example illustrates how to revoke roles and privileges in Oracle Streams AQ.

Example 2–38 Revoking Roles and Privileges in Oracle Streams AQ

Assume user `tkaqusr` has enqueue privilege on a queue `tkaqusr_q1`. Then an example of revoke would be:

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE('ENQUEUE', 'tkaqusr_q, 'tkaqusr');
```

Deploying Oracle Streams AQ with XA

You must set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table => 'aq.qtable',
   queue_payload_type => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name => 'aq.aqsqueue',
   queue_table => 'aq.qtable');

EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'aq.aqsqueue');
```

The following example illustrates how to deploy Oracle Streams AQ with XA.

Example 2–39 Deploying Oracle Streams AQ with XA

```
/*
 * The program uses the XA interface to enqueue 100 messages and then
 * dequeue them.
 * Login: aq/aq
 * Requires: AQ_USER_ROLE to be granted to aq
 *        a RAW queue called "aqsqueue" to be created in aqs schema
 *        (preceding steps can be performed by running aqaq.sql)
 * Message Format: Msgno: [0-1000] HELLO, WORLD!
 * Author: schandra@us.oracle.com
 */

#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <xa.h>

/* XA open string */
char xaoinfo[] =  "oracle_xa+ACC=P/AQ/AQ+SESTM=30+Objects=T";

/* template for generating XA XIDs */
XID  xidtempl = { 0x1e0a0a1e, 12, 8, "GTRID001BQual001" };

/* Pointer to Oracle XA function table */
extern struct xa_switch_t xaosw;                         /* Oracle XA switch */
static struct xa_switch_t *xafunc = &xaosw;

/* dummy stubs for ax_reg and ax_unreg */
int ax_reg(rmid, xid, flags)
int  rmid;
XID *xid;
```

```
long flags;
{
  xid->formatID = -1;
  return 0;
}

int ax_unreg(rmid, flags)
int     rmid;
long    flags;
{
  return 0;
}

/* generate an XID */
void xidgen(xid, serialno)
XID *xid;
int  serialno;
{
  char seq [11];

  sprintf(seq, "%d", serialno);
  memcpy((void *)xid, (void *)&xidtempl, sizeof(XID));
  strncpy((&xid->data[5]), seq, 3);
}

/* check if XA operation succeeded */
#define checkXAerr(action, funcname)    \
    if ((action) != XA_OK)        \
    {                   \
      printf("%s failed!\n", funcname);   \
      exit(-1);            \
    } else

/* check if OCI operation succeeded */
static void checkOCIerr(errhp, status)
OCIError *errhp;
sword      status;
{
  text errbuf[512];
  ub4 buflen;
  sb4 errcode;

  if (status == OCI_SUCCESS) return;

  if (status == OCI_ERROR)
```

```
  {
    OCIErrorGet((dvoid *) errhp, 1, (text *)0, &errcode, errbuf,
      (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
  }
  else
    printf("Error - %d\n", status);
  exit (-1);
}


void main(argc, argv)
int    argc;
char **argv;
{
  int         msgno = 0;             /* message being enqueued */
  OCIEnv       *envhp;                /* OCI environment handle */
  OCIError     *errhp;                 /* OCI Error handle */
  OCISvcCtx    *svchp;                    /* OCI Service handle */
  char      message[128];                /* message buffer */
  ub4      mesglen;             /* length of message */
  OCIRaw       *rawmesg = (OCIRaw *)0;       /* message in OCI RAW format */
  OCIInd      ind = 0;                 /* OCI null indicator */
  dvoid          *indptr = (dvoid *)&ind;       /* null indicator pointer */
  OCIType      *mesg_tdo = (OCIType *) 0;         /* TDO for RAW datatype */
  XID      xid;                 /* XA's global transaction id */
  ub4      i;                      /* array index */

  checkXAerr(xafunc->xa_open_entry(xaoinfo, 1, TMNOFLAGS), "xaoopen");

  svchp = xaoSvcCtx((text *)0);           /* get service handle from XA */
  envhp = xaoEnv((text *)0);          /* get enviornment handle from XA */

  if (!svchp || !envhp)
  {
    printf("Unable to obtain OCI Handles from XA!\n");
    exit (-1);
  }

  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&errhp,
       OCI_HTYPE_ERROR, 0, (dvoid **)0);  /* allocate error handle */

  /* enqueue 1000 messages, 1 message for each XA transaction */
  for (msgno = 0; msgno < 1000; msgno++)
  {
```

```
    sprintf((const char *)message, "Msgno: %d, Hello, World!", msgno);
    mesglen = (ub4)strlen((const char *)message);
    xidgen(&xid, msgno);                 /* generate an XA xid */

    checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");

    checkOCIerr(errhp, OCIRawAssignBytes(envhp, errhp, (ub1 *)message, mesglen,
                &rawmesg));

    if (!mesg_tdo)          /* get Type descriptor (TDO) for RAW type */
      checkOCIerr(errhp, OCITypeByName(envhp, errhp, svchp,
                        (CONST text *)"AQADM", strlen("AQADM"),
                         (CONST text *)"RAW", strlen("RAW"),
                     (text *)0, 0, OCI_DURATION_SESSION,
                     OCI_TYPEGET_ALL, &mesg_tdo));

    checkOCIerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"aqsqueue",
               0, 0, mesg_tdo, (dvoid **)&rawmesg, &indptr,
            0, 0));

    checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaoend");
    checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
    printf("%s Enqueued\n", message);
  }

  /* dequeue 1000 messages within one XA transaction */
  xidgen(&xid, msgno);                           /* generate an XA xid */
  checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");
  for (msgno = 0; msgno < 1000; msgno++)
  {
    checkOCIerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"aqsqueue",
             0, 0, mesg_tdo, (dvoid **)&rawmesg, &indptr,
            0, 0));
    if (ind)
      printf("Null Raw Message");
    else
      for (i = 0; i < OCIRawSize(envhp, rawmesg); i++)
   printf("%c", *(OCIRawPtr(envhp, rawmesg) + i));
    printf("\n");

  }
  checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaoend");
  checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
}
```

Oracle Streams AQ and Memory Usage

You must set up the following data structures for certain examples to work:

```
/* Create_types.sql */
CONNECT system/manager
GRANT AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE TO scott;
CONNECT scott/tiger
CREATE TYPE MESSAGE AS OBJECT (id NUMBER, data VARCHAR2(80));
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'qt',
   queue_payload_type => 'message');
EXECUTE DBMS_AQADM.CREATE_QUEUE('msgqueue', 'qt');
EXECUTE DBMS_AQADM.START_QUEUE('msgqueue');
```

The following examples illustrate Oracle Streams AQ memory usage:

- Deploying Oracle Streams AQ with XA
- Enqueuing Messages (Free Memory After Every Call) Using OCI
- Enqueuing Messages (Reuse Memory) Using OCI
- Dequeuing Messages (Free Memory After Every Call) Using OCI
- Dequeuing Messages (Reuse Memory) Using OCI

Example 2–40 Enqueuing Messages (Free Memory After Every Call) Using OCI

This program, `enqnoreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in the `scott` schema using `create_types.sql`. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c` (see the following). If there are no messages, then it waits for 60 seconds before timing out. In this program, the dequeue subroutine does not reuse client side objects' memory. It allocates the required memory before dequeue and frees it after the dequeue is complete.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

OCIEnv       *envhp;
OCIError     *errhp;
```

```
OCISvcCtx    *svchp;

struct message
{
  OCINumber    id;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text  *buf;
ub4   *buflen;
{
  OCIType         *mesgtdo = (OCIType *)0;   /* type descr of SCOTT.MESSAGE */
  message         *mesg    = (dvoid *)0;     /* instance of SCOTT.MESSAGE */
  null_message    *mesgind = (dvoid *)0;     /* null indicator */
  OCIAQDeqOptions *deqopt  = (OCIAQDeqOptions *)0;
  ub4              wait    = 60;             /* timeout after 60 seconds */
  ub4              navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */


  /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
          OCI_TYPEGET_ALL, &mesgtdo));

  /* Allocate an instance of SCOTT.MESSAGE, and get its null indicator: */
  checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
          mesgtdo, (dvoid *)0, OCI_DURATION_SESSION,
          TRUE, (dvoid **)&mesg));
  checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *)mesg,
           (dvoid **)&mesgind));

  /* Allocate a descriptor for dequeue options and set wait time, navigation: */
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
```

```
          OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
          (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
          (dvoid *)&navigation, 0,
          OCI_ATTR_NAVIGATION, errhp));

  /* Dequeue the message and commit: */
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
          deqopt, 0, mesgtdo, (dvoid **)&mesg,
          (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

  /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
          (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));

  /* Free the dequeue options descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

  /* Free the memory for the objects: */
  Checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *)mesg,
          OCI_OBJECTFREE_FORCE));
}                           /* end deqmesg */

void main()
{
  OCIServer     *srvhp;
  OCISession    *usrhp;
  dvoid         *tmp;
  text           buf[80];                /* payload text */
  ub4            buflen;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
               (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );
```

```
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

  /* Set attribute server context in the service context: */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
             (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);
}                          /* end main */

static void checkerr(errhp, status)
OCIError  *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4 errcode;

  if (status == OCI_SUCCESS) return;
```

```
  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                          /* end checkerr */
```

Example 2–41   Enqueuing Messages (Reuse Memory) Using OCI

This program, `enqreuse.c`, enqueues each line of text into a queue 'msgqueue' that has been created in the `scott` schema by executing `create_types.sql`. Each line of text entered by the user is stored in the queue until user enters `EOF`. In this program the enqueue subroutine reuses the memory for the message payload, as well as the Oracle Streams AQ message properties descriptor.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void enqmesg(ub4 msgno, text *buf);

struct message
{
  OCINumber     id;
  OCIString    *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
```

```
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

/* Global data reused on calls to enqueue: */
OCIEnv            *envhp;
OCIError          *errhp;
OCISvcCtx         *svchp;
message            msg;
null_message       nmsg;
OCIAQMsgProperties *msgprop;

static void enqmesg(msgno, buf)
ub4     msgno;
text   *buf;
{
  OCIType          *mesgtdo = (OCIType *)0; /* type descr of SCOTT.MESSAGE */
  message          *mesg = &msg;            /* instance of SCOTT.MESSAGE */
  null_message     *mesgind = &nmsg;        /* null indicator */
  text              corrid[128];            /* correlation identifier */

  /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

  /* Fill in the attributes of SCOTT.MESSAGE: */
  checkerr(errhp, OCINumberFromInt(errhp, &msgno, sizeof(ub4), 0, &mesg->id));
  checkerr(errhp, OCIStringAssignText(envhp, errhp, buf, strlen(buf),
           &mesg->data));
  mesgind->null_adt = mesgind->null_id = mesgind->null_data = 0;

  /* Set the correlation id in the message properties descriptor: */
  sprintf((char *)corrid, "Msg#: %d", msgno);
  checkerr(errhp, OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
           (dvoid *)&corrid, strlen(corrid),
           OCI_ATTR_CORRELATION, errhp));

  /* Enqueue the message and commit: */
  checkerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"msgqueue",
           0, msgprop, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));
```

```
  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));
}                           /* end enqmesg */

void main()
{
  OCIServer    *srvhp;
  OCISession   *usrhp;
  dvoid        *tmp;
  text          buf[80];                /* user supplied text */
  int           msgno = 0;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

  /* Set attribute server context in the service context: */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
            (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
            (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
            (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
```

```
          OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

  /* Allocate a message properties descriptor to fill in correlation ID :*/
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
          OCI_DTYPE_AQMSG_PROPERTIES,
          0, (dvoid **)0));
  do {
    printf("Enter a line of text (max 80 chars):");
    if (!gets((char *)buf))
      break;
    enqmesg((ub4)msgno++, buf);
  } while(1);

  /* Free the message properties descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)msgprop,
          OCI_DTYPE_AQMSG_PROPERTIES));

}                          /* end main */

static void checkerr(errhp, status)
OCIError   *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
```

```
  }
  exit(-1);
}                          /* end checkerr */
```

Example 2–42 Dequeuing Messages (Free Memory After Every Call) Using OCI

This program, `deqnoreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in the `scott` schema by executing `create_types.sql`. Messages are enqueued using `enqnoreuse` or `enqreuse`. If there are no messages, then it waits for 60 seconds before timing out. In this program the dequeue subroutine does not reuse client side objects' memory. It allocates the required memory before dequeue and frees it after the dequeue is complete.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

OCIEnv       *envhp;
OCIError     *errhp;
OCISvcCtx    *svchp;

struct message
{
  OCINumber     id;
  OCIString    *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text      *buf;
ub4       *buflen;
```

```
{
  OCIType          *mesgtdo = (OCIType *)0;  /* type descr of SCOTT.MESSAGE */
  message          *mesg = (dvoid *)0;       /* instance of SCOTT.MESSAGE */
  null_message     *mesgind   = (dvoid *)0;       /* null indicator */
  OCIAQDeqOptions  *deqopt    = (OCIAQDeqOptions *)0;
  ub4               wait      = 60;               /* timeout after 60 seconds */
  ub4              navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */

   /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

  /* Allocate an instance of SCOTT.MESSAGE, and get its null indicator: */
  checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
           mesgtdo, (dvoid *)0, OCI_DURATION_SESSION,
           TRUE, (dvoid **)&mesg));
  checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *)mesg,
           (dvoid **)&mesgind));

  /* Allocate a descriptor for dequeue options and set wait time, navigation: */
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
           OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp));

  /* Dequeue the message and commit: */
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
           deqopt, 0, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

  /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
           (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));
```

```
  /* Free the dequeue options descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

  /* Free the memory for the objects: */
  checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *)mesg,
           OCI_OBJECTFREE_FORCE));
}                                       /* end deqmesg */

void main()
{
  OCIServer    *srvhp;
  OCISession   *usrhp;
  dvoid        *tmp;
  text          buf[80];                /* payload text */
  ub4           buflen;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

  /* Set attribute server context in the service context: */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
               (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);
```

```
  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);
}                         /* end main */

static void checkerr(errhp, status)
OCIError   *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                          /* end checkerr */
```

Example 2–43 Dequeuing Messages (Reuse Memory) Using OCI

This program, `deqreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in the `scott` schema by executing `create_types.sql`. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c`. If there are no messages, then it waits for 60 seconds before timing out. In this program, the dequeue subroutine reuses client side objects' memory between invocation of `OCIAQDeq`.

- During the first call to `OCIAQDeq`, OCI automatically allocates the memory for the message payload.
- During subsequent calls to `OCIAQDeq`, the same payload pointers are passed and OCI automatically resizes the payload memory if necessary.
- #ifndef OCI_ORACLE

```
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

struct message
{
  OCINumber    id;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

/* Global data reused on calls to enqueue: */
OCIEnv          *envhp;
OCIError        *errhp;
OCISvcCtx       *svchp;
OCIAQDeqOptions *deqopt;
message         *mesg = (message *)0;
```

```
null_message    *mesgind = (null_message *)0;

static void deqmesg(buf, buflen)
text       *buf;
ub4        *buflen;
{

  OCIType        *mesgtdo    = (OCIType *)0; /* type descr of SCOTT.MESSAGE */
  ub4             wait       = 60;           /* timeout after 60 seconds */
  ub4             navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */

  /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

  /* Set wait time, navigation in dequeue options: */
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp));

  /*
   * Dequeue the message and commit. The memory for the payload is
   * automatically allocated/resized by OCI:
   */
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
           deqopt, 0, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

  /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
           (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));
}                           /* end deqmesg */

void main()
{
```

```
OCIServer    *srvhp;
OCISession   *usrhp;
dvoid        *tmp;
text          buf[80];                /* payload text */
ub4           buflen;

OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

/* set attribute server context in the service context */
OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* allocate a user context handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
               (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
           (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
         OCI_DEFAULT));

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
           (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* allocate the dequeue options descriptor */
```

```
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
           OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);

  /*
   * This program never reaches this point as the dequeue times out & exits.
   * If it does reach here, it is a good place to free the dequeue
   * options descriptor using OCIDescriptorFree and free the memory allocated
   * by OCI for the payload using OCIObjectFree
   */
}                         /* end main */

static void checkerr(errhp, status)
OCIError *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                          /* end checkerr */
```

Frequently Asked Questions

The following lists Oracle Streams AQ installation and general questions:

- Oracle Streams AQ Installation Questions
- General Oracle Streams AQ Questions

Oracle Streams AQ Installation Questions

How do I set up Internet access for Oracle Streams AQ? What components are required?

See Chapter 17, "Internet Access to Oracle Streams AQ" for a full discussion. The following summarizes the steps required to set up Internet access for Oracle Streams AQ queues:

1. Set up the Oracle Streams AQ **servlet**: If you are using a servlet execution engine that supports the Java Servlet 2.2 specification (such as Tomcat), then you must create a servlet that extends the `oracle.AQ.xml.AQxmlServlet` class. If you are using a servlet execution engine that supports the Java Servlet 2.0 specification (like Apache Jserv), then you must create a servlet that extends the `oracle.AQ.xml.AQxmlServlet20` class. Implement the `init()` method in the servlet to specify database connection parameters.
2. Set up user authentication: Configure the Web server to authenticate all the users that send `POST` requests to the Oracle Streams AQ servlet. Only authenticated users are allowed to access the Oracle Streams AQ servlet.
3. Set up user authorization: Register the Oracle Streams AQ agent name that is used to perform Oracle Streams AQ operations using `DBMS_AQADM.CREATE_AQ_AGENT`. Map the agent to the database users using `DBMS_AQADM.ENABLE_DB_ACCESS`.
4. Now clients can write **Simple Object Access Protocol** (SOAP) requests and send them to the Oracle Streams AQ servlet using HTTP `POST`.

How do I set up e-mail notifications?

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

1. Set the SMTP mail host: Invoke `DBMS_AQELM.SET_MAILHOST` as an Oracle Streams AQ administrator.
2. Set the SMTP mail port: Invoke `DBMS_AQELM.SET_MAILPORT` as an Oracle Streams AQ administrator. If not explicit, set defaults to 25.

3. Set the SendFrom address: Invoke `DBMS_AQELM.SET_SENDFROM`.
4. After setup, you can register for e-mail notifications using the **Oracle Call Interface** (OCI) or PL/SQL **API**.

How do I set up Oracle Streams AQ propagation over the Internet?

See Chapter 17, "Internet Access to Oracle Streams AQ" for a full discussion. In summary, follow the steps for setting up Internet access for Oracle Streams AQ. The destination databases must be set up for Internet access, as follows:

1. At the source database, create the database link with protocol as http, and host and port of the Web server running the Oracle Streams AQ servlet with the username password for authentication with the Web server/servlet runner. For example, if the Web server is running on computer `webdest.oracle.com` and listening for requests on port 8081, then the connect string of the database is:

   ```
   (DESCRIPTION=(ADDRESS=(PROTOCOL=http)(HOST=webdest.oracle.com)(PORT=8081))
   ```

   If SSL is used, then specify https as the protocol in the connect string. The database link is created as follows:

   ```
   create public database link propdb connect to john IDENTIFIED BY welcome
   using '(DESCRIPTION=(ADDRESS=(PROTOCOL=http) (HOST=webdest.oracle.com)
   (PORT=8081))';
   ```

   where user `John` with password `Welcome` is used to authenticate with the Web server, and is also known by the term Oracle Streams AQ HTTP agent.

   Note: You cannot use `net_service_name` in `tnsnames.ora` with the database link. Doing so results in error ORA-12538.

   ---

2. If SSL is used, then create an Oracle wallet and specify the wallet path at the source database:

   ```
   EXECUTE DBMS_AQADM.SET_AQ_PROPAGATIONWALLET('/home/myuid/cwallet.sso',
   'welcome');
   ```

3. Deploy the Oracle Streams AQ servlet at the destination database: Create a class `AQPropServlet` that extends `oracle.AQ.xml.AQxmlServlet20` (if you are using a Servlet 2.0 execution engine like Apache Jserv) or extends `oracle.AQ.xml.AQxmlServlet` (if you are using a Servlet 2.2 execution engine like Tomcat). This servlet must connect to the destination database. The servlet must be deployed on the Web server in the path `aqserv/servlet`.

> **Note:** In Oracle9*i*, the **propagation** servlet name and deployment path are fixed. That is, they must be AQPropServlet and the aqserv/servlet respectively.

4. At the destination database: Set up the authorization and authentication for the Internet user performing propagation, in this case, `John`.
5. Start propagation at the source site by calling:

```
DBMS_AQADM.SCHEDULE_PROPAGATION('src_queue', 'propdb').
```

General Oracle Streams AQ Questions

How are messages that have been dequeued but are still retained in the queue table accessed?

Access messages using SQL. Messages in the **queue table** (either because they are being retained or because they have not yet been processed). Each **queue** has a view that you can use (see "Number of Messages in Different States for the Whole Database View" on page 9-17).

Message retention means the messages are there, but how does the subscriber access these messages?

Typically we expect the **subscriber** to access the messages using the **dequeue** interface. If, however, you would like to see processed or waiting messages, then you can either dequeue by message ID or use SQL.

Can the sort order be changed after the queue table is created?

You cannot change the sort order for messages after you have created the queue table.

How do I dequeue from an exception queue?

The **exception queue** for a multiconsumer queue must also be a multiconsumer queue.

Expired messages in multiconsumer queues cannot be dequeued by the intended recipients of the **message**. However, they can be dequeued in the `REMOVE` mode once (and only once) using a `NULL` **consumer** name in dequeue options. Messages can also be dequeued from an exception queue by specifying the message ID.

Expired messages can be dequeued only by specifying message ID if the multiconsumer exception queue was created in a queue table without the compatible parameter or with the compatible parameter set to '8.0'

What does the latency parameter mean in scheduling propagation?

If a latency less than 0 was specified in the propagation schedule, then the job is rescheduled to run after the specified latency. The time at which the job actually 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

How can I control the tablespaces in which the queue tables are created?

You can pick a tablespace for storing the queue table and all its ancillary objects using the `storage_clause` parameter in `DBMS_AQADM.CREATE_QUEUE_TABLE`. However, once you pick the tablespace, any **index-organized table** (IOT) or index created for that queue table goes to the specified tablespace. Currently, you do not have a choice to split them between different tablespaces.

How do you associate Real Application Clusters instance affinities with queue tables?

In 8.1 you can associate RAC instance affinities with queue tables. If you are using `q1` and `q2` in different instances, then you can use `alter_queue_table` (or even create queue table) on the queue table and set the `primary_instance` to the appropriate `instance_id`.

Can you give me some examples of a subscriber rule containing - message properties - message data properties?

Yes, here is a simple rule that specifies message properties: rule = 'priority = 1'; here are example **rules** that specify a combination of message properties and data attributes: rule = `'priority = 1 AND tab.userdata.sal = 1000'` `rule = '((priority between 0 AND 3)` OR `correlation = "BACK_ORDERS")` AND `tab.userdata.customer_name` like "JOHN DOE")'

User data properties or attributes apply only to object payloads and must be prefixed with tab.userdata in all cases. Check documentation for more examples.

Is registration for notification (OCI) the same as starting a listener?

No. Registration is an OCI client call to be used for **asynchronous** notifications (that is, push). It provides a notification from the server to the client when a message is available for dequeue. A client side function (callback) is invoked by the server when the message is available. Registration for notification is both nonblocking and nonpolling.

What is the use of nonpersistent queues?

To provide a mechanism for notification to all users that are currently connected. The nonpersistent queue mechanism supports the **enqueue** of a message to a nonpersistent queue and OCI notifications are used to deliver such messages to users that are currently registered for notification.

Is there a limit on the length of a recipient list? Or on the number of subscribers for a particular queue?

Yes, 1024 subscribers or recipients for any queue.

How can I clean out a queue with UNDELIVERABLE messages?

You can dequeue these messages by `msgid.` You can find the `msgid` by querying the queue table view. Eventually the messages are moved to the exception queue (you must have the Oracle Streams AQ Queue Monitor Process running for this to happen). You can dequeue these messages from the exception queue with a usual dequeue.

Is it possible to update the message payload after it has been enqueued?

Only by dequeuing and enqueuing the message again. If you are changing the message payload, then it is a different message.

Can asynchronous notification be used to invoke an executable every time there is a new message?

Notification is possible only to OCI clients. The client need not be connected to the database to receive notifications. The client specifies a callback function which is executed 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.

Does propagation work from multiconsumer queues to single-consumer queues and vice versa?

Propagation from a multiconsumer queue to a single consumer queue is possible. The reverse is not possible (propagation is not possible from a single consumer queue).

Why do I sometimes get ORA-1555 error on dequeue?

You are probably using the `NEXT_MESSAGE` navigation option for dequeue. This 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. This might not perform as well, so we suggest you dequeue them in batches: `FIRST_MESSAGE` for one, and `NEXT_MESSAGE` for the next, say, 1000 messages, and then `FIRST_MESSAGE` again, and so on.

After a message has been moved to an exception queue, is there a way, using SQL or otherwise, of identifying which queue the message resided in before moving to the exception queue?

No, Oracle Streams AQ does not provide this information. To get around this, the application could save this information in the message.

What is the order in which messages are dequeued if many messages are enqueued in the same second?

When the `enq_time` is the same for messages, there is another field called step_no that is monotonically increasing (for each message that has the same `enq_time`). Hence this helps in maintaining the order of the messages. There is no situation when both `enq_time` and `step_no` are the same for more than one message enqueued from the same session.

What happened to OMB? When should we use Oracle Streams AQ and when should we use Oracle MessageBroker?

In Oracle9*i* and higher, OMB functionality is provided in Oracle Database. If you are using Oracle9*i* or higher database, then use the functionality offered by the database. You do not need OMB. Note also that from Oracle9i release 2 (9.2) Oracle Messaging Gateway (MGW) provides the OMB functionality.

With Oracle8*i*, use MGW in the following scenarios:

- To integrate with Websphere MQ
- To use HTTP framework

Use **Java Message Service** (JMS) functionality directly from the database in other scenarios.

Can I use Oracle Streams AQ with Virtual Private Database?

Yes, you can specify a security policy with Oracle Streams AQ queue tables. While dequeuing, use the dequeue condition (`deq_cond`) or the correlation ID 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.

How do I clean up my retained messages?

The Oracle Streams AQ retention feature can be used to automatically clean up messages after the user-specified duration after consumption.

I have an application in which I inserted the messages for the wrong subscriber. How do I clean up those messages?

You can do a dequeue with the subscriber name or by message ID. This consumes the messages, which are cleaned up after their retention time expires.

I'm running propagation between multiple installations of Oracle Database. For some reason, one of the destination databases has gone down for an extended duration. How do I clean up messages for that destination?

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.

Transformation Questions

How do you do transformation of XML data?

Transformation of XML data can be accomplished in one of the following ways:

- Using the `extract()` method supported on `XMLType` to return an object of `XMLType` after applying the supplied `XPath` expression

- Creating a PL/SQL function that transforms the `XMLType` object by applying an XSLT transformation to it, using the package `XSLPROCESSOR`

3 Basic Components

This chapter describes the Oracle Streams Advanced Queuing (AQ) basic components.

This chapter contains the following topics:

- Object Name (object_name)
- Type Name (type_name)
- AQ Agent Type (aq$_agent)
- AQ Recipient List Type (aq$_recipient_list_t)
- AQ Agent List Type (aq$_agent_list_t)
- AQ Subscriber List Type (aq$_subscriber_list_t)
- AQ Registration Information List Type (aq$_reg_info_list)
- AQ Post Information List Type (aq$_post_info_list)
- AQ Registration Information Type (aq$_reg_info)
- AQ Notification Descriptor Type
- AQ Post Information Type
- Enumerated Constants in the Oracle Streams AQ Administrative Interface
- Enumerated Constants in the Oracle Streams AQ Operational Interface
- INIT.ORA Parameter File Considerations

See Also:

- Chapter 8, "Oracle Streams AQ Administrative Interface"
- Chapter 10, "Oracle Streams AQ Operational Interface: Basic Operations"

Object Name (object_name)

Purpose

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

Syntax

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

Usage

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 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 (type_name)

Purpose

Defines queue types.

Syntax

```
type_name := VARCHAR2
type_name := object_type | "RAW"
```

Usage

For details on creating object types refer to *Oracle Database Concepts*. The maximum number of attributes in the object type is limited to 900.

To store payload of type RAW, Oracle Streams AQ 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 Streams AQ 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 (aq$_agent)

Purpose

Identifies a **producer** or a **consumer** of a **message**.

Syntax

```
TYPE aq$_agent IS OBJECT (
   name              VARCHAR2(30),
   address           VARCHAR2(1024),
   protocol          NUMBER)
```

Usage

All consumers that are added as subscribers to a multiconsumer queue must have unique values for the AQ$_AGENT parameters. You can add more subscribers by repeatedly using the DBMS_AQADM.ADD_SUBSCRIBER procedure up to a maximum of 1024 subscribers for a multiconsumer queue. Two subscribers cannot have the same values for the NAME, ADDRESS, and PROTOCOL attributes for the AQ$_AGENT type. At least one of the three attributes must be different for two subscribers.

Parameters

name (VARCHAR2(30))

Name of a producer or consumer of a message. The name of the agent 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 Reference*.

address (VARCHAR2(1024))
A character field of up to 1024 bytes that is interpreted in the context of the protocol. If the protocol is 0 (default), then the address is of the form `[schema.]queue[@dblink]`.

protocol (NUMBER)
Protocol to interpret the address and propagate the message. The default value is 0.

AQ Recipient List Type (aq$_recipient_list_t)

Purpose
Identifies the list of agents that receive the message.

Syntax
```
TYPE aq$_recipient_list_t IS TABLE OF aq$_agent
           INDEX BY BINARY_INTEGER;
```

AQ Agent List Type (aq$_agent_list_t)

Purpose
Identifies the list of agents for `DBMS_AQ.LISTEN` to listen for.

Syntax
```
TYPE aq$_agent_list_t IS TABLE OF aq$_agent
           INDEX BY BINARY INTEGER;
```

AQ Subscriber List Type (aq$_subscriber_list_t)

Purpose
Identifies the list of subscribers that subscribe to this queue.

Syntax
```
TYPE aq$_subscriber_list_t IS TABLE OF aq$_agent
          INDEX BY BINARY INTEGER;
```

AQ Registration Information List Type (aq$_reg_info_list)

Purpose

Identifies the list of registrations to a queue.

Syntax

```
TYPE aq$_reg_info_list AS VARRAY(1024) OF sys.aq$_reg_info
```

AQ Post Information List Type (aq$_post_info_list)

Purpose

Identifies the list of anonymous subscriptions to which messages are posted.

Syntax

```
TYPE aq$_post_info_list AS VARRAY(1024) OF sys.aq$_post_info
```

AQ Registration Information Type (aq$_reg_info)

Purpose

The `aq$_reg_info` data structure identifies a producer or a consumer of a message.

Syntax

```
TYPE sys.aq$_reg_info IS OBJECT (
   name      VARCHAR2(128),
   namespace NUMBER,
   callback  VARCHAR2(4000),
   context   RAW(2000));
```

Attributes

name

Specifies the name of the subscription. The subscription name is of the form *`schema.queue`* if the registration is for a single consumer queue and *`schema.queue:consumer_name`* if the registration is for a multiconsumer queue.

namespace

Specifies the namespace of the subscription. To receive notifications from Oracle Streams AQ queues, the namespace must be `DBMS_AQ.NAMESPACE_AQ`. To receive notifications from other applications using `DBMS_AQ.POST` or `OCISubscriptionPost()`, the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`.

callback

Specifies the action to be performed on message notification. For e-mail notifications, the form is `mailto://xyz@company.com`. For Oracle Streams AQ PL/SQL Callback, use `plsql://`*`schema.procedure`*?PR=0 for raw message payload or `plsql://`*`schema.procedure`*`?PR=1` for Oracle object type message payload converted to XML.

context

Specifies the context that is to be passed to the callback function. The default is `NULL`.

Table 3–1 shows the actions performed for **nonpersistent** queues for different notification mechanisms when RAW presentation is specified. Table 3–2 shows the actions performed when XML presentation is specified.

Table 3–1 Actions Performed for Nonpersistent Queues When RAW Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
RAW	OCI callback receives the RAW data in the payload.	Not supported	PL/SQL callback receives the RAW data in the payload.
Oracle object type	Not supported	Not supported	Not supported

Table 3–2 Actions Performed for Nonpersistent Queues When XML Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
RAW	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.
Oracle object type	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.

AQ Notification Descriptor Type

Purpose

The `aq$_descriptor` data structure specifies the Oracle Streams AQ Descriptor received by the Oracle Streams AQ PL/SQL callbacks upon notification.

Syntax

```
TYPE sys.aq$_descriptor IS OBJECT (
  queue_name    VARCHAR2(30),
  consumer_name VARCHAR2(30),
  msg_id        RAW(16),
  msg_prop      msg_prop_t);
```

Attributes

queue_name
Name of the queue in which the message was enqueued which resulted in the notification.

consumer_name
Name of the consumer for the multiconsumer queue.

msg_id
ID of the message.

msg_prop
Message properties.

AQ Post Information Type

Purpose

The `aq$_post_info` data structure specifies anonymous subscriptions to which you want to post messages.

Syntax

```
TYPE sys.aq$_post_info IS OBJECT (
  name       VARCHAR2(128),
  namespace  NUMBER,
  payload    RAW(2000));
```

Attributes

name
Name of the anonymous subscription to which you want to post.

namespace
To receive notifications from other applications using DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.

payload
The payload to be posted to the anonymous subscription. The default is NULL.

Enumerated Constants in the Oracle Streams AQ 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:

```
DBMS_AQADM.NORMAL_QUEUE
```

Table 3–3 lists the enumerated constants.

Table 3–3 Enumerated Constants in the Oracle Streams AQ Administrative Interface

Parameter	Options
retention	0,1,2...INFINITE
message_grouping	TRANSACTIONAL, NONE
queue_type	NORMAL_QUEUE, EXCEPTION_QUEUE,NON_PERSISTENT_QUEUE

Enumerated Constants in the Oracle Streams AQ 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 3–4 lists the enumerated constants.

Table 3–4 Enumerated Constants in the Oracle Streams AQ Operational Interface

Parameter	Options
visibility	IMMEDIATE, ON_COMMIT
dequeue mode	BROWSE, LOCKED, REMOVE, REMOVE_NODATA
navigation	FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION
state	WAITING, READY, PROCESSED, EXPIRED
sequence_deviation	BEFORE, TOP
wait	FOREVER, NO_WAIT
delay	NO_DELAY
expiration	NEVER
namespace	NAMESPACE_AQ, NAMESPACE_ANONYMOUS

INIT.ORA Parameter File Considerations

You can specify the AQ_TM_PROCESSES and JOB_QUEUE_PROCESSES parameters in the init.ora parameter file.

AQ_TM_PROCESSES Parameter No Longer Needed in init.ora

Prior to Oracle Database 10*g*, Oracle Streams AQ time manager processes were controlled by the init.ora parameter AQ_TM_PROCESSES, which had to be set to nonzero to perform time monitoring on queue messages and for processing messages with delay and expiration properties specified. These processes were named QMNO-9 and could be changed using statement:

```
ALTER SYSTEM SET AQ_TM_PROCESSES=X
```

Parameter X ranged from 0 to 10. When X was set to 1 or more, that number of QMN processes were then started. If the parameter was not specified, or was set to 0, then queue monitor processes were not started.

In Oracle Streams AQ release 10.1, this has been changed to a coordinator-slave architecture, where a coordinator is automatically spawned if Oracle Streams AQ or Streams is being used in the system. This process, named **QMNC**, dynamically spawns slaves depending on the system load. The slaves, named qXXX, do various background tasks for Oracle Streams AQ or Streams. Because the number of processes is determined automatically and tuned constantly, you no longer need set AQ_TM_PROCESSES.

Even though it is no longer necessary to set AQ_TM_PROCESSES when Oracle Streams AQ or Streams is used, if you do specify a value, then that value is taken into account. However, the number of qXXX 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.

Table 3–5 lists AQ_TM_PROCESSES parameter information.

Table 3–5 AQ_TM_PROCESSES Parameters

Parameter	Options
Parameter Name	aq_tm_processes
Parameter Type	integer
Parameter Class	Dynamic
Allowable Values	0 to 10
Syntax	aq_tm_processes = *allowable_value*
Name of process	ora_qmnc_*ORACLE_SID* ora_q00*n*_*ORACLE_SID*
Example	aq_tm_processes = 1

JOB_QUEUE_PROCESSES Parameter

Propagation is handled by job queue (Jnnn) processes. The number of job queue processes started in an instance is controlled by the init.ora parameter JOB_QUEUE_PROCESSES. The default value of this parameter is 0. For message **propagation** to take place, this parameter must be set to at least 2. The database administrator can set it to higher values if there are many queues from which the messages must be propagated, or if there are many destinations to which the messages must be propagated, or if there are other jobs in the job queue.

See Also: *Oracle Database SQL Reference* for more information on `JOB_QUEUE_PROCESSES`

4

Oracle Streams AQ: Programmatic Environments

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` has been deprecated in release 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 Environments for Accessing Oracle Streams AQ
- Using PL/SQL to Access Oracle Streams AQ
- Using OCI to Access Oracle Streams AQ
- Using OCCI to Access Oracle Streams AQ
- Using Visual Basic (OO4O) to Access Oracle Streams AQ
- Using Oracle Java Message Service (OJMS) to Access Oracle Streams AQ
- Using Oracle Streams AQ XML Servlet to Access Oracle Streams AQ
- Comparing Oracle Streams AQ Programmatic Environments

Programmatic Environments for Accessing Oracle Streams AQ

Table 4–1 lists Oracle Streams AQ programmatic environments, functions supported in each environment, and syntax references.

Table 4–1 Oracle Streams AQ Programmatic Environments

Language	Precompiler or Interface Program	Functions Supported	Syntax References
PL/SQL	`DBMS_AQADM` and `DBMS_AQ` Packages	Administrative and operational	*PL/SQL Packages and Types Reference*
C	**Oracle Call Interface** (OCI)	Operational only	*Oracle Call Interface Programmer's Guide*
Visual Basic	**Oracle Objects for OLE** (OO4O)	Operational only	Online help available from Application Development submenu of Oracle installation.
Java (JMS)	`oracle.JMS` package using JDBC **API**	Administrative and operational	*Oracle Streams Advanced Queuing Java API Reference*
AQ XML **servlet**	`oracle.AQ.xml.AQxmlServlet` using HTTP	Operational only	*Oracle XML API Reference*

Using PL/SQL to Access Oracle Streams AQ

The PL/SQL packages `DBMS_AQADM` and `DBMS_AQ` support access to Oracle Streams AQ administrative and operational functions using the native Oracle Streams AQ 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 Streams AQ Internet agent

- Grant or revoke privileges of database users to Oracle Streams AQ 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 Streams AQ objects in a LDAP server

> **See Also:** *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 4–2 through Table 4–9.

Using OCI to Access Oracle Streams AQ

OCI provides an interface to Oracle Streams AQ functions using the native Oracle Streams AQ 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 Streams AQ descriptors and the message payload.

> **See Also:**
>
> - "Enqueuing and Dequeuing Oracle Streams AQ Messages" on page 2-9 for OCI interface examples
> - "Oracle Streams AQ and Memory Usage" on page 2-65 for examples illustrating management of OCI descriptors

Using OCCI to Access Oracle Streams AQ

C++ applications can use OCCI, which has a set of Oracle Streams AQ interfaces that enable messaging clients to access Oracle Streams AQ. 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 Streams AQ

Visual Basic (OO4O) supports access to Oracle Streams AQ operational functions using the native Oracle Streams AQ 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

Using Oracle Java Message Service (OJMS) to Access Oracle Streams AQ

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 Streams AQ 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 `AdtMessages`

  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 Streams AQ for that subscriber.

See Also:

- *Java Message Service Specification*, version 1.1, March 18, 2002, Sun Microsystems, Inc.
- `http://otn.oracle.com/docs/products/aq/doc_library/ojms/index.html` for more information on Oracle JMS
- Part V, "Using Oracle JMS and Oracle Streams AQ"
- *Oracle Streams Advanced Queuing Java API Reference*

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 Streams AQ 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/jdbc/lib/orai18n.jar
$ORACLE_HOME/jdk/jre/lib/ext/jta.jar
$ORACLE_HOME/jdk/jre/lib/ext/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/jdbc/lib/orai18n.jar
$ORACLE_HOME/jdk/jre/lib/ext/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 Streams AQ XML Servlet to Access Oracle Streams AQ

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

Using the Oracle Streams AQ 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

The servlet can be created by defining a Java class that extends the `oracle.AQ.xml.AQxmlServlet` or `oracle.AQ.xml.AQxmlServlet20` class. These classes in turn extend the `javax.servlet.http.HttpServlet` class.

The servlet can be deployed on any Web server or ServletRunner that implements the Javasoft Servlet 2.0 or Servlet 2.2 interfaces. With Javasoft Servlet 2.0, you must define a class that extends `oracle.AQ.xml.AQxmlServlet20`. With Javasoft Servlet 2.2, you must define a class that extends `oracle.AQ.xml.AQxmlServlet`.

The servlet can be compiled using JDK 1.2.x, JDK 1.3.x, or JDK 1.4.x libraries.

For JDK 1.4.x the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/ojdbc14.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$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
```

For JDK 1.3.x the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$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
```

For JDK 1.2.x the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$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
```

Because the servlet uses JDBC OCI drivers to connect to the Oracle Database server, the Oracle Database client libraries must be installed on the computer that hosts the servlet. The `LD_LIBRARY_PATH` must contain `$ORACLE_HOME/lib`.

> **See Also:** Chapter 17, "Internet Access to Oracle Streams AQ" for more information on Internet access to Oracle Streams AQ

Comparing Oracle Streams AQ Programmatic Environments

Available functions for the Oracle Streams AQ programmatic environments are listed by use case in Table 4–2 through Table 4–9. Use cases are described in Chapter 8 through Chapter 10 and Chapter 12 through Chapter 15.

Oracle Streams AQ Administrative Interfaces

Table 4–2 lists the equivalent Oracle Streams AQ administrative functions for the PL/SQL and Java (JMS) programmatic environments.

Table 4–2 Comparison of Oracle Streams AQ Programmatic Environments: Administrative Interface

Use Case	PL/SQL	Java (JMS)
Create a connection factory	N/A	`AQjmsFactory.getQueueConnectionFactory` `AQjmsFactory.getTopicConnectionFactory`
Register a **connection factory** in an LDAP server	N/A	`AQjmsFactory.registerConnectionFactory`
Create a queue table	`DBMS_AQADM.CREATE_QUEUE_TABLE`	`AQjmsSession.createQueueTable`
Get a queue table	Use *`schema.queue_table_name`*	`AQjmsSession.getQueueTable`
Alter a queue table	`DBMS_AQADM.ALTER_QUEUE_TABLE`	`AQQueueTable.alter`

Table 4–2 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Administrative

Use Case	PL/SQL	Java (JMS)
Drop a queue table	`DBMS_AQADM.DROP_QUEUE_TABLE`	`AQQueueTable.drop`
Create a queue	`DBMS_AQADM.CREATE_QUEUE`	`AQjmsSession.createQueue`
Get a queue	Use *`schema.queue_name`*	`AQjmsSession.getQueue`
Create a nonpersistent queue	`DBMS_AQADM.CREATE_NP_QUEUE`	Not supported
Create a multiconsumer queue/topic in a queue table with multiple consumers enabled	`DBMS_AQADM.CREATE_QUEUE`	`AQjmsSession.createTopic`
Get a multiconsumer queue/topic	Use *`schema.queue_name`*	`AQjmsSession.getTopic`
Alter a queue/topic	`DBMS_AQADM.ALTER_QUEUE`	`AQjmsDestination.alter`
Start a queue/topic	`DBMS_AQADM.START_QUEUE`	`AQjmsDestination.start`
Stop a queue/topic	`DBMS_AQADM.STOP_QUEUE`	`AQjmsDestination.stop`
Drop a queue/topic	`DBMS_AQADM.DROP_QUEUE`	`AQjmsDestination.drop`
Grant system privileges	`DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE`	`AQjmsSession.grantSystemPrivilege`
Revoke system privileges	`DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE`	`AQjmsSession.revokeSystemPrivilege`
Grant a queue/topic privilege	`DBMS_AQADM.GRANT_QUEUE_PRIVILEGE`	`AQjmsDestination.grantQueuePrivilege` `AQjmsDestination.grantTopicPrivilege`
Revoke a queue/topic privilege	`DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE`	`AQjmsDestination.revokeQueuePrivilege` `AQjmsDestination.revokeTopicPrivilege`
Verify a queue type	`DBMS_AQADM.VERIFY_QUEUE_TYPES`	Not supported
Add a subscriber	`DBMS_AQADM.ADD_SUBSCRIBER`	See Table 4–6
Alter a subscriber	`DBMS_AQADM.ALTER_SUBSCRIBER`	See Table 4–6
Remove a subscriber	`DBMS_AQADM.REMOVE_SUBSCRIBER`	See Table 4–6

Table 4–2 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Administrative

Use Case	PL/SQL	Java (JMS)
Schedule propagation	DBMS_AQADM.SCHEDULE_ PROPAGATION	AQjmsDestination.scheduleP ropagation
Enable a propagation schedule	DBMS_AQADM.ENABLE_ PROPAGATION_SCHEDULE	AQjmsDestination.enablePro pagationSchedule
Alter a propagation schedule	DBMS_AQADM.ALTER_ PROPAGATION_SCHEDULE	AQjmsDestination.alterProp agationSchedule
Disable a propagation schedule	DBMS_AQADM.DISABLE_ PROPAGATION_SCHEDULE	AQjmsDestination.disablePr opagationSchedule
Unschedule a propagation	DBMS_AQADM.UNSCHEDULE_ PROPAGATION	AQjmsDestination.unschedul ePropagation
Create an Oracle Streams AQ Internet Agent	DBMS_AQADM.CREATE_AQ_AGENT	Not supported
Alter an Oracle Streams AQ Internet Agent	DBMS_AQADM.ALTER_AQ_AGENT	Not supported
Drop an Oracle Streams AQ Internet Agent	DBMS_AQADM.DROP_AQ_AGENT	Not supported
Grant database user privileges to an Oracle Streams AQ Internet Agent	DBMS_AQADM.ENABLE_AQ_AGENT	Not supported
Revoke database user privileges from an Oracle Streams AQ Internet Agent	DBMS_AQADM.DISABLE_AQ_ AGENT	Not supported
Add alias for queue, agent, ConnectionFactory in a LDAP server	DBMS_AQADM.ADD_ALIAS_TO_ LDAP	Not supported
Delete alias for queue, agent, ConnectionFactory in a LDAP server	DBMS_AQADM.DEL_ALIAS_FROM_ LDAP	Not supported

Oracle Streams AQ Operational Interfaces

Table 4–3 through Table 4–9 list equivalent Oracle Streams AQ operational functions for the programmatic environments PL/SQL, OCI, Oracle Streams AQ XML Servlet, and JMS, for various use cases.

Table 4–3 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Create Connection, Session, Message Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Create a connection	N/A	`OCIServerAttach`	Open an HTTP connection after authenticating with the Web server	`AQjmsQueueConnectionFactory.createQueueConnection` `AQjmsTopicConnectionFactory.createTopicConnection`
Create a session	N/A	`OCISessionBegin`	An HTTP servlet session is automatically started with the first SOAP request	`QueueConnection.createQueueSession` `TopicConnection.createTopicSession`

Table 4–3 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Create Connection, Session, Message Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Create a RAW message	Use SQL RAW type for message	Use OCIRaw for Message	Supply the hex representation of the message payload in the XML message. For example, `<raw>023f4523</raw>`	Not supported

Create a message with structured data	Use SQL Oracle **object type** for message	Use SQL Oracle object type for message	For Oracle object type queues that are not JMS queues (that is, they are not type `AQ$_JMS_*`), the XML specified in `<message payload>` must map to the SQL type of the payload for the queue table.
			
			For JMS queues, the XML specified in the `<message_payload>` must be one of the following: `<jms_text_message>`, `<jms_map_message>`, `<jms_bytes_message>`, `<jms_object_message>`
				`Session.createTextMessage`
				
				`Session.createObjectMessage`
				
				`Session.createMapMessage`
				
				`Session.createBytesMessage`
				
				`Session.createStreamMessage`
				
				`AQjmsSession.createAdtMessage`

Create a message **producer**	N/A	N/A	N/A	`QueueSession.createSender`
				
				`TopicSession.createPublisher`

Table 4–4 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Enqueue a message to a single-consumer queue	`DBMS_AQ.enqueue`	`OCIAQEnq`	`<AQXmlSend>`	`QueueSender.send`
Enqueue a message to a queue and specify visibility options	`DBMS_AQ.enqueue` Specify visibility in `ENQUEUE_OPTIONS`	`OCIAQEnq` Specify `OCI_ATTR_VISIBILITY` in `OCIAQEnqOptions`	`<AQXmlSend>` Specify `<visibility>` in `<producer_options>`	Not supported
Enqueue a message to a single-consumer queue and specify message properties priority and expiration	`DBMS_AQ.enqueue` Specify priority, expiration in `MESSAGE_PROPERTIES`	`OCIAQEnq` Specify `OCI_ATTR_PRIORITY`, `OCI_ATTR_EXPIRATION` in `OCIAQMsgProperties`	`<AQXmlSend>` Specify `<priority>`, `<expiration>` in `<message_header>`	Specify `priority` and `TimeToLive` during `QueueSender.send` or `.setTimeToLive` and `MessageProducer.setPriority` followed by `QueueSender.send`

Table 4–4 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Enqueue a message to a single-consumer queue and specify message properties correlationID, delay, and **exception queue**	DBMS_AQ.enqueue Specify correlation, delay, exception_ queue in MESSAGE_PROPERTIES	OCIAQEnq Specify OCI_ATTR_CORRELATION, OCI_ATTR_DELAY, OCI_ATTR_EXCEPTION_QUEUE in OCIAQMsgProperties	<AQXmlSend> Specify <correlation_id>, <delay>, <exception_queue> in <message_header>	Message.setJMSCorrelationID Delay and exception queue specified as provider specific message properties JMS_OracleDelay JMS_OracleExcpQ followed by QueueSender.send
Enqueue a message to a single-consumer queue and specify user-defined message properties	Not supported Properties should be part of payload	Not supported Properties should be part of payload	<AQXmlSend> Specify <name> and <int_value>, <string_value>, <long_value>, and so on in <user_properties>	Message.setIntProperty Message.setStringProperty Message.setBooleanProperty and so forth, followed by QueueSender.send
Enqueue a message to a single-consumer queue and specify message **transformation**	DBMS_AQ.enqueue Specify transformation in ENQUEUE_OPTIONS	OCIAQEnq Specify OCI_ATTR_TRANSFORMATION in OCIAQEnqOptions	<AQXmlSend> Specify <transformation> in <producer_options>	AQjmsQueueSender.setTransformation followed by QueueSender.send

Table 4–5 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Publish a message to a multiconsumer queue/topic using default subscription list	`DBMS_AQ.enqueue` Set `recipient_list` to `NULL` in `MESSAGE_PROPERTIES`	`OCIAQEnq` Set `OCI_ATTR_RECIPIENT_LIST` to `NULL` in `OCIAQMsgProperties`	`<AQXmlPublish>`	`TopicPublisher.publish`
Publish a message to a multiconsumer queue/topic using specific recipient list See footnote-1	`DBMS_AQ.enqueue` Specify recipient list in `MESSAGE_PROPERTIES`	`OCIAQEnq` Specify `OCI_ATTR_RECIPIENT_LIST` in `OCIAQMsgProperties`	`<AQXmlPublish>` Specify `<recipient_list>` in `<message_header>`	`AQjmsTopicPublisher.publish` Specify recipients as an array of `AQjmsAgent`
Publish a message to a multiconsumer queue/topic and specify message properties priority and expiration	`DBMS_AQ.enqueue` Specify priority, expiration in `MESSAGE_PROPERTIES`	`OCIAQEnq` Specify `OCI_ATTR_PRIORITY`, `OCI_ATTR_EXPIRATION` in `OCIAQMsgProperties`	`<AQXmlPublish>` Specify `<priority>`, `<expiration>` in the `<message_header>`	Specify `priority` and `TimeToLive` during `TopicPublisher.publish` or `MessageProducer.setTimeToLive` and `MessageProducer.setPriority` followed by `TopicPublisher.publish`

Table 4–5 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Publish a message to a multiconsumer queue/topic and specify send options correlationID, delay, and exception queue	`DBMS_AQ.enqueue`
	Specify correlation, delay, `exception_queue` in `MESSAGE_PROPERTIES`
		`OCIAQEnq`
		Specify `OCI_ATTR_CORRELATION`, `OCI_ATTR_DELAY`, `OCI_ATTR_EXCEPTION_QUEUE` in `OCIAQMsgProperties`
			<AQXmlPublish> Specify <correlation_id>, <delay>, <exception_queue> in <message_header>	`Message.setJMSCorrelationID`
				Delay and exception queue specified as provider-specific message properties
				`JMS_OracleDelay`
				`JMS_OracleExcpQ`
				followed by
				`TopicPublisher.publish`

Publish a message to a topic and specify user-defined message properties	Not supported
	Properties should be part of payload
		Not supported
		Properties should be part of payload
			<AQXmlPublish> Specify <name> and <int_value>, <string_value>, <long_value>, and so on in <user_properties>	`Message.setIntProperty`
				`Message.setStringProperty`
				`Message.setBooleanProperty`
				and so forth, followed by
				`TopicPublisher.publish`

Publish a message to a topic and specify message transformation	`DBMS_AQ.enqueue`
	Specify transformation in `ENQUEUE_OPTIONS`
		`OCIAQEnq`
		Specify `OCI_ATTR_TRANSFORMATION` in `OCIAQEnqOptions`
			<AQXmlPublish> Specify <transformation> in <producer_options>	`AQjmsTopicPublisher.setTransformation`
				followed by
				`TopicPublisher.publish`

Table 4–6 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Subscribing for Messages in a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Add a subscriber	See administrative interfaces	Not supported	Not supported	`TopicSession.createDurableSubscriber` `AQjmsSession.createDurableSubscriber`
Alter a subscriber	See administrative interfaces	Not supported	Not supported	`TopicSession.createDurableSubscriber` `AQjmsSession.createDurableSubscriber` using the new selector
Remove a subscriber	See administrative interfaces	Not supported	Not supported	`AQjmsSession.unsubscribe`

Table 4–7 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Browse Messages in a Queue Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Browse messages in a queue/topic	`DBMS_AQ.dequeue`
	Set `dequeue_mode` to `BROWSE` in `DEQUEUE_OPTIONS`
		`OCIAQDeq`
		Set `OCI_ATTR_DEQ_MODE` to `BROWSE` in `OCIAQDeqOptions`
			<AQXmlReceive> Specify <dequeue_mode> BROWSE in <consumer_options>	`QueueSession.createBrowser`
				`QueueBrowser.getEnumeration`
				Not supported on topics
				`oracle.jms.AQjmsSession.createBrowser`
				`oracle.jms.TopicBrowser.getEnumeration`

Browse messages in a queue/topic and lock messages while browsing	`DBMS_AQ.dequeue`
	Set `dequeue_mode` to `LOCKED` in `DEQUEUE_OPTIONS`
		`OCIAQDeq`
		Set `OCI_ATTR_DEQ_MODE` to `LOCKED` in `OCIAQDeqOptions`
			<AQXmlReceive> Specify <dequeue_mode> LOCKED in <consumer_options>	`AQjmsSession.createBrowser` set `locked` to `TRUE`.
				`QueueBrowser.getEnumeration`
				Not supported on topics
				`oracle.jms.AQjmsSession.createBrowser`
				`oracle.jms.TopicBrowser.getEnumeration`

Table 4–8 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Receive Messages from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Start a connection for receiving messages	N/A	N/A	N/A	`Connection.start`

Create a message consumer	N/A	N/A	N/A	`QueueSession.createQueueReceiver`
				`TopicSession.createDurableSubscriber`
				`AQjmsSession.createTopicReceiver`

Dequeue a message from a queue/topic and specify visibility	`DBMS_AQ.dequeue` Specify visibility in `DEQUEUE_OPTIONS`	`OCIAQDeq` Specify `OCI_ATTR_VISIBILITY` in `OCIAQDeqOptions`	`<AQXmlReceive>` Specify `<visibility>` in `<consumer_options>`	Not supported

Dequeue a message from a queue/topic and specify transformation	`DBMS_AQ.dequeue`
	Specify transformation in `DEQUEUE_OPTIONS`
		`OCIAQDeq`
		Specify `OCI_ATTR_TRANSFORMATION` in `OCIAQDeqOptions`
			`<AQXmlReceive>` Specify `<transformation>` in `<consumer_options>`	`AQjmsQueueReceiver.setTransformation`
				`AQjmsTopicSubscriber.setTransformation`
				`AQjmsTopicReceiver.setTransformation`

Dequeue a message from a queue/topic and specify navigation mode	`DBMS_AQ.dequeue`
	Specify navigation in `DEQUEUE_OPTIONS`
		`OCIAQDeq`
		Specify `OCI_ATTR_NAVIGATION` in `OCIAQDeqOptions`
			`<AQXmlReceive>` Specify `<navigation>` in `<consumer_options>`	`AQjmsQueueReceiver.setNavigationMode`
				`AQjmsTopicSubscriber.setNavigationMode`
				`AQjmsTopicReceiver.setNavigationMode`

Table 4–8 (Cont.) Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Receive Messages from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Dequeue a message from a single consumer queue	`DBMS_AQ.dequeue` Set `dequeue_mode` to `REMOVE` in `DEQUEUE_OPTIONS`	`OCIAQDeq` Set `OCI_ATTR_DEQ_MODE` to `REMOVE` in `OCIAQDeqOptions`	`<AQXmlReceive>`	`QueueReceiver.receive` or `QueueReceiver.receiveNoWait` or `AQjmsQueueReceiver.receiveNoData`
Dequeue a message from a multiconsumer queue/topic using subscription name	`DBMS_AQ.dequeue` Set `dequeue_mode` to `REMOVE` and set `consumer_name` to subscription name in `DEQUEUE_OPTIONS`	`OCIAQDeq` Set `OCI_ATTR_DEQ_MODE` to `REMOVE` and set `OCI_ATTR_CONSUMER_NAME` to subscription name in `OCIAQDeqOptions`	`<AQXmlReceive>` Specify `<consumer_name>` in `<consumer_options>`	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	`DBMS_AQ.dequeue`
	Set `dequeue_mode` to `REMOVE` and set `consumer_name` to recipient name in `DEQUEUE_OPTIONS`
		`OCIAQDeq`
		Set `OCI_ATTR_DEQ_MODE` to `REMOVE` and set `OCI_ATTR_CONSUMER_NAME` to recipient name in `OCIAQDeqOptions`
			`<AQXmlReceive>` Specify `<consumer_name>` in `<consumer_options>`	Create a `TopicReceiver` on the topic using the recipient name, then
				`AQjmsSession.createTopicReceiver`
				`AQjmsTopicReceiver.receive` or
				`AQjmsTopicReceiver.receiveNoWait` or
				`AQjmsTopicReceiver.receiveNoData`

Table 4–9 Comparison of Oracle Streams AQ Programmatic Environments: Operational Interface—Register to Receive Messages Asynchronously from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Receive messages asynchronously from a single-consumer queue	Define a PL/SQL callback procedure
	Register it using `DBMS_AQ.REGISTER`
		`OCISubscriptionRegister`
		Specify `queue_name` as subscription name
		`OCISubscriptionEnable`
			<AQXmlRegister> Specify queue name in <destination> and notification mechanism in <notify_url>	Create a `QueueReceiver` on the queue, then
				`QueueReceiver.setMessageListener`

Receive messages asynchronously from a multiconsumer queue/topic	Define a PL/SQL callback procedure
	Register it using `DBMS_AQ.REGISTER`
		`OCISubscriptionRegister`
		Specify `queue:OCI_ATTR_CONSUMER_NAME` as subscription name
		`OCISubscriptionEnable`
			<AQXmlRegister> Specify queue name in <destination>, consumer in <consumer_name> and notification mechanism in <notify_url>	Create a `TopicSubscriber` or `TopicReceiver` on the topic, then
				`TopicSubscriber.setMessageListener`

Listen for messages on multiple queues/topics	-	-	-	-
Listen for messages on one (many) single-consumer queues	`DBMS_AQ.LISTEN` Use `agent_name` as `NULL` for all agents in `agent_list`	`OCIAQListen` Use `agent_name` as `NULL` for all agents in `agent_list`	Not supported	Create multiple `QueueReceivers` on a `QueueSession`, then `QueueSession.setMessageListener`
Listen for messages on one (many) multiconsumer queues/Topics	`DBMS_AQ.LISTEN` Specify `agent_name` for all agents in `agent_list`	`OCIAQListen` Specify `agent_name` for all agents in `agent_list`	Not supported	Create multiple `TopicSubscribers` or `TopicReceivers` on a `TopicSession`, then `TopicSession.setMessageListener`

Part II

Managing and Tuning Oracle Streams AQ

Part II describes how to manage and tune your Oracle Streams Advanced Queuing (AQ) application.

This part contains the following chapters:

- Chapter 5, "Managing Oracle Streams AQ"
- Chapter 6, "Oracle Streams AQ Performance and Scalability"

5

Managing Oracle Streams AQ

This chapter discusses topics related to managing Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- Oracle Streams AQ Compatibility Parameters
- Queue Security and Access Control
- Queue Table Export-Import
- Oracle Enterprise Manager Support
- Using Oracle Streams AQ with XA
- Restrictions on Queue Management
- Managing Propagation
- 8.0-Compatible Queues

Oracle Streams AQ Compatibility Parameters

For 8.1-compatible 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 the following features:

- Queue-level access control
- Nonpersistent queues

  Database compatibility should be 8.1 or higher for creating non-persistent queues.
- 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 Streams AQ Security
- Queue Security
- Queue Privileges and Access Control
- OCI Applications and Queue Access
- Security Required for Propagation

Oracle Streams AQ 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. Users also need the MANAGE ANY QUEUE privilege to create and manage queues in other schemas.

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 Streams AQ 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 on 8.1-compatible or higher queues.

Your database administrator has the option of granting the system privileges ENQUEUE ANY QUEUE and DEQUEUE ANY QUEUE, 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 Streams AQ Object Types

All internal Oracle Streams AQ objects are now accessible to PUBLIC.

Queue Security

Oracle Streams AQ administrators of Oracle Database can create 8.1-compatible or higher queues. All 8.1 security features are enabled for 8.1-compatible or higher queues. Oracle Streams AQ 8.1 security features work only with 8.1-compatible or higher queues. When you create queues, the default value of the compatible parameter in DBMS_AQADM.CREATE_QUEUE_TABLE is 8.1.3 if the database compatibility is less than 10.0. If database compatibility is 10.1, then the default value of the compatible parameter is also 10.0.

The AQ_ADMINISTRATOR_ROLE role is supported for 8.1-compatible or higher queues. To enqueue/dequeue on 8.1-compatible or higher queues, users need EXECUTE rights on DBMS_AQ and either enqueue/dequeue privileges on target queues or ENQUEUE ANY QUEUE/DEQUEUE ANY QUEUE system privileges.

Queue Privileges and Access Control

You can grant or revoke privileges at the object level on 8.1- compatible or higher queues. You can also grant or revoke various system-level privileges. Table 5–1 lists all common Oracle Streams AQ operations and the privileges needed to perform these operations for an 8.1-compatible or higher queue.

Table 5–1 Operations and Required Privileges for 8.1-compatible and Higher Queues

Operation(s)	Privileges Required
CREATE/DROP/MONITOR own queues	Must be granted EXECUTE rights on DBMS_AQADM. No other privileges needed.
CREATE/DROP/MONITOR any queues	Must be granted EXECUTE rights on DBMS_AQADM and be granted AQ_ADMINISTRATOR_ROLE by another user who has been granted this role (SYS and SYSTEM are the first granters of AQ_ADMINISTRATOR_ROLE)
ENQUEUE/ DEQUEUE to own queues	Must be granted EXECUTE rights on DBMS_AQ. No other privileges needed.
ENQUEUE/ DEQUEUE to another's queues	Must be granted EXECUTE rights on DBMS_AQ and be granted privileges by the owner using DBMS_AQADM.GRANT_QUEUE_PRIVILEGE.
ENQUEUE/ DEQUEUE to any queues	Must be granted EXECUTE rights on DBMS_AQ and be granted ENQUEUE ANY QUEUE or DEQUEUE ANY QUEUE system privileges by an Oracle Streams AQ administrator using DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE.

OCI Applications and Queue Access

For an **Oracle Call Interface** (OCI) application to access an 8.1-compatible or higher 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 if the queue he intends to access is an 8.1-compatible or higher queue.

Security Required for Propagation

Oracle Streams AQ 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 Streams AQ privileges must be granted.

See Also: "Propagation from Object Queues" on page 5-16

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.

Note: Oracle Streams AQ does not currently support the new Data Pump `expdp` and `impdp` utilities. Use the original `exp` and `imp` utilities for queue table export-import.

Note: 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 only 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 (for 8.1-compatible and higher queue tables)
- A history index-organized table (for 8.1-compatible and higher queue tables)

These tables are exported automatically during full database mode and user mode exports, but not during table mode export. See "Export Modes" on page 5-7.

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 a 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.

Oracle does not recommend table mode. If you must export a queue table in table mode, then you must export all related objects that belong to that queue table. For example, when exporting an 8.1-compatible or higher multiconsumer queue table named `MCQ`, you must also export the following tables:

- `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.

Note: Transportable tablespace export/import of tablespaces with queue tables across releases fails on import. The metadata import from the lower release fails with an error indicating that the tablespace is read only. The workaround is to make the tablespace read/write before importing the metadata.

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 (for 8.1-compatible or higher queue tables)
- A history IOT (for 8.1-compatible or higher queue tables)

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

Creating Oracle Streams AQ Administrators and Users

Example 5–1 shows how to create an Oracle Streams AQ Administrator named aqadm. The last two lines, which are optional, show how to grant this user EXECUTE privileges on the Oracle Streams AQ packages. This allows the user to run the package procedures from within a user procedure.

Example 5–1 Creating a User as an Oracle Streams AQ Administrator

```
CONNECT system/manager
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT AQ_ADMINISTRATOR_ROLE TO aqadm;
GRANT CONNECT, RESOURCE TO aqadm;

GRANT EXECUTE ON DBMS_AQADM TO aqadm;  --optional
GRANT EXECUTE ON DBMS_AQ TO aqadm;     --optional
```

The procedure to create Oracle Streams AQ users who create and access queues within their own schemas is similar to "Creating a User as an Oracle Streams AQ Administrator", except you do not grant the AQ_ADMINISTRATOR_ROLE. Example 5–2 shows how to create an own-schema user named aquser1. The last two lines, which are optional, show how to grant this user EXECUTE privileges on the Oracle Streams AQ packages. This allows the user to run the package procedures from within a user procedure.

Example 5–2 Creating a User to Create and Access Queues in Own Schema

```
CONNECT system/manager
CREATE USER aquser1 IDENTIFIED BY aquser1;
GRANT CONNECT, RESOURCE TO aquser1;
GRANT EXECUTE ON DBMS_AQADM to aquser1;
```

The procedure to create an Oracle Streams AQ user who does not create queues but uses a queue in another schema is identical to that for the own-schema user, as shown in Example 5–3 for user aquser2. But you must also grant object level privileges in the other schema. Example 5–4 does this for aquser2 in the aquser1 schema. However, this applies only to queues defined using 8.1-compatible or higher queue tables.

Example 5–3 Creating a User to Access Queues in Another Schema

```
CONNECT system/manager
CREATE USER aquser2 IDENTIFIED BY aquser2;
GRANT CONNECT, RESOURCE TO aquser2;
GRANT EXECUTE ON DBMS_AQ TO aquser2;
```

For `aquser2` to access the queue `aquser1_q1` in `aquser1` schema, `aquser1` must run the following statements:

Example 5–4 Granting Queue Privileges in Another Queue

```
CONNECT aquser1/aquser1
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
  'ENQUEUE','aquser1_q1','aquser2',FALSE);
```

Oracle Enterprise Manager Support

Oracle Enterprise Manager supports most of the administrative functions of Oracle Streams AQ. Oracle Streams AQ 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 Streams AQ with XA

You must specify "`Objects=T`" in the `xa_open` string if you want to use the Oracle Streams AQ 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 Streams AQ 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 Streams AQ raw messages because Oracle Streams AQ provides a simple RAW buffer abstraction (although they are stored as LOBs).

When using the Oracle Streams AQ 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 Application Developer's Guide - Fundamentals* 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:

- Remote 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
- Tablespace Point-in-Time Recovery
- Nonpersistent Queues

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`.

Remote Subscribers

For this release, only 32 remote subscribers are allowed for each remote destination database.

DML Not Supported on Queue Tables or Associated IOTs

Oracle Streams AQ 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 Streams AQ 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 5–5.

Example 5–5 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 Streams AQ PL/SQL calls resolve synonyms on queues and queue tables. Although you can create synonyms, you should not apply them to the Oracle Streams AQ interface.

Tablespace Point-in-Time Recovery

Oracle Streams AQ 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 Streams AQ does support regular point-in-time recovery.

Nonpersistent Queues

Currently 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`.

Managing Propagation

Propagation makes use of the system queue `aq$_prop_notify_X`, where `X` is the instance number of the instance where the source queue of a schedule resides, for handling propagation run-time events. Messages in this queue are stored in the system table `aq$_prop_table_X`, where `X` is the instance number of the instance where the source queue of a schedule resides.

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

This section contains these topics:

- EXECUTE Privileges Required for Propagation
- The Number of Job Queue Processes
- Optimizing Propagation
- Message States During Client Requests for Enqueue
- Propagation from Object Queues
- Debugging Oracle Streams AQ Propagation Problems

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 does 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.

The Number of Job Queue Processes

The scheduling algorithm places the restriction that at least two job queue processes be available for propagation. If there are jobs unrelated to propagation, then more job queue processes are needed. If heavily loaded conditions (a large number of active schedules, all of which have messages to be propagated) are expected, then you should start a larger number of job queue processes and keep in mind the need for nonpropagation jobs as well. In a system that only has propagation jobs, two job queue processes can handle all schedules. However, with more job queue processes, messages are propagated faster. Because one job queue process can propagate messages from multiple schedules, it is not necessary to have the number of job queue processes equal to the number of schedules.

Optimizing Propagation

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.

Message States During Client Requests for Enqueue

Client requests for enqueue, **send** and publish requests, use the following methods:

- `AQXmlSend`—to enqueue to a single-consumer queue
- `AQXmlPublish`—to enqueue to multiconsumer queues/topics

In `message_header`, the `message_state` attribute represents the state of the message filled in automatically during dequeue, as follows:

- 0 (the message is ready to be processed)
- 1 (the message delay has not yet been reached)
- 2 (the message has been processed and is retained)
- 3 (the message has been moved to the **exception queue**)

Propagation from Object Queues

Propagation from object queues with BFILEs is supported in Oracle Database 10*g*. To be able to propagate object queues with BEFILEs, 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.

Note: Propagation of BFILES from object queues without specifying a database link is not supported.

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

Debugging Oracle Streams AQ Propagation Problems

See: Chapter 25, "Troubleshooting Oracle Streams AQ"

8.0-Compatible Queues

If you use 8.0-compatible queues and 8.1 or higher database compatibility, then the following features are not available:

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

To use these features, you should migrate to 8.1-compatible or higher queues.

See Also:

- "Security Required for Propagation" on page 5-5
- *Oracle Database Upgrade Guide*

Migrating To and From 8.0

To upgrade a 8.0-compatible queue table to an 8.1-compatible or higher queue table or to downgrade a 8.1-compatible or higher queue table to an 8.0-compatible queue table, use `DBMS_AQADM.MIGRATE_QUEUE_TABLE`.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE(
    queue_table        IN       VARCHAR2,
    compatible         IN       VARCHAR2)
```

Parameters

queue_table (IN VARCHAR2)
Specifies name of the queue table that is to be migrated.

compatible
Set to `8.1` to upgrade an 8.0 queue table to 8.1 compatibility. Set to `8.0` to downgrade an 8.1 queue table to 8.0 compatibility.

Example

You must set up the following data structures for the following example to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table            => 'qtable1',
   multiple_consumers     => TRUE,
   queue_payload_type     => 'aq.message_typ',
   compatible             =>'8.0');
```

Example 5–6   Upgrading an 8.0 Queue Table to an 8.1-Compatible Queue Table

```
EXECUTE DBMS_AQADM.MIGRATE_QUEUE_TABLE(
    queue_table  => 'qtable1',
   compatible   => '8.1');
```

Importing and Exporting with 8.0-Style Queues

Because the metadata tables contain ROWIDs of some rows in the queue table, the import and export processes generate a note about the ROWIDs being obsoleted when importing the metadata tables. This message can be ignored, because the queuing system automatically corrects the obsolete ROWIDs as a part of the import operation. However, if another problem is encountered while doing the import or export (such as running out of rollback segment space), then you should correct the problem and repeat the import or export.

Roles in 8.0

Access to Oracle Streams AQ operations in Oracle8 was granted to users through roles that provided execution privileges on the Oracle Streams AQ procedures. The fact that there was no control at the database object level when using Oracle8 meant that a user with the `AQ_USER_ROLE` could enqueue and dequeue to any queue in the system. For finer-grained access control, use 8.1-compatible or higher queue tables in an 8.1- compatible or higher database.

Oracle Streams AQ administrators of an 8.1-compatible or higher database can create queues with 8.0 compatibility. These queues are protected by the 8.0-compatible security features.

If you want to use 8.1 security features on a queue originally created in an 8.0 database, then the queue table must be converted to 8.1-compatible or higher by running `DBMS_AQADM.MIGRATE_QUEUE_TABLE` on the queue table.

> **See Also:** *PL/SQL Packages and Types Reference* for more information on `DBMS_AQADM.MIGRATE_QUEUE_TABLE`

If a database downgrade is necessary, then all 8.1-compatible or higher queue tables must be either converted back to 8.0 compatibility or dropped before the database downgrade can be carried out. During the conversion, all 8.1-compatible security features on the queues, like the object privileges, are dropped. When a queue is converted to 8.0-compatible, the 8.0-compatible security model applies to the queue, and only 8.0-compatible security features are supported.

Security with 8.0-Style Queues

The following Oracle Streams AQ security features and privilege equivalences are supported with 8.0-compatible queues:

- `AQ_USER_ROLE`

  The grantee is given the `EXECUTE` right of `DBMS_AQ` through the role.

- `AQ_ADMINISTRATOR_ROLE`
- `EXECUTE` right on `DBMS_AQ`

  `EXECUTE` right on `DBMS_AQ` should be granted to developers who write Oracle Streams AQ applications in PL/SQL.

Access to Oracle Streams AQ Object Types

The procedure `grant_type_access` was made obsolete in release 8.1.5 for 8.0-compatible queues.

OCI Application Access to 8.0-Style Queues

For an OCI application to access an 8.0-compatible queue, the session user must be granted the `EXECUTE` rights of `DBMS_AQ`.

Pluggable Tablespaces and 8.0-Style Multiconsumer Queues

A tablespace that contains 8.0-compatible multiconsumer queue tables should not be transported using the pluggable tablespace mechanism. The mechanism does work, however, with tablespaces that contain only single-consumer queues as well as 8.1 compatible or higher multiconsumer queues. Before you can export a tablespace in pluggable mode, you must alter the tablespace to read-only mode. If you try to import a read-only tablespace that contains 8.0-compatible multiconsumer queues, then you get an Oracle Streams AQ error indicating that you cannot update the queue table index at import time.

Autocommit Features in the DBMS_AQADM Package

The autocommit parameters in the CREATE_QUEUE_TABLE, DROP_QUEUE_TABLE, CREATE_QUEUE, DROP_QUEUE, and ALTER_QUEUE calls of the DBMS_AQADM package are deprecated for 8.1.5 and subsequent releases. Oracle continues to support this parameter in the interface for backward compatibility.

6

Oracle Streams AQ Performance and Scalability

This chapter discusses performance and scalability issues relating to Oracle Streams Advanced Queuing (AQ).

This chapter contains the following topics:

- Performance Overview
- Basic Tuning Tips
- Propagation Tuning Tips

Performance Overview

Queues are stored in database tables. The performance characteristics of **queue** operations 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 `SELECT`, `DELETE`, and `UPDATE` operations on similar tables.

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

Oracle Streams AQ and Oracle Real Application Clusters

Oracle Real Application Clusters 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 Oracle Real Application Clusters 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.

Oracle Streams AQ 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 *PL/SQL Packages and Types Reference* for more information on the wait option

Basic Tuning Tips

Oracle Streams AQ table layout is similar to a layout with ordinary database tables and indexes.

See Also: *Oracle Database Performance Tuning Guide* for tuning recommendations

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.

I/O Configuration

Because Oracle Streams AQ 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.

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 is higher for enqueuers than for dequeuers when the number of processes is increased. Usually, the dequeue operations throughput is much less than the enqueue operation (INSERT) throughput, because dequeue operations perform SELECT, DELETE, and UPDATE.

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.

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 Streams AQ 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 Streams AQ 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.

Part III

Oracle Streams AQ: Sample Application

Part III describes the Oracle Streams Advanced Queuing (AQ) sample application used for most examples in this manual.

This part contains the following chapters:

- Chapter 7, "Oracle Streams AQ Sample Application"

7

Oracle Streams AQ Sample Application

This chapter discusses the features of Oracle Streams Advanced Queuing (AQ) in a sample application based on a hypothetical company called BooksOnLine.

This chapter contains these topics:

- A Sample Application
- General Features of Oracle Streams AQ
- System-Level Access Control
- Queue-Level Access Control
- Message Format Transformation
- Structured Payloads
- Nonpersistent Queues
- Retention and Message History
- Publish/Subscribe Support
- Oracle Real Application Clusters Support
- Statistics Views and Oracle Streams AQ
- Internet Access for Oracle Streams AQ
- Enqueue Features
- Dequeue Features
- Exception Handling
- Asynchronous Notifications
- Propagation Features

Note: For further helpful examples on using Oracle Streams AQ, search for the "Oracle By Example Series" at the OTN Web site:

`http://otn.oracle.com/index.html`

A Sample Application

The operations of a large bookseller, BooksOnLine, are based on an online book ordering system that automates activities across the various departments involved in the sale. The front end of the system is an order entry application used to enter new orders. Incoming orders are processed by an order processing application that validates and records the order. Shipping departments located at regional warehouses are responsible for ensuring that orders are shipped on time.

There are three regional warehouses: one serving the East Region, another serving the West Region, and a third warehouse for shipping International orders. After an order is shipped, the order information is routed to a central billing department that handles payment processing. The customer service department, located at a separate site, is responsible for maintaining order status and handling inquiries.

The features of Oracle Streams AQ are exemplified in the BooksOnLine scenario to demonstrate the possibilities of Oracle Streams AQ technology. The sample code is provided in Appendix A, "Scripts for Implementing BooksOnLine".

General Features of Oracle Streams AQ

This section contains these topics:

- System-Level Access Control
- Queue-Level Access Control
- Message Format Transformation
- Structured Payloads
- Creating Queues with XMLType Payloads
- Nonpersistent Queues
- Retention and Message History
- Publish/Subscribe Support

- Oracle Real Application Clusters Support
- Propagation Features

System-Level Access Control

Oracle Streams AQ 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 Streams AQ administrative and operational interfaces on any queue in the database. This simplifies the 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 5-10

PL/SQL (DBMS_AQADM Package): Scenario and Code

In the BooksOnLine application, the DBA creates `BOLADM`, the BooksOnLine Administrator account, as the queue administrator of the database. This allows `BOLADM` to create, drop, manage, and monitor queues in the database. If PL/SQL packages are needed in the BOLADM schema for applications to **enqueue** and **dequeue**, then the DBA should grant `ENQUEUE_ANY` and `DEQUEUE_ANY` system privileges to BOLADM:

Example 7–1 Creating BOLADM, the BooksOnLine Administrator Account

```
CREATE USER BOLADM IDENTIFIED BY BOLADM;
GRANT CONNECT, RESOURCE, aq_administrator_role TO BOLADM;
GRANT EXECUTE ON DBMS_AQ TO BOLADM;
GRANT EXECUTE ON DBMS_AQADM TO BOLADM;
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','BOLADM',FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('DEQUEUE_ANY','BOLADM',FALSE);
```

If using the Java AQ **API**, then BOLADM must be granted `EXECUTE` privileges on the DBMS_AQIN package:

```
GRANT EXECUTE ON DBMS_AQIN to BOLADM;
```

In the application, Oracle Streams AQ propagators populate messages from the Order Entry (OE) schema to:

- The Western Sales (WS) schema
- Eastern Sales (ES) schema

- Worldwide Sales (TS) schema

The WS, ES, and TS schemas in turn populate messages to:

- Customer Billing (CB) schema
- Customer Service (CS) schema

Hence the OE, WS, ES, and TS schemas all host queues that serve as the source queues for the propagators.

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.

Example 7–2 Granting ENQUEUE_ANY System Privilege to All Schemas Hosting a Source Queue

To simplify administration, all schemas that host a source queue in the BooksOnLine application are granted the ENQUEUE_ANY system privilege:

```
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','OE',FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','WS',FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','ES',FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','TS',FALSE);
```

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 QUEUE privilege, or be granted the rights to enqueue to the destination queue. If the login user in the database link also owns the queue tables at the destination, then no explicit privilege grant is needed.

Visual Basic (OO4O): Example Code

Use the dbexecutesql interface from the database for this functionality.

Java (JDBC): Example Code

No example is provided with this release.

Queue-Level Access Control

Oracle Streams AQ 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 must grant only minimal access privileges to the applications that run outside the queue schema. The supported access privileges on a queue are ENQUEUE, DEQUEUE and ALL.

See Also: "Oracle Enterprise Manager Support" on page 5-10

Scenario

The BooksOnLine application processes customer billings in its CB (Customer Billing) and CBADM schemas. The CB schema hosts the customer billing application and the CBADM schema hosts all related billing data stored as queue tables.

To protect the billing data, the billing application and the billing data reside in different schemas. The billing application is allowed only to dequeue messages from CBADM_shippedorders_que, the shipped order queue. It processes the messages, and then enqueues new messages into CBADM_billedorders_que, the billed order queue.

To protect the queues from other unauthorized operations from the application, the following two grant calls are needed:

Example 7–3 PL/SQL (DBMS_AQADM Package): Granting Dequeue Privilege on Shipped Orders Queue to CB Application

```
/* Grant dequeue privilege on the shipped orders queue to the Customer
   Billing application. The CB application retrieves orders that are shipped but
   not billed from the shipped orders queue. */
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
   'DEQUEUE','CBADM_shippedorders_que', 'CB', FALSE);

/* Grant enqueue privilege on the billed orders queue to Customer Billing
   application. The CB application is allowed to put billed orders into this
   queue after processing the orders. */

EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
   'ENQUEUE', 'CBADM_billedorders_que', 'CB', FALSE);
```

Visual Basic (OO4O): Example Code

Use the dbexecutesql interface from the database for this functionality.

Example 7–4 Java (JDBC): Granting Dequeue Privilege on Shipped Orders Queue to CB Application

```
public static void grantQueuePrivileges(Connection db_conn)
{
    AQSession  aq_sess;
    AQQueue    sh_queue;
    AQQueue    bi_queue;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* Grant dequeue privilege on the shipped orders queue to the Customer
           Billing application. The CB application retrieves orders that are
           shipped but not billed from the shipped orders queue. */

        sh_queue = aq_sess.getQueue("CBADM", "CBADM_shippedorders_que");

        sh_queue.grantQueuePrivilege("DEQUEUE", "CB", false);

        /* Grant enqueue privilege on the billed orders queue to Customer
           Billing application. The CB application is allowed to put billed
           orders into this queue after processing the orders. */

        bi_queue = aq_sess.getQueue("CBADM", "CBADM_billedorders_que");

        bi_queue.grantQueuePrivilege("ENQUEUE", "CB", false);
    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }
}
```

Message Format Transformation

You can define **transformation** mappings between different **message** payload types. Transformation mappings are defined as SQL expressions that can include PL/SQL functions (including callouts) and Java stored procedures. Only one-to-one message transformations are supported. The transformation engine is tightly integrated with Oracle Streams AQ to facilitate transformation of messages when they move through the database messaging system.

Transformation mappings can be used during enqueue, dequeue, and **propagation** operations. To use a transformation at:

- Enqueue, the mapping is specified in the enqueue options.
- Dequeue, the mapping is specified either in the dequeue options or when you add a **subscriber**. A mapping specified in the dequeue options overrides a mapping specified with ADD_SUBSCRIBER.
- Propagation, the mapping is specified when you add a subscriber.

Example 7–5 PL/SQL (DBMS_TRANSFORM Package): Creating Types for the OE Application

In the BooksOnLine application, assume that the order type is represented differently in the Order Entry (OE) and the Shipping applications. The order type and other types for the Order Entry application are created as follows:

```
CREATE OR REPLACE TYPE order_typ AS object (
        orderno         number,
        status          varchar2(30),
        ordertype       varchar2(30),
        orderregion     varchar2(30),
        custno          number,
        paymentmethod   varchar2(30),
        items           orderitemlist_vartyp,
        ccnumber        varchar2(20),
        order_date      date);

CREATE OR REPLACE TYPE customer_typ AS object (
        custno          number,
        custid          varchar2(20),
        name            varchar2(100),
        street          varchar2(100),
        city            varchar2(30),
        state           varchar2(2),
        zip             number,
```

```
        country          varchar2(100));

CREATE OR REPLACE TYPE book_typ AS object (
        title            varchar2(100),
        authors          varchar2(100),
        ISBN             varchar2(20),
        price            number);

CREATE OR REPLACE TYPE orderitem_typ AS object (
        quantity         number,
        item             book_typ,
        subtotal         number);

CREATE OR REPLACE TYPE orderitemlist_vartyp AS varray (20) of
orderitem_typ;
```

Example 7–6 Creating Types for the Shipping Application

```
CREATE OR REPLACE TYPE order_typ_sh AS object (
        orderno          number,
        status           varchar2(30),
        ordertype        varchar2(30),
        orderregion      varchar2(30),
        customer         customer_typ_sh,
        paymentmethod    varchar2(30),
        items            orderitemlist_vartyp,
        ccnumber         varchar2(20),
        order_date       date);

CREATE OR REPLACE TYPE customer_typ_sh AS object (
        custno           number,
        name             varchar2(100),
        street           varchar2(100),
        city             varchar2(30),
        state            varchar2(2),
        zip              number);

CREATE OR REPLACE TYPE book_typ_sh AS object (
        title            varchar2(100),
        authors          varchar2(100),
        ISBN             varchar2(20),
        price            number);

CREATE OR REPLACE TYPE orderitem_typ_sh AS object (
```

```
        quantity        number,
        item            book_typ,
        subtotal        number);

CREATE OR REPLACE TYPE orderitemlist_vartyp_sh AS varray (20) of
orderitem_typ_sh;
```

The Overseas Shipping application uses an `XMLType` attribute.

Creating Transformations

You can create transformations by creating a single PL/SQL function or by creating an expression for each target type attribute.

Creating a Single PL/SQL Function

This 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.

Example 7–7 DBMS_TRANSFORM.create transformation: Creating a Single PL/SQL Function to Return Target Type

```
EXECUTE DBMS_TRANSFORM.CREATE_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           from_schema => 'OE', from_type => 'order_typ',
           to_schema => 'WS', to_type => 'order_typ_sh',
           transformation(
               'WS.order_typ_sh(source.user_data.orderno,
                                source.user_data.status,
                                source.user_data.ordertype,
                                source.user_data.orderregion,

WS.get_customer_info(source.user_data.custno),
                                source.user_data.paymentmethod,
                                source.user_data.items,
                                source.user_data.ccnumber,
                                source.user_data.order_date)');
```

In the BooksOnline application, assume that the Overseas Shipping site represents the order as an XMLType payload. The Order Entry site represents the order as an Oracle object, `ORDER_TYP`. Because the Overseas Shipping site subscribes to messages in the `OE_BOOKEDORDERS_QUE` queue, a transformation is applied before messages are propagated from the Order Entry site to the Overseas Shipping site.

Example 7–8 Applying a Transformation Before Messages are Propagated from the OE Site

The transformation is defined as follows:

```
CREATE OR REPLACE FUNCTION CONVERT_TO_ORDER_XML(input_order TYPE OE.ORDER_TYP)
RETURN XMLType AS
   new_order XMLType;
BEGIN
   select SYS_XMLGEN(input_order) into new_order from dual;
   RETURN new_order;
END CONVERT_TO_ORDER_XML;

EXECUTE DBMS_TRANSFORM.CREATE_TRANSFORMATION(
   schema =>         'TS',
   name   =>         'OE2XML',
   from_schema =>    'OE',
   from_type =>      'ORDER_TYP',
   to_schema =>      'SYS',
   to_type =>        'XMLTYPE',
   transformation => 'CONVERT_TO_ORDER_XML(source.user_data)');

/*  Add a rule-based subscriber for Overseas Shipping to the Booked Orders
queues with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
 subscriber     aq$_agent;
BEGIN
 subscriber := aq$_agent('Overseas_Shipping','TS.TS_bookedorders_que',null);

 DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name     => 'OE.OE_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.orderregion = ''INTERNATIONAL'''
        transformation => 'TS.OE2XML');
END;
```

Creating an Expression for Each Target Type Attribute

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

Example 7–9 DBMS_TRANSFORM.create_transformation: Creating an Expression for Each Target Type Attribute

```
/* first create the transformation without any transformation expression*/
EXECUTE DBMS_TRANSFORM.CREATE_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           from_schema => 'OE', from_type => 'order_typ',
           to_schema => 'WS', to_type => 'order_typ_sh');

/* specify each attribute of the target type as a function of the source type*/
EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.orderno');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.status');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.ordertype');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.orderregion');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation =>
'WS.get_customer_info(source.user_data.custno)');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.payment_method');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
           schema => 'OE', name => 'OE2WS',
           attribute_number => 1,
           transformation => 'source.user_data.orderitemlist_vartyp');
```

```
EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
          schema => 'OE', name => 'OE2WS',
          attribute_number => 1,
          transformation => 'source.user_data.ccnumber');

EXECUTE DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
          schema => 'OE', name => 'OE2WS',
          attribute_number => 1,
          transformation => 'source.user_data.order_date');
```

Visual Basic (OO4O): Example Code

No example is provided with this release.

Java (JDBC): Example Code

No example is provided with this release.

Structured Payloads

With Oracle Streams AQ, you can use object types to structure and manage the payload of messages. The object-relational capabilities of Oracle Database provide a rich set of data types that range from traditional relational data types to user-defined types.

Using strongly typed content, that is, content whose format is defined by an Oracle **object type** system, makes the following features available:

- Content-based routing

  Oracle Streams AQ can examine the content and automatically route messages to another queue based on content.

- Content-based subscription

  A publish and subscribe system can be built on top of a messaging system so that you can create subscriptions based on content.

- XML

  Use the flexibility and extensibility of XML with Oracle Streams AQ messages. `XMLType` has additional operators to simplify the use of XML data. Operators include `XMLType.existsNode()` and `XMLType.extract()`.

You can also create payloads that contain Oracle objects with `XMLType` attributes. These can be used for transmitting and storing messages that contain XML documents. By defining Oracle objects with `XMLType` attributes, you can do the following:

- Store more than one type of XML document in the same queue. The documents are stored internally as **CLOB** objects.
- Query `XMLType` attributes using the methods `XMLType.existsNode()`, `XMLType.extract()`, and so on.

> **See Also:** *Oracle XML DB Developer's Guide*

Example 7–10 PL/SQL (DBMS_AQADM Package): Creating Various Structured Payloads

The BooksOnLine application uses a rich set of data types to model book orders as message content.

Customers are modeled as an object type called `customer_typ`.

```
CREATE OR REPLACE TYPE customer_typ AS OBJECT (
        custno          NUMBER,
        name            VARCHAR2(100),
        street          VARCHAR2(100),
        city            VARCHAR2(30),
        state           VARCHAR2(2),
        zip             NUMBER,
        country         VARCHAR2(100));
```

Books are modeled as an object type called `book_typ`.

```
CREATE OR REPLACE TYPE book_typ AS OBJECT (
        title           VARCHAR2(100),
        authors         VARCHAR2(100),
        ISBN            NUMBER,
        price           NUMBER);
```

An order item that represents an order line item is modeled as an object type called `orderitem_typ`. An order item is a nested type that includes the book type.

```
CREATE OR REPLACE TYPE orderitem_typ AS OBJECT (
        quantity        NUMBER,
        item            BOOK_TYP,
        subtotal        NUMBER);
```

An order item list is used to represent a list of order line items and is modeled as a **VARRAY** of order items.

```
CREATE OR REPLACE TYPE orderitemlist_vartyp AS VARRAY (20) OF orderitem_typ;
```

An order is modeled as an object type called `order_typ`. The order type is a composite type that includes nested object types defined earlier. The order type captures details of the order, the customer information, and the item list.

```
CREATE OR REPLACE TYPE order_typ as object (
        orderno         NUMBER,
        status          VARCHAR2(30),
        ordertype       VARCHAR2(30),
        orderregion     VARCHAR2(30),
        customer        CUSTOMER_TYP,
        paymentmethod   VARCHAR2(30),
        items           ORDERITEMLIST_VARTYP,
        total           NUMBER);
```

Some queues in the BooksOnline application model an order using an `XMLType` payload.

Visual Basic (OO4O): Example Code

Use the `dbexecutesql` interface from the database for this functionality.

Example 7–11 Java (JDBC): Generating Java Classes to Map Structured Payloads to SQL Types

After creating the types, use JPublisher to generate Java classes that map to the SQL types.

1. Create an input file `jaqbol.typ` for JPublisher with the following lines:

   ```
   TYPE boladm.customer_typ AS Customer
   TYPE boladm.book_typ AS Book
   TYPE boladm.orderitem_typ AS OrderItem
   TYPE boladm.orderitemlist_vartyp AS OrderItemList
   TYPE boladm.order_typ AS Order
   ```

2. Run JPublisher with the following arguments:

   ```
   jpub -input=jaqbol.typ -user=boladm/boladm -case=mixed -methods=false
   -compatible=CustomDatum
   ```

   This creates Java classes Customer, Book, OrderItem, and OrderItemList that map to the SQL object types created earlier.

3. Load the Java AQ driver and create a JDBC connection:

```
public static Connection loadDriver(String user, String passwd)
{
   Connection db_conn = null;
   try
   {
         Class.forName("oracle.jdbc.driver.OracleDriver");

      /* your actual hostname, port number, and SID will
      vary from what follows. Here we use 'dlsun736,' '5521,'
      and 'test,' respectively: */

      db_conn =
               DriverManager.getConnection(
               "jdbc:oracle:thin:@dlsun736:5521:test",
               user, passwd);

      System.out.println("JDBC Connection opened ");
      db_conn.setAutoCommit(false);

      /* Load the Oracle Database AQ driver: */
      Class.forName("oracle.AQ.AQOracleDriver");

      System.out.println("Successfully loaded AQ driver ");
   }
   catch (Exception ex)
   {
      System.out.println("Exception: " + ex);
      ex.printStackTrace();
   }
   return db_conn;
```

Creating Queues with XMLType Payloads

You can create queues with `XMLType` payloads. These can be used for transmitting and storing messages that contain XML documents. By defining Oracle objects with `XMLType` attributes, you can do the following:

- Store more than one type of XML document in the same queue. The documents are stored internally as CLOBs.
- Selectively dequeue messages with `XMLType` attributes using the operators `XMLType.existsNode()`, `XMLType.extract()`, and so on.

> **See Also:** *Oracle XML DB Developer's Guide* for details on `XMLType` operations

- Define transformations to convert Oracle objects to `XMLType`.
- Define rule-based subscribers that query message content using `XMLType` methods such as `XMLType.existsNode()` and `XMLType.extract()`.

Example 7–12 DBMS_AQADM: Creating a Queue Table and Queue for an XMLType Order

In the BooksOnline application, assume that the Overseas Shipping site represents the order as `XMLType`. The Order Entry (OE) site represents the order as an Oracle object, `ORDER_TYP`. The Overseas **queue table** and queue are created as follows:

```
BEGIN
DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'TS_orders_pr_mqtab',
   comment            => 'Overseas Shipping MultiConsumer Orders queue table',
   multiple_consumers => TRUE,
   queue_payload_type => 'SYS.XMLtype',
   compatible         => '8.1');
END;

BEGIN
DBMS_AQADM.CREATE_QUEUEcreate_queue (
   queue_name   => 'TS_bookedorders_que',
   queue_table  => 'TS_orders_pr_mqtab');
END;
```

Example 7–13 Transforming Messages Before Propagation to the Overseas Shipping Site

Because the representation of orders at the Overseas Shipping site is different from the representation of orders at the Order Entry site, a transformation is applied before messages are propagated from the Order Entry site to the Overseas Shipping site.

```
/*  Add a rule-based subscriber (for Overseas Shipping) to the Booked Orders
 queues with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('Overseas_Shipping','TS.TS_bookedorders_que',null);
```

```
  DBMS_AQADM.ADD_SUBSCRIBER(
   queue_name     => 'OE.OE_bookedorders_que',
   subscriber     => subscriber,
   rule           => 'tab.user_data.orderregion = ''INTERNATIONAL''',
   transformation => 'TS.OE2XML');
END;
```

> **See Also:** "Creating Transformations" on page 7-9 for more details on defining transformations that convert the type used by the Order Entry application to the type used by Overseas Shipping

Example 7–14 DBMS_AQ: Dequeuing XMLType Messages to Process Orders for Canadian Customers

Assume that an application processes orders for customers in Canada. This application can dequeue messages using the following procedure:

```
/*  Create procedures to enqueue into single-consumer queues: */
create or replace procedure get_canada_orders() as
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           XMLtype;
no_messages              exception;
pragma exception_init    (no_messages, -25228);
new_orders               BOOLEAN := TRUE;

begin
        dopt.wait := 1;

/* Specify dequeue condition to select Orders for Canada */
        dopt.deq_condition := 'tab.user_data.extract(
''/ORDER_TYP/CUSTOMER/COUNTRY/text()'').getStringVal()=''CANADA''';

            dopt.consumer_name : = 'Overseas_Shipping';

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name         => 'TS.TS_bookedorders_que',
                dequeue_options    => dopt,
                message_properties => mprop,
                payload            => deq_order_data,
                msgid              => deq_msgid);
```

```
            commit;

            dbms_output.put_line(' Order for Canada - Order: ' ||
                                   deq_order_data.getStringVal());

          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE ORDERS  ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;
end;
```

Nonpersistent Queues

A message in a **nonpersistent** queue is not stored in a database table. You create a nonpersistent queue, which can be either a single-consumer or multiconsumer type. These queues are created in a system-created queue table (`AQ$_MEM_SC` for single-consumer queues and `AQ$_MEM_MC` for multiconsumer queues) in the schema specified by the `create_np_queue` command. Subscribers can be added to the multiconsumer queues. Nonpersistent queues can be destinations for propagation.

See Also: "Creating a Nonpersistent Queue" on page 8-16

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. You retrieve messages from a nonpersistent queue through the **asynchronous** notification mechanism, registering for the notification (using `OCISubscriptionRegister` or `DBMS_AQADM.REGISTER`) for the queues you are interested in.

See Also: "Registering for Notification" on page 10-39

When a message is enqueued into a queue, it is delivered to clients with active registrations for the queue. The messages are published to the interested clients without incurring the overhead of storing them in the database.

See Also:

- "DBMS_AQADM.REGISTER" in *PL/SQL Packages and Types Reference*
- "OCISubscriptionRegister" in *Oracle Call Interface Programmer's Guide*

Scenario

Assume that there are three application processes servicing user requests at the Order Entry system. The connection dispatcher shares out connection requests from the application processes. It attempts to maintain a count of the number of users logged on to the Order Entry system and the number of users for each application process. The application processes are named APP1, APP2, and APP3. Application process failures are not considered in this example.

Using nonpersistent queues meets the requirements in this scenario. When a user logs on to the database, the application process enqueues to the multiconsumer nonpersistent queue, LOGIN_LOGOUT, with the application name as the **consumer** name. The same process occurs when a user logs out. To distinguish between the two events, the correlation of the message is LOGIN for logins and LOGOUT for logouts.

The callback function counts the login and logout events for each application process.

Note: The dispatcher process must connect to the database only for registering the subscriptions. The notifications themselves can be received while the process is disconnected from the database.

Example 7–15 PL/SQL (DBMS_AQADM). Creating Multiconsumer Nonpersistent Queues in OE Schema

```
CONNECT oe/oe;
/* Create the Object Type/ADT adtmsg */
CREATE OR REPLACE TYPE adtmsg AS OBJECT (id NUMBER, data VARCHAR2(4000));

/* Create the multiconsumer nonpersistent queue in OE schema: */
EXECUTE DBMS_AQADM.CREATE_NP_QUEUE(queue_name         => 'LOGIN_LOGOUT',
                                   multiple_consumers => TRUE);

/* Enable the queue for enqueue and dequeue: */
```

```
EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'LOGIN_LOGOUT');

/* Nonpersistent Queue Scenario - procedure to be executed upon login: */
CREATE OR REPLACE PROCEDURE  User_Login(app_process IN VARCHAR2)
AS
  msgprop        dbms_aq.message_properties_t;
  enqopt         dbms_aq.enqueue_options_t;
  enq_msgid      RAW(16);
  payload        RAW(1);
BEGIN
  /* Visibility must always be immediate for NonPersistent queues */
  enqopt.visibility:=dbms_aq.IMMEDIATE;
  msgprop.correlation:= 'LOGIN';
  msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
  /* payload is NULL */
  dbms_aq.enqueue(
        queue_name         => 'LOGIN_LOGOUT',
        enqueue_options    => enqopt,
        message_properties => msgprop,
        payload            => payload,
        msgid              => enq_msgid);

END;

/* Nonpersistent queue scenario - procedure to be executed upon logout: */
CREATE OR REPLACE PROCEDURE  User_logout(app_process IN VARCHAR2)
AS
  msgprop        dbms_aq.message_properties_t;
  enqopt         dbms_aq.enqueue_options_t;
  enq_msgid      RAW(16);
  payload        adtmsg;
BEGIN
  /* Visibility must always be immediate for NonPersistent queues: */
  enqopt.visibility:=dbms_aq.IMMEDIATE;
  msgprop.correlation:= 'LOGOUT';
  msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
  /* Payload is NOT NULL: */
payload := adtmsg(1, 'Logging Off');

dbms_aq.enqueue(
        queue_name         => 'LOGIN_LOGOUT',
        enqueue_options    => enqopt,
        message_properties => msgprop,
        payload            => payload,
        msgid              => enq_msgid);
```

```
 END;
/


/* If there is a login at APP1, then enqueue a message into 'login_logout' with
   correlation 'LOGIN': */
EXECUTE User_login('APP1');

/* If there is a logout at APP3, then enqueue a message into 'login_logout' with
   correlation 'LOGOUT' and payload adtmsg(1, 'Logging Off'): */
EXECUTE User_logout('APP3');

 /* The OCI program which waits for notifications: */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#ifdef WIN32COMMON
#define sleep(x)   Sleep(1000*(x))
#endif

/* LOGIN / password:  */
static text *username = (text *) "OE";
static text *password = (text *) "OE";

/* The correlation strings of messages: */
static char  *login = "LOGIN";
static char  *logout = "LOGOUT";

/* The possible consumer names of queues: */
static char *applist[] = {"APP1", "APP2","APP3"};

static OCIEnv *envhp;
static OCIServer *srvhp;
static OCIError *errhp;
static OCISvcCtx *svchp;

static void checkerr(/*_ OCIError *errhp, sword status _*/);

struct process_statistics
{
  ub4  login;
  ub4  logout;
};
```

```
typedef struct process_statistics process_statistics;

int main(/*_ int argc, char *argv[] _*/);

/* Notify Callback: */
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *pay;
ub4    payl;
dvoid *desc;
ub4    mode;
{
 text                *subname;   /* subscription name */
 ub4                  lsub;      /* length of subscription name */
 text                *queue;     /* queue name */
 ub4                 *lqueue;    /* queue name */
 text                *consumer;  /* consumer name */
 ub4                  lconsumer;
 text                *correlation;
 ub4                  lcorrelation;
 ub4                  size;
 ub4                  appno;
 OCIRaw              *msgid;
 OCIAQMsgProperties  *msgprop;   /* message properties descriptor */
 process_statistics   *user_count = (process_statistics *)ctx;

 OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
           (dvoid *)&subname, &lsub, OCI_ATTR_SUBSCR_NAME, errhp);

 /* Extract the attributes from the AQ descriptor: */
 /* Queue name: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
            OCI_ATTR_QUEUE_NAME, errhp);

 /* Consumer name: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &lconsumer,
            OCI_ATTR_CONSUMER_NAME, errhp);

 /* Message properties: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
            OCI_ATTR_MSG_PROP, errhp);

 /* Get correlation from message properties: */
  checkerr(errhp, OCIAttrGet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
```

```
          (dvoid *)&correlation, &lcorrelation, OCI_ATTR_CORRELATION, errhp));

if (lconsumer == strlen(applist[0]))
{
  if (!memcmp((dvoid *)consumer, (dvoid *)applist[0], strlen(applist[0])))
   appno = 0;
  else if (!memcmp((dvoid *)consumer, (dvoid *)applist[1],strlen(applist[1])))
   appno = 1;
  else if (!memcmp((dvoid *)consumer, (dvoid *)applist[2],strlen(applist[2])))
   appno = 2;
  else
  {
   printf("Wrong consumer in notification");
   return;
  }
}
else
{  /* consumer name must be "APP1", "APP2" or "APP3"  */
  printf("Wrong consumer in notification");
  return;
}

if (lcorrelation == strlen(login) &&                   /* login event */
     !memcmp((dvoid *)correlation, (dvoid *)login, strlen(login)))
{
   user_count[appno].login++;
                      /* increment login count for the app process */
   printf("Login by APP%d \n", (appno+1));
   printf("Login Payload length = %d \n", pay1);
 }
else if  (lcorrelation == strlen(logout) &&          /* logout event */
     !memcmp((dvoid *)correlation,(dvoid *)logout, strlen(logout)))
{
   user_count[appno].logout++;
                     /* increment logout count for the app process */
   printf("logout by APP%d \n", (appno+1));
   printf("logout Payload length = %d \n", pay1);
}
else                            /* correlation is "LOGIN" or "LOGOUT" */
   printf("Wrong correlation in notification");

printf("Total  : \n");

printf("App1 : %d \n", user_count[0].login-user_count[0].logout);
printf("App2 : %d \n", user_count[1].login-user_count[1].logout);
```

```
  printf("App3 : %d \n", user_count[2].login-user_count[2].logout);

}

int main(argc, argv)
int argc;
char *argv[];
{
  OCISession *authp = (OCISession *) 0;
  OCISubscription *subscrhp[3];
  ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;
  process_statistics  ctx[3] = {{0,0}, {0,0}, {0,0}};
  ub4 sleep_time = 0;

  printf("Initializing OCI Process\n");

  /* Initialize OCI environment with OCI_EVENTS flag set: */
  (void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                      (dvoid * (*)(dvoid *, size_t)) 0,
                      (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                      (void (*)(dvoid *, dvoid *)) 0 );

  printf("Initialization successful\n");

  printf("Initializing OCI Env\n");
  (void) OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0, (dvoid **) 0
);
  printf("Initialization successful\n");

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp,
    OCI_HTYPE_ERROR, (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp,
    OCI_HTYPE_SERVER, (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp,
    OCI_HTYPE_SVCCTX, (size_t) 0, (dvoid **) 0));

  printf("connecting to server\n");
  checkerr(errhp, OCIServerAttach( srvhp, errhp, (text *)"inst1_alias",
          strlen("inst1_alias"), (ub4) OCI_DEFAULT));
  printf("connect successful\n");

  /* Set attribute server context in the service context: */
  checkerr(errhp, OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
```

```
                (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp));

checkerr(errhp, OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
                (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0));

/* Set username and password in the session handle: */
checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) username, (ub4) strlen((char *)username),
                (ub4) OCI_ATTR_USERNAME, errhp));

checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
               (dvoid *) password, (ub4) strlen((char *)password),
               (ub4) OCI_ATTR_PASSWORD, errhp));

/* Begin session: */
checkerr(errhp, OCISessionBegin ( svchp,  errhp, authp, OCI_CRED_RDBMS,
               (ub4) OCI_DEFAULT));

(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
               (dvoid *) authp, (ub4) 0,
               (ub4) OCI_ATTR_SESSION, errhp);

/* Register for notification: */
 printf("allocating subscription handle\n");
subscrhp[0] = (OCISubscription *)0;
(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
               (ub4) OCI_HTYPE_SUBSCRIPTION,
               (size_t) 0, (dvoid **) 0);

/* For application process APP1: */
printf("setting subscription name\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *) "OE.LOGIN_LOGOUT:APP1",
               (ub4) strlen("OE.LOGIN_LOGOUT:APP1"),
               (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

printf("setting subscription callback\n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *) notifyCB, (ub4) 0,
               (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *)&ctx, (ub4)sizeof(ctx),
               (ub4) OCI_ATTR_SUBSCR_CTX, errhp);
```

```
 printf("setting subscription namespace\n");
 (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) &namespace, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

 printf("allocating subscription handle\n");
 subscrhp[1] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[1],
                (ub4) OCI_HTYPE_SUBSCRIPTION,
                (size_t) 0, (dvoid **) 0);

 /* For application process APP2: */
 printf("setting subscription name\n");
 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) "OE.LOGIN_LOGOUT:APP2",
                (ub4) strlen("OE.LOGIN_LOGOUT:APP2"),
                (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

 printf("setting subscription callback\n");
 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) notifyCB, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *)&ctx, (ub4)sizeof(ctx),
                (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

 printf("setting subscription namespace\n");
 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) &namespace, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

 printf("allocating subscription handle\n");
 subscrhp[2] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[2],
                (ub4) OCI_HTYPE_SUBSCRIPTION,
                (size_t) 0, (dvoid **) 0);

 /* For application process APP3: */
 printf("setting subscription name\n");
 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) "OE.LOGIN_LOGOUT:APP3",
                (ub4) strlen("OE.LOGIN_LOGOUT:APP3"),
                (ub4) OCI_ATTR_SUBSCR_NAME, errhp);
```

```
  printf("setting subscription callback\n");
  (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) notifyCB, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *)&ctx, (ub4)sizeof(ctx),
                (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

  printf("setting subscription namespace\n");
  (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) &namespace, (ub4) 0,
                (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

  printf("Registering fornotifications \n");
  checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 3, errhp,
                OCI_DEFAULT));

  sleep_time = (ub4)atoi(argv[1]);
  printf ("waiting for %d s \n", sleep_time);
  sleep(sleep_time);

  printf("Exiting");
  exit(0);
}

void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  sb4 errcode = 0;

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
    (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    (void) printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    (void) printf("Error - OCI_NODATA\n");
```

```
    break;
  case OCI_ERROR:
    (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                        errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
    (void) printf("Error - %.*s\n", 512, errbuf);
    break;
  case OCI_INVALID_HANDLE:
    (void) printf("Error - OCI_INVALID_HANDLE\n");
    break;
  case OCI_STILL_EXECUTING:
    (void) printf("Error - OCI_STILL_EXECUTE\n");
    break;
  case OCI_CONTINUE:
    (void) printf("Error - OCI_CONTINUE\n");
    break;
  default:
    break;
  }
}

/* End of file tkaqdocn.c */
```

Visual Basic (OO4O): Example Code

This feature is not supported currently.

Java (JDBC): Example Code

This feature is not supported through the Java API.

Retention and Message History

Oracle Streams AQ allows the retention of the message history after consumption. The messages and their histories can be queried using SQL. This allows business analysis of the integrated system. In certain cases, messages must be tracked. 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. Taken together, retention, message identifiers, and SQL queries make it possible to build powerful message warehouses.

Scenario

Assume that you must determine the average order processing time. This includes the time the order must wait in the `back_order` queue. You want to know the average wait time in the `backed_order` queue. SQL queries can determine the wait time for orders in the shipping application. Specify the retention as `TRUE` for the shipping queues and specify the order number in the correlation field of the message.

For simplicity, only orders that have already been processed are analyzed. The processing time for an order in the shipping application is the difference between the enqueue time in the `WS_bookedorders_que` and the enqueue time in the `WS_shipped_orders_que`.

> **See Also:** "tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues, and Subscribers" on page A-2

PL/SQL (DBMS_AQADM Package): Example Code

```
SELECT  SUM(SO.enq_time - BO.enq_time) / count (*) AVG_PRCS_TIME
   FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
   WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
   AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_que';

/* Average waiting time in the backed order queue: */
SELECT SUM(BACK.deq_time - BACK.enq_time)/count (*) AVG_BACK_TIME
   FROM WS.AQ$WS_orders_mqtab BACK
   WHERE BACK.msg_state = 'PROCESSED' AND BACK.queue = 'WS_backorders_que';
```

Visual Basic (OO4O): Example Code

Use the dbexecutesql interface from the database for this functionality.

Java (JDBC): Example Code

No example is provided with this release.

Publish/Subscribe Support

Oracle Streams AQ supports the **publish/subscribe** model of application integration. In the model, publishing applications put the message in the queue. The subscribing applications subscribe to the message in the queue. More publishing and subscribing applications can be dynamically added without changing the existing publishing and subscribing applications.

Oracle Streams AQ also supports content-based subscriptions. The subscriber can subscribe to a subset of messages in the queue based on the message properties and the contents of the messages. A subscriber to a queue can also be another queue or a consumer on another queue.

You can implement a publish/subscribe model of communication using Oracle Streams AQ as follows:

- 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.
- 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.
- Publisher applications publish messages to the queue by invoking an enqueue call.
- Subscriber applications can receive messages in the following manner:
  - A dequeue call retrieves messages that match the subscription criteria.
  - A listen call can be used 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.
  - 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.

Scenario

The BooksOnLine application illustrates the use of a publish/subscribe model for communicating between applications. The following subsections give some examples.

Defining queues

The Order Entry application defines a queue (`OE_booked_orders_que`) to communicate orders that are booked to various applications. The Order Entry application is not aware of the various subscriber applications and thus, a new subscriber application can be added without disrupting any setup or logic in the Order Entry (publisher) application.

Setting Up Subscriptions

The various Shipping applications and the Customer Service application (that is, Eastern Region shipping, Western Region shipping, Overseas Shipping and Customer Service) are defined as subscribers to the `booked_orders` queue of the Order Entry application. Oracle Streams AQ uses **rules** to route messages of interest to the various subscribers. Thus, Eastern Region shipping, which handles shipment of all orders for the East Coast and all rush U.S. orders, expresses the subscription rule as follows:

```
rule  => 'tab.user_data.orderregion = ''EASTERN'' OR
(tab.user_data.ordertype = ''RUSH'' AND
tab.user_data.customer.country = ''USA'') '
```

Each subscriber can specify a local queue where messages are to be delivered. The Eastern Region shipping application specifies a local queue (`ES_booked_orders_que`) for message delivery by specifying the subscriber address as follows:

```
subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
```

Setting Up Propagation

Enable propagation from each publisher application queue. To allow subscribed messages to be delivered to remote queues, the Order Entry application enables propagation by means of the following statement:

```
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'OE.OE_bookedorders_que');
Publishing Messages
```

Booked orders are published by the Order Entry application when it enqueues orders (into the `OE_booked_order_que`) that have been validated and are ready for shipping. These messages are then routed to each of the subscribing applications. Messages are delivered to local queues (if specified) at each of the subscriber applications.

Receiving Messages

Each of the shipping applications and the Customer Service application then receives these messages in their local queues. For example, Eastern Region Shipping only receives booked orders that are for East Coast addresses or any U.S. order that are marked `RUSH`. This application then dequeues messages and processes its orders for shipping.

Oracle Real Application Clusters Support

Real Application Clusters can be used to improve Oracle Streams AQ 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.

The Oracle Streams AQ 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 Streams AQ 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 those 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.

See Also:

- "Scheduling a Queue Propagation" on page 8-32
- *Oracle Real Application Clusters Installation and Configuration Guide*

Scenario

In the BooksOnLine example, operations on the `new_orders_queue` and `booked_order_queue` at the order entry (OE) site can be made faster if the two queues are associated with different instances. This is accomplished by creating the queues in different queue tables and specifying different affinities for the queue tables in the `create_queue_table()` command.

In the example, the queue table `OE_orders_sqtab` stores queue `new_orders_queue` and the primary and secondary are instances 1 and 2 respectively. Queue table `OE_orders_mqtab` stores queue `booked_order_queue` and the primary and secondary are instances 2 and 1 respectively.

The objective is to let instances 1 and 2 manage the two queues in parallel. By default, only one instance is available, in which case the owner instances of both queue tables are set to instance 1. However, if Real Application Clusters are set up correctly and both instances 1 and 2 are available, then queue table `OE_orders_sqtab` is owned by instance 1 and the other queue table is owned by instance 2.

The primary and secondary instance specification of a queue table can be changed dynamically using the `alter_queue_table()` command as shown in the following example. Information about the primary, secondary and owner instance of a queue table can be obtained by querying the view `USER_QUEUE_TABLES`.

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`.

See Also: "Queue Tables in User Schema View" on page 9-12

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Create queue tables, queues for OE  */
CONNECT OE/OE;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
        queue_table        => 'OE_orders_sqtab',\
        comment            => 'Order Entry Single-Consumer Orders queue table',\
        queue_payload_type => 'BOLADM.order_typ',\
        compatible         => '8.1',\
        primary_instance   => 1,\
        secondary_instance => 2);

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(\
        queue_table        => 'OE_orders_mqtab',\
        comment            => 'Order Entry Multi Consumer Orders queue table',\
        multiple_consumers => TRUE,\
        queue_payload_type => 'BOLADM.order_typ',\
        compatible         => '8.1',\
        primary_instance   => 2,\
        secondary_instance => 1);

EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
        queue_name         => 'OE_neworders_que',\
        queue_table        => 'OE_orders_sqtab');

EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
```

```
        queue_name         => 'OE_bookedorders_que',\
        queue_table        => 'OE_orders_mqtab');

/* Check instance affinity of OE queue tables from AQ administrative view: */
SELECT queue_table, primary_instance, secondary_instance, owner_instance
FROM user_queue_tables;

/* Alter instance affinity of OE queue tables: */
EXECUTE DBMS_AQADM.ALTER_QUEUE_TABLE( \
        queue_table        => 'OE.OE_orders_sqtab',\
        primary_instance   => 2,\
        secondary_instance => 1);

EXECUTE DBMS_AQADM.ALTER_QUEUE_TABLE(  \
        queue_table        => 'OE.OE_orders_mqtab', \
        primary_instance   => 1,\
        secondary_instance => 2);

/* Check instance affinity of OE queue tables from AQ administrative view: */
SELECT queue_table, primary_instance, secondary_instance, owner_instance
FROM user_queue_tables;
```

Visual Basic (OO4O): Example Code

This feature currently not supported.

Java (JDBC): Example Code

```
public static void createQueueTablesAndQueues(Connection db_conn)
{
    AQSession            aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTableProperty mqt_prop;
    AQQueueTable         sq_table;
    AQQueueTable         mq_table;
    AQQueueProperty      q_prop;
    AQQueue              neworders_q;
    AQQueue              bookedorders_q;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* Create a single-consumerorders queue table */
        sqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
```

```
        sqt_prop.setComment("Order Entry Single-Consumer Orders queue table");
        sqt_prop.setCompatible("8.1");
        sqt_prop.setPrimaryInstance(1);
        sqt_prop.setSecondaryInstance(2);

        sq_table = aq_sess.createQueueTable("OE", "OE_orders_sqtab", sqt_prop);

        /* Create a multiconsumer orders queue table */
        mqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        mqt_prop.setComment("Order Entry Multiconsumer Orders queue table");
        mqt_prop.setCompatible("8.1");
        mqt_prop.setMultiConsumer(true);
        mqt_prop.setPrimaryInstance(2);
        mqt_prop.setSecondaryInstance(1);

        mq_table = aq_sess.createQueueTable("OE", "OE_orders_mqtab", mqt_prop);


        /* Create queues in these queue tables */
        q_prop = new AQQueueProperty();

        neworders_q = aq_sess.createQueue(sq_table, "OE_neworders_que",
                                          q_prop);

        bookedorders_q = aq_sess.createQueue(mq_table, "OE_bookedorders_que",
                                             q_prop);

    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }
}

public static void alterInstanceAffinity(Connection db_conn)
{
    AQSession            aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTableProperty mqt_prop;
    AQQueueTable         sq_table;
    AQQueueTable         mq_table;
    AQQueueProperty      q_prop;

    try
    {
```

```
            /* Create an AQ session: */
            aq_sess = AQDriverManager.createAQSession(db_conn);

            /* Check instance affinities */
            sq_table = aq_sess.getQueueTable("OE", "OE_orders_sqtab");

            sqt_prop = sq_table.getProperty();
            System.out.println("Current primary instance for OE_orders_sqtab: " +
                               sqt_prop.getPrimaryInstance());

            mq_table = aq_sess.getQueueTable("OE", "OE_orders_mqtab");
            mqt_prop = mq_table.getProperty();
            System.out.println("Current primary instance for OE_orders_mqtab: " +
                               mqt_prop.getPrimaryInstance());

            /* Alter queue table affinities */
            sq_table.alter(null, 2, 1);

            mq_table.alter(null, 1, 2);

            sqt_prop = sq_table.getProperty();
            System.out.println("Current primary instance for OE_orders_sqtab: " +
                               sqt_prop.getPrimaryInstance());

            mq_table = aq_sess.getQueueTable("OE", "OE_orders_mqtab");
            mqt_prop = mq_table.getProperty();
            System.out.println("Current primary instance for OE_orders_mqtab: " +
                               mqt_prop.getPrimaryInstance());

        }
        catch (AQException ex)
        {
            System.out.println("AQ Exception: " + ex);
        }
    }
```

Statistics Views and Oracle Streams AQ

Each instance keeps its own Oracle Streams AQ 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 Streams AQ statistics information to the instance issuing the query.

Scenario

The gv$ 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. The order processing application can use this to dynamically tune the number of order processing processes.

See Also: "Number of Messages in Different States for the Whole Database View" on page 9-17

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT oe/oe

/* Count the number of messages and the average time for which the messages have
   been waiting: */
SELECT READY, AVERAGE_WAIT FROM gv$aq Stats, user_queues Qs
  WHERE Stats.qid = Qs.qid and Qs.Name = 'OE_neworders_que';
```

Visual Basic (OO4O): Example Code

Use the dbexecutesql interface from the database for this functionality.

Java (JDBC): Example Code

No example is provided with this release.

Internet Access for Oracle Streams AQ

See Chapter 17, "Internet Access to Oracle Streams AQ" for information on Internet access to Oracle Streams AQ features.

Enqueue Features

This section contains these topics:

- Subscriptions and Recipient Lists
- Priority and Ordering of Messages
- Time Specification: Delay
- Time Specification: Expiration
- Message Grouping
- Retry with Delay Interval
- Message Transformation During Enqueue
- Enqueue Using the Oracle Streams AQ XML Servlet

Subscriptions and Recipient Lists

After consumption by dequeue, messages are retained for the time specified in `retention_time`. When `retention_time` expires, messages are removed by the time manager process.

After processing, the message is removed if the `retention_time` of the queue is 0, or retained for the specified retention time. While the message is retained the message can either be queried using SQL on the queue table view or by dequeuing using the `BROWSE` mode and specifying the message ID of the processed message.

Oracle Streams AQ allows a single message to be processed and consumed by more than one consumer. To use this feature, you must create multiconsumer queues and enqueue the messages into these multiconsumer queues. Oracle Streams AQ allows two methods of identifying the list of consumers for a message: subscriptions and **recipient** lists.

Subscriptions

You can add a subscription to a queue by using the `DBMS_AQADM.ADD_SUBSCRIBER` PL/SQL procedure. This lets you specify a consumer by means of the `AQ$_AGENT` parameter for enqueued messages. You can add more subscribers by repeatedly using the `DBMS_AQADM.ADD_SUBSCRIBER` procedure up to a maximum of 1024 subscribers for a multiconsumer queue.

See Also: "Adding a Subscriber" on page 8-26

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

> **See Also:** "AQ Agent Type (aq$_agent)" on page 3-3 for a formal description of this data structure

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 DBMS_AQADM.ADD_SUBSCRIBER procedure is completed. In other words, messages that had been enqueued before this procedure is executed are not available for dequeue by this consumer.

You can remove a subscription by using the DBMS_AQADM.REMOVE_SUBSCRIBER procedure. Oracle Streams AQ automatically removes from the queue all data corresponding to the consumer identified by the AQ$_AGENT parameter. In other words, 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 after the REMOVE_SUBSCRIBER procedure is executed.

> **See Also:** "Removing a Subscriber" on page 8-30

In a queue table that is created with the compatible parameter set to '8.1' or higher, such messages that were not dequeued by the consumer are shown as "UNDELIVERABLE" in the AQ$*queue_table* view. 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.

Recipient Lists

You are not required to specify subscriptions for a multiconsumer queue if the producers of messages for enqueue supply a recipient list of consumers. In some situations it can be desirable to enqueue a message that is targeted to a specific set of consumers rather than the default list of subscribers. You accomplish this by specifying a recipient list at the time of enqueuing the message.

- In PL/SQL you specify the recipient list by adding elements to the recipient_list field of the message_properties record.

- In OCI the recipient list is specified by using the `OCISetAttr` procedure to specify an array of `OCI_DTYPE_AQAGENT` descriptors as the recipient list (`OCI_ATTR_RECIPIENT_LIST` attribute) of an `OCI_DTYPE_AQMSG_PROPERTIES` message properties descriptor.

If a recipient list is specified during enqueue, then it overrides the subscription list. In other words, messages that have a specified recipient list are not available for dequeue by the subscribers of the queue. The consumers specified in the recipient list may or may not be subscribers for the queue. It is an error if the queue does not have any subscribers and the enqueue does not specify a recipient list.

See Also: "Enqueuing a Message" on page 10-2

Priority and Ordering of Messages

The message ordering dictates the order that messages are dequeued from a queue. The ordering method for a queue is specified when a queue table is created.

See Also: "Creating a Queue Table" on page 8-2

Priority ordering of messages is achieved by specifying priority, enqueue time as the sort order for the message. If priority ordering is chosen, then each message is assigned a priority at enqueue time by the enqueuer. 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 the enqueue time, priority as the sort order of the messages.

Scenario

In the BooksOnLine application, a customer can request:

- FedEx shipping (priority 1)
- Priority air shipping (priority 2)
- Regular ground shipping (priority 3)

The Order Entry application uses a priority queue to store booked orders. Booked orders are propagated to the regional booked orders queues. At each region, orders in these regional booked orders queues are processed in the order of the shipping priorities.

The following calls create the priority queues for the Order Entry application.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Create a priority queue table for OE: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
   queue_table         => 'OE_orders_pr_mqtab', \
   sort_list           =>'priority,enq_time', \
   comment             => 'Order Entry Priority  \
                          MultiConsumer Orders queue table',\
   multiple_consumers  => TRUE, \
   queue_payload_type  => 'BOLADM.order_typ', \
   compatible          => '8.1', \
   primary_instance    => 2, \
   secondary_instance  => 1);

EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
   queue_name          => 'OE_bookedorders_que', \
   queue_table         => 'OE_orders_pr_mqtab');

/* When an order arrives, the order entry application can use the following
   procedure to enqueue the order into its booked orders queue. A shipping
   priority is specified for each order: */
CREATE OR REPLACE procedure order_enq(book_title        IN VARCHAR2,
                                      book_qty          IN NUMBER,
                                      order_num         IN NUMBER,
                                      shipping_priority IN NUMBER,
                                      cust_state        IN VARCHAR2,
                                      cust_country      IN VARCHAR2,
                                      cust_region       IN VARCHAR2,
                                      cust_ord_typ      IN VARCHAR2) AS

OE_enq_order_data        BOLADM.order_typ;
OE_enq_cust_data         BOLADM.customer_typ;
OE_enq_book_data         BOLADM.book_typ;
OE_enq_item_data         BOLADM.orderitem_typ;
OE_enq_item_list         BOLADM.orderitemlist_vartyp;
enqopt                   dbms_aq.enqueue_options_t;
msgprop                  dbms_aq.message_properties_t;
enq_msgid                RAW(16);

BEGIN
   msgprop.correlation := cust_ord_typ;
   OE_enq_cust_data    := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
                                cust_state, NULL, cust_country);
   OE_enq_book_data    := BOLADM.book_typ(book_title, NULL, NULL, NULL);
   OE_enq_item_data    := BOLADM.orderitem_typ(book_qty,
```

```
                              OE_enq_book_data, NULL);
   OE_enq_item_list    := BOLADM.orderitemlist_vartyp(
                              BOLADM.orderitem_typ(book_qty,
                              OE_enq_book_data, NULL));
   OE_enq_order_data   := BOLADM.order_typ(order_num, NULL,
                              cust_ord_typ, cust_region,
                              OE_enq_cust_data, NULL,
                              OE_enq_item_list, NULL);

   /*Put the shipping priority into message property before enqueuing
     the message: */
   msgprop.priority    := shipping_priority;
   dbms_aq.enqueue('OE.OE_bookedorders_que', enqopt, msgprop,
                      OE_enq_order_data, enq_msgid);
        COMMIT;
  END;
  /


/* At each region, similar booked order queues are created. The orders are
   propagated from the central Order Entry's booked order queues to the regional
   booked order queues. For example, at the Western Region, the booked orders
   queue is created. Create a priority queue table for WS shipping: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
   queue_table        =>  'WS_orders_pr_mqtab',
   sort_list          =>'  priority,enq_time',  \
   comment            =>  'West Shipping Priority  \
                           MultiConsumer Orders queue table',\
   multiple_consumers => TRUE, \
   queue_payload_type => 'BOLADM.order_typ', \
   compatible         => '8.1');

/* Booked orders are stored in the priority queue table: */
EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
   queue_name         => 'WS_bookedorders_que', \
   queue_table        => 'WS_orders_pr_mqtab');

/* At each region, the shipping application dequeues orders from the regional
   booked order queue according to the orders' shipping priorities, processes
   the orders, and enqueues the processed orders into the shipped orders queues
   or the backorders queues. */
```

Visual Basic (OO4O): Example Code

```
Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraCust,OraBook,OraItem,OraItemList as Object
Dim Msgid as String

   Set OraSession = CreateObject("OracleInProcServer.XOraSession")
   Set OraDatabase = OraSession.DbOpenDatabase("dbname", "user/pwd", 0&)
   set oraAq = OraDatabase.CreateAQ("OE.OE_bookedorders_que")
   Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
   Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")
   Set OraCust = OraDatabase.CreateOraObject("BOLADM.Customer_typ")
   Set OraBook = OraDatabase.CreateOraObject("BOLADM.book_typ")
   Set OraItem = OraDatabase.CreateOraObject("BOLADM.orderitem_typ")
   Set OraItemList = OraDatabase.CreateOraObject("BOLADM.orderitemlist_vartyp")

   ' Get the values of cust_state,cust_country etc from user(form_based
   ' input) and then a cmd_click event for Enqueue
   ' will run the subroutine order_enq.
   Private Sub Order_enq()

   OraMsg.correlation = txt_correlation
   'Initialize the customer details
        OraCust("state") = txt_cust_state
   OraCust("country") = txt_cust_country
        OraBook("title") = txt_book_title
   OraItem("quantity") = txt_book_qty
   OraItem("item") = OraBook
   OraItemList(1) = OraItem
   OraOrder("orderno") = txt_order_num
   OraOrder("ordertype") = txt_cust_order_typ
   OraOrder("orderregion") = cust_region
   OraOrder("customer") = OraCust
   OraOrder("items") = OraItemList

   'Put the shipping priority into message property before enqueuing
   '  the message:
   OraMsg.priority = priority
   OraMsg = OraOrder
   Msgid = OraAq.enqueue

   'Release all allocations
```

```
End Sub
```

Java (JDBC): Example Code

```
public static void createPriorityQueueTable(Connection db_conn)
{
    AQSession            aq_sess;
    AQQueueTableProperty mqt_prop;
    AQQueueTable         pr_mq_table;
    AQQueueProperty      q_prop;
    AQQueue              bookedorders_q;

    try
    {

        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* Create a priority queue table for OE */
        mqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        mqt_prop.setComment("Order Entry Priority " +
                            "MultiConsumer Orders queue table");
        mqt_prop.setCompatible("8.1");
        mqt_prop.setMultiConsumer(true);

        mqt_prop.setSortOrder("priority,enq_time");

        pr_mq_table = aq_sess.createQueueTable("OE", "OE_orders_pr_mqtab",
                                            mqt_prop);

        /* Create a queue in this queue table */
        q_prop = new AQQueueProperty();

        bookedorders_q = aq_sess.createQueue(pr_mq_table,
                                             "OE_bookedorders_que", q_prop);

        /* Enable enqueue and dequeue on the queue */
        bookedorders_q.start(true, true);

    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }
}
```

```
/* When an order arrives, the order entry application can use the following
   procedure to enqueue the order into its booked orders queue. A shipping
   priority is specified for each order.
 */
public static void order_enqueue(Connection db_conn, String book_title,
                                 double book_qty, double order_num,
                                 int ship_priority, String cust_state,
                                 String cust_country, String cust_region,
                                 String cust_order_type)
{
    AQSession         aq_sess;
    AQQueue           bookedorders_q;
    Order             enq_order;
    Customer          cust_data;
    Book              book_data;
    OrderItem         item_data;
    OrderItem[]       items;
    OrderItemList     item_list;
    AQEnqueueOption   enq_option;
    AQMessageProperty m_property;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    byte[]            enq_msg_id;

    try
    {

        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        cust_data = new Customer();
        cust_data.setCountry(cust_country);
        cust_data.setState(cust_state);

        book_data = new Book();
        book_data.setTitle(book_title);

        item_data = new OrderItem();
        item_data.setQuantity(new BigDecimal(book_qty));
        item_data.setItem(book_data);

        items = new OrderItem[1];
        items[0] = item_data;
```

```
            item_list = new OrderItemList(items);

            enq_order = new Order();
            enq_order.setCustomer(cust_data);
            enq_order.setItems(item_list);
            enq_order.setOrderno(new BigDecimal(order_num));
            enq_order.setOrdertype(cust_order_type);

            bookedorders_q = aq_sess.getQueue("OE", "OE_bookedorders_que");

            message = bookedorders_q.createMessage();

            /* Put the shipping priority into message property before enqueuing */
            m_property = message.getMessageProperty();

            m_property.setPriority(ship_priority);

            obj_payload = message.getObjectPayload();

            obj_payload.setPayloadData(enq_order);

            enq_option = new AQEnqueueOption();

            /* Enqueue the message */
            enq_msg_id = bookedorders_q.enqueue(enq_option, message);

            db_conn.commit();

        }
        catch (AQException aq_ex)
        {
            System.out.println("AQ Exception: " + aq_ex);
        }
        catch (SQLException sql_ex)
        {
            System.out.println("SQL Exception: " + sql_ex);
        }

    }

/* At each region, similar booked order queues are created. The orders are
   propagated from the central Order Entry's booked order queues to the
   regional booked order queues.
   For example, at the Western Region, the booked orders queue is created.
```

```
   Create a priority queue table for WS shipping
 */
public static void createWesternShippingQueueTable(Connection db_conn)
{
    AQSession            aq_sess;
    AQQueueTableProperty mqt_prop;
    AQQueueTable         mq_table;
    AQQueueProperty      q_prop;
    AQQueue              bookedorders_q;

    try
    {

        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);


        /* Create a priority queue table for WS: */
        mqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        mqt_prop.setComment("Western Shipping Priority " +
                            "MultiConsumer Orders queue table");
        mqt_prop.setCompatible("8.1");
        mqt_prop.setMultiConsumer(true);
        mqt_prop.setSortOrder("priority,enq_time");

        mq_table = aq_sess.createQueueTable("WS", "WS_orders_pr_mqtab",
                                            mqt_prop);

        /* Booked orders are stored in the priority queue table: */
        q_prop = new AQQueueProperty();

        bookedorders_q = aq_sess.createQueue(mq_table, "WS_bookedorders_que",
                                             q_prop);

        /* Start the queue:*/
        bookedorders_q.start(true, true);

    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }

  /* At each region, the shipping application dequeues orders from the
     regional booked order queue according to the orders' shipping priorities,
```

```
    processes the orders, and enqueues the processed orders into the shipped
    orders queues or the backorders queues.
  */
}
```

Time Specification: Delay

Oracle Streams AQ supports delay delivery of messages by letting the enqueuer specify a delay interval on a message when enqueuing the message, that is, the time before that a message cannot be retrieved by a dequeue call. The delay interval determines when an enqueued message is marked as available to the dequeuers after the message is enqueued.

> **See Also:** "Enqueuing a Message and Specifying Options" on page 10-3

When a message is enqueued with a delay time set, the message is marked in a `WAIT` state. Messages in `WAIT` state are masked from the default dequeue calls. A background time-manager daemon wakes up periodically, scans an internal index for all `WAIT` state messages, and marks messages as `READY` if their delay time has passed. The time-manager then posts to all foreground processes that are waiting on queues for messages that have just been made available.

Scenario

In the BooksOnLine application, delay can be used to implement deferred billing. A billing application can define a queue where shipped orders that are not billed immediately can be placed in a deferred billing queue with a delay. For example, a certain class of customer accounts, such as those of corporate customers, may not be billed for 15 days. The billing application dequeues incoming shipped order messages (from the `shippedorders` queue) and if the order is for a corporate customer, this order is enqueued into a deferred billing queue with a delay.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Enqueue an order to implement deferred billing so that the order is not made
   visible again until delay has expired: */
CREATE OR REPLACE PROCEDURE defer_billing(deferred_billing_order order_typ)
AS
  defer_bill_queue_name    VARCHAR2(62);
  enqopt                   dbms_aq.enqueue_options_t;
  msgprop                  dbms_aq.message_properties_t;
  enq_msgid                RAW(16);
```

```
BEGIN

/* Enqueue the order into the deferred billing queue with a delay of 15 days: */
  defer_bill_queue_name := 'CBADM.deferbilling_que';
  msgprop.delay := 15*60*60*24;
  dbms_aq.enqueue(defer_bill_queue_name, enqopt, msgprop,
                  deferred_billing_order, enq_msgid);
END;
/
```

Visual Basic (OO4O): Example Code

```
set oraAq = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub defer_billing

OraMsg = OraOrder
OraMsg.delay = 15*60*60*24
OraMsg = OraOrder 'OraOrder contains the order details
Msgid = OraAq.enqueue

End Sub
```

Java (JDBC): Example Code

```
public static void defer_billing(Connection db_conn, Order deferred_order)
{
    AQSession         aq_sess;
    AQQueue           def_bill_q;
    AQEnqueueOption   enq_option;
    AQMessageProperty m_property;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    byte[]            enq_msg_id;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        def_bill_q = aq_sess.getQueue("CBADM", "deferbilling_que");

        message = def_bill_q.createMessage();
```

```
        /* Enqueue the order into the deferred billing queue with a delay
           of 15 days */
        m_property = message.getMessageProperty();
        m_property.setDelay(15*60*60*24);

        obj_payload = message.getObjectPayload();
        obj_payload.setPayloadData(deferred_order);

        enq_option = new AQEnqueueOption();

        /* Enqueue the message */
        enq_msg_id = def_bill_q.enqueue(enq_option, message);

        db_conn.commit();
    }
    catch (Exception ex)
    {
        System.out.println("Exception " + ex);
    }

}
```

Time Specification: Expiration

Messages can be enqueued with an expiration that specifies the interval of time the message is available for dequeuing. Expiration processing requires that the queue monitor be running. The **producer** can also specify the time when a message expires, at which time the message is moved to an **exception queue**.

Scenario

In the BooksOnLine application, expiration can be used to control the amount of time that is allowed to process a back order. The shipping application places orders for books that are not available in a back order queue. If the shipping policy is that all back_order must be shipped within a week, then messages can be enqueued into the back order queue with an expiration of 1 week. In this case, any back_order that are not processed within one week are moved to the exception queue with the message state set to `EXPIRED`. This can be used to flag any orders that have not been shipped according to the back order shipping policy.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT BOLADM/BOLADM
/* Re-enqueue a backorder into a backorder queue and set a delay of 7 days;
```

```
   all backorders must be processed in 7 days or they are moved to the
   exception queue: */
CREATE OR REPLACE PROCEDURE requeue_back_order(sale_region varchar2,
                                               backorder order_typ)
AS
  back_order_queue_name    VARCHAR2(62);
  enqopt                   dbms_aq.enqueue_options_t;
  msgprop                  dbms_aq.message_properties_t;
  enq_msgid                RAW(16);
BEGIN
  /* Look up a backorder queue based the the region by means of a directory
     service: */
  IF sale_region = 'WEST' THEN
    back_order_queue_name := 'WS.WS_backorders_que';
  ELSIF sale_region = 'EAST' THEN
    back_order_queue_name := 'ES.ES_backorders_que';
  ELSE
    back_order_queue_name := 'TS.TS_backorders_que';
  END IF;

  /* Enqueue the order with expiration set to 7 days: */
  msgprop.expiration := 7*60*60*24;
  dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
                  backorder, enq_msgid);
END;
/
```

Visual Basic (OO4O): Example Code

```
   set oraAq1 = OraDatabase.CreateAQ("WS.WS_backorders_que")
   set oraAq2 = OraDatabase.CreateAQ("ES.ES_backorders_que")
   set oraAq3 = OraDatabase.CreateAQ("CBADM.deferbilling_que")
   Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
   Set OraBackOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub Requeue_backorder
   Dim q as oraobject
   If sale_region = WEST then
      q = oraAq1
   else if sale_region = EAST then
      q = oraAq2
   else
      q = oraAq3
   end if
```

```
   OraMsg.delay = 7*60*60*24
   OraMsg = OraBackOrder 'OraOrder contains the order details
   Msgid = q.enqueue

End Sub
```

Java (JDBC): Example Code

```
/* Re-enqueue a backorder into a backorder queue and set a delay of 7 days;
   all backorders must be processed in 7 days or they are moved to the
   exception queue */
public static void requeue_back_order(Connection db_conn,
                                      String sale_region, Order back_order)
{
    AQSession         aq_sess;
    AQQueue           back_order_q;
    AQEnqueueOption   enq_option;
    AQMessageProperty m_property;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    byte[]            enq_msg_id;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* Look up a backorder queue based on the region */
        if(sale_region.equals("WEST"))
        {
            back_order_q = aq_sess.getQueue("WS", "WS_backorders_que");
        }
        else if(sale_region.equals("EAST"))
        {
            back_order_q = aq_sess.getQueue("ES", "ES_backorders_que");
        }
        else
        {
            back_order_q = aq_sess.getQueue("TS", "TS_backorders_que");
        }

        message = back_order_q.createMessage();
```

```
            m_property = message.getMessageProperty();

            /* Enqueue the order with expiration set to 7 days: */
            m_property.setExpiration(7*60*60*24);

            obj_payload = message.getObjectPayload();
            obj_payload.setPayloadData(back_order);

            enq_option = new AQEnqueueOption();

            /* Enqueue the message */
            enq_msg_id = back_order_q.enqueue(enq_option, message);

            db_conn.commit();
        }
        catch (Exception ex)
        {
            System.out.println("Exception :" + ex);
        }
    }
```

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 **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. With this feature, you can segment complex messages into simple messages.

See Also: "Creating a Queue Table" on page 8-2

For example, messages directed to a queue containing invoices can be constructed as a group of messages starting with the header message, followed by messages representing details, followed by the 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.

The general message properties (priority, delay, expiration) for the messages in a group are determined solely by the message properties specified for the first message (head) of the 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 must be 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.

See Also:

- "Dequeue Methods" on page 7-61
- "Modes of Dequeuing" on page 7-73

Scenario

In the BooksOnLine application, message grouping can be used to handle new orders. Each order contains a number of books ordered one by one in succession. Items ordered over the Web exhibit similar action.

In the following example, each enqueue corresponds to an individual book that is part of an order and the group/transaction represents a complete order. Only the first enqueue contains customer information. The `OE_neworders_que` is stored in the table `OE_orders_sqtab`, which has been enabled for transactional grouping. Refer to the example code for descriptions of procedures `new_order_enq`() and `same_order_enq`().

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`.

PL/SQL (DBMS_AQADM Package): Example Code

```
connect OE/OE;

/* Create queue table for OE:  */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
        queue_table        => 'OE_orders_sqtab',\
        comment            => 'Order Entry Single-Consumer Orders queue table',\
        queue_payload_type => 'BOLADM.order_typ',\
        message_grouping   => DBMS_AQADM.TRANSACTIONAL, \
        compatible         => '8.1',  \
        primary_instance   => 1,\
```

```
        secondary_instance => 2);

/* Create neworders queue for OE: */
EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
        queue_name         => 'OE_neworders_que',
        queue_table        => 'OE_orders_sqtab');

/* Login into OE account:*/
CONNECT OE/OE;
SET serveroutput on;
/* Enqueue some orders using message grouping into OE_neworders_que,
   First Order Group: */
EXECUTE BOLADM.new_order_enq('My First   Book', 1, 1001, 'CA');
EXECUTE BOLADM.same_order_enq('My Second  Book', 2);
COMMIT;
/
/* Second Order Group: */
EXECUTE BOLADM.new_order_enq('My Third   Book', 1, 1002, 'WA');
COMMIT;
/
/* Third Order Group: */
EXECUTE BOLADM.new_order_enq('My Fourth  Book', 1, 1003, 'NV');
EXECUTE BOLADM.same_order_enq('My Fifth   Book', 3);
EXECUTE BOLADM.same_order_enq('My Sixth   Book', 2);
COMMIT;
/
/* Fourth Order Group: */
EXECUTE BOLADM.new_order_enq('My Seventh Book', 1, 1004, 'MA');
EXECUTE BOLADM.same_order_enq('My Eighth  Book', 3);
EXECUTE BOLADM.same_order_enq('My Ninth   Book', 2);
COMMIT;
/
```

Visual Basic (OO4O): Example Code

This functionality is currently not available.

Java (JDBC): Example Code

```
public static void createMsgGroupQueueTable(Connection db_conn)
{
    AQSession            aq_sess;
    AQQueueTableProperty sqt_prop;
    AQQueueTable         sq_table;
    AQQueueProperty      q_prop;
```

```
    AQQueue             neworders_q;

    try
    {

        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        /* Create a single-consumer orders queue table */
        sqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        sqt_prop.setComment("Order Entry Single-Consumer Orders queue table");
        sqt_prop.setCompatible("8.1");
        sqt_prop.setMessageGrouping(AQQueueTableProperty.TRANSACTIONAL);

        sq_table = aq_sess.createQueueTable("OE", "OE_orders_sqtab", sqt_prop);

        /* Create new orders queue for OE */
        q_prop = new AQQueueProperty();

        neworders_q = aq_sess.createQueue(sq_table, "OE_neworders_que",
                                          q_prop);

    }
    catch (AQException ex)
    {
        System.out.println("AQ Exception: " + ex);
    }
}
```

Message Transformation During Enqueue

Continuing the scenario introduced in "Message Format Transformation" on page 7-7, the Order Entry and Shipping applications have different representations for the order item:

- The Order Entry application represents the order item in the form of the Oracle object type `OE.order_typ`.
- The Western Region Shipping application represents the order item in the form of the Oracle object type `WS.order_typ_sh`.

Therefore, the queues in the OE schema are of payload type `OE.orders_typ` and those in the WS schema are of payload type `WS.orders_typ_sh`.

Message transformation can be used during enqueue. This is especially useful for verification and transformation of messages during enqueue. An application can generate a message based on its own data model. The message can be transformed to the data type of the queue before it is enqueued using transformation mapping.

Scenario

At enqueue time, assume that instead of propagating messages from the `OE_booked_orders_topic`, an application dequeues the order, and, if it is meant for Western Region Shipping, publishes it to the `WS_booked_orders_topic`.

PL/SQL (DBMS_AQ Package): Example Code

The application can use transformations at enqueue time as follows:

```
CREATE OR REPLACE FUNCTION
  fwd_message_to_ws_shipping(booked_order  OE.order_typ)
   RETURNS boolean AS

 enq_opt   dbms_aq.enqueue_options_t;
 msg_prp   dbms_aq.message_properties_t;
BEGIN

 IF (booked_order.order_region = 'WESTERN' and
     booked_order.order_type != 'RUSH') THEN
   enq_opt.transformation := 'OE.OE2WS';
   msg_prp.recipient_list(0) := aq$_agent('West_shipping', null, null);

   dbms_aq.enqueue('WS.ws_bookedorders_topic',
                    enq_opt, msg_prp, booked_order);

   RETURN true;
 ELSE
    RETURN false;
 END IF;
END;
```

Visual Basic (OO4O): Example Code

No example is provided with this release.

Java (JDBC): Example Code

No example is provided with this release.

Enqueue Using the Oracle Streams AQ XML Servlet

You can perform enqueue requests over the Internet using **Internet Data Access Presentation** (IDAP).

> **See Also:** Chapter 17, "Internet Access to Oracle Streams AQ" for more information on sending Oracle Streams AQ requests using IDAP

Scenario

In the BooksOnLine application, a customer can request:

- FedEx shipping (priority 1),
- Priority air shipping (priority 2)
- Regular ground shipping (priority 3)

The Order Entry application uses a priority queue to store booked orders. Booked orders are propagated to the regional booked orders queues. At each region, orders in these regional booked orders queues are processed in the order of the shipping priorities.

The following calls create the priority queues for the Order Entry application.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Create a priority queue table for OE: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE( \
   queue_table          => 'OE_orders_pr_mqtab', \
   sort_list            =>'priority,enq_time', \
   comment              => 'Order Entry Priority  \
                           MultiConsumer Orders queue table',\
   multiple_consumers   => TRUE, \
   queue_payload_type   => 'BOLADM.order_typ', \
   compatible           => '8.1', \
   primary_instance     => 2, \
   secondary_instance   => 1);

EXECUTE DBMS_AQADM.CREATE_QUEUE ( \
   queue_name           => 'OE_bookedorders_que', \
   queue_table          => 'OE_orders_pr_mqtab');
```

Assume that a customer, John, wants to send an enqueue request using **Simple Object Access Protocol** (SOAP). The XML message has the following format:

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

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>
              <message_header>
                <correlation>ORDER1</correlation>
<priority>1</priority>
 <sender_id>
   <agent_name>john</agent_name>
         </sender_id>
              </message_header>

              <message_payload>

               <ORDER_TYP>
                     <ORDERNO>100</ORDERNO>
                     <STATUS>NEW</STATUS>
                     <ORDERTYPE>URGENT</ORDERTYPE>
                     <ORDERREGION>EAST</ORDERREGION>
                     <CUSTOMER>
                        <CUSTNO>1001233</CUSTNO>
                        <CUSTID>JOHN</CUSTID>
                        <NAME>JOHN DASH</NAME>
                        <STREET>100 EXPRESS STREET</STREET>
                        <CITY>REDWOOD CITY</CITY>
                        <STATE>CA</STATE>
                        <ZIP>94065</ZIP>
                        <COUNTRY>USA</COUNTRY>
                     </CUSTOMER>
                     <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
                     <ITEMS>
                        <ITEMS_ITEM>
                           <QUANTITY>10</QUANTITY>
```

```
                    <ITEM>
                       <TITLE>Perl handbook</TITLE>
                       <AUTHORS>Randal</AUTHORS>
                       <ISBN>345620200</ISBN>
                       <PRICE>19</PRICE>
                    </ITEM>
                    <SUBTOTAL>190</SUBTOTAL>
                 </ITEMS_ITEM>
                 <ITEMS_ITEM>
                    <QUANTITY>10</QUANTITY>
                    <ITEM>
                       <TITLE>JDBC guide</TITLE>
                       <AUTHORS>Taylor</AUTHORS>
                       <ISBN>123420212</ISBN>
                       <PRICE>59</PRICE>
                    </ITEM>
                    <SUBTOTAL>590</SUBTOTAL>
                 </ITEMS_ITEM>
              </ITEMS>
              <CCNUMBER>NUMBER01</CCNUMBER>
              <ORDER_DATE>08/23/2000 12:45:00</ORDER_DATE>
        </ORDER_TYP>
      </message_payload>
     </message>
   </message_set>

   <AQXmlCommit/>
  </AQXmlSend>
 </Body>
</Envelope>
```

Dequeue Features

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.

This section contains these topics:

- Dequeue Methods
- Multiple Recipients

- Local and Remote Recipients
- Message Navigation in Dequeue
- Modes of Dequeuing
- Optimization of Waiting for Arrival of Messages
- Retry with Delay Interval
- Exception Handling
- Rule-Based Subscription
- Listen Capability
- Message Transformation During Dequeue
- Dequeue Using the Oracle Streams AQ XML Servlet

Dequeue Methods

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

- Correlation identifier
- Message identifier
- Dequeue condition
- Default dequeue

A correlation identifier is a user-defined message property (of `VARCHAR2` datatype) while a message identifier is a system-assigned value (of `RAW` datatype). Multiple messages with the same correlation identifier can be present in a queue, while only one message with a given message identifier can be present. If there are multiple messages with the same correlation identifier, then the ordering (enqueue order) between messages may not be preserved on dequeue calls. The correlation identifier cannot be changed between successive dequeue calls without specifying the `FIRST_MESSAGE` navigation option.

A dequeue condition is an expression that is similar in syntax to the `WHERE` clause of a SQL query. Dequeue conditions are expressed in terms of the attributes that represent message properties or message content. The messages in the queue are evaluated against the conditions and a message that satisfies the given condition is returned.

A default dequeue means that the first available message for the consumer of a multiconsumer queue or the first available message in a single-consumer queue is dequeued.

Dequeuing with correlation identifier, message identifier, or dequeue condition does not preserve the message grouping property.

See Also:

- "Message Grouping" on page 7-53
- "Message Navigation in Dequeue" on page 7-68

Scenario

In the BooksOnLine example, rush orders received by the East shipping site are processed first. This is achieved by dequeuing the message using the correlation identifier, which has been defined to contain the order type (rush/normal). For an illustration of dequeuing using a message identifier, refer to the `get_northamerican_orders` procedure discussed in the example under "Modes of Dequeuing" on page 7-73.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT boladm/boladm;

/*  Create procedures to dequeue RUSH orders */
create or replace procedure get_rushtitles(consumer in varchar2) as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
qname                    varchar2(30);
no_messages              exception;
pragma exception_init    (no_messages, -25228);
new_orders               BOOLEAN := TRUE;

begin

        dopt.consumer_name := consumer;
        dopt.wait := 1;
        dopt.correlation := 'RUSH';
```

```
        IF (consumer = 'West_Shipping') THEN
                qname := 'WS.WS_bookedorders_que';
        ELSIF (consumer = 'East_Shipping') THEN
                qname := 'ES.ES_bookedorders_que';
        ELSE
                qname := 'TS.TS_bookedorders_que';
        END IF;

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);
            commit;

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;

            dbms_output.put_line(' rushorder book_title: ' ||
                                    deq_book_data.title ||
                        ' quantity: ' || deq_item_data.quantity);
          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE RUSH TITLES ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/

CONNECT EXECUTE on get_rushtitles to ES;

/* Dequeue the orders: */
CONNECT ES/ES;

/*  Dequeue all rush order titles for East_Shipping: */
EXECUTE BOLADM.get_rushtitles('East_Shipping');
```

Visual Basic (OO4O): Example Code

```
   set oraAq1 = OraDatabase.CreateAQ("WS.WS_backorders_que")
   set oraAq2 = OraDatabase.CreateAQ("ES.ES_backorders_que")
   set oraAq3 = OraDatabase.CreateAQ("CBADM.deferbilling_que")
   Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
   Set OraBackOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")

Private Sub Requeue_backorder
   Dim q as oraobject
   If sale_region = WEST then
      q = oraAq1
   else if sale_region = EAST then
      q = oraAq2
   else
      q = oraAq3
   end if

   OraMsg.delay = 7*60*60*24
   OraMsg = OraBackOrder 'OraOrder contains the order details
   Msgid = q.enqueue

End Sub
```

Java (JDBC): Example Code

```
public static void getRushTitles(Connection db_conn, String consumer)
{
    AQSession         aq_sess;
    Order             deq_order;
    byte[]            deq_msgid;
    AQDequeueOption   deq_option;
    AQMessageProperty msg_prop;
    AQQueue           bookedorders_q;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    boolean           new_orders = true;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        deq_option = new AQDequeueOption();

        deq_option.setConsumerName(consumer);
```

```
        deq_option.setWaitTime(1);
        deq_option.setCorrelation("RUSH");

        if(consumer.equals("West_Shipping"))
        {
            bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");
        }
        else if(consumer.equals("East_Shipping"))
        {
            bookedorders_q = aq_sess.getQueue("ES", "ES_bookedorders_que");
        }
        else
        {
            bookedorders_q = aq_sess.getQueue("TS", "TS_bookedorders_que");
        }

        while(new_orders)
        {
            try
            {
              /* Dequeue the message */
              message = bookedorders_q.dequeue(deq_option, Order.getFactory());

              obj_payload = message.getObjectPayload();

              deq_order = (Order)(obj_payload.getPayloadData());

              System.out.println("Order number " + deq_order.getOrderno() +
                                   " is a rush order");

            }
            catch (AQException aqex)
            {
              new_orders = false;
              System.out.println("No more rush titles");
              System.out.println("Exception-1: " + aqex);
            }
        }
    }
    catch (Exception ex)
    {
        System.out.println("Exception-2: " + ex);
    }

}
```

Multiple Recipients

A consumer can dequeue a message from a multiconsumer, usual 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.

- In PL/SQL the consumer name is supplied using the `consumer_name` field of the `dequeue_options_t` record.
- In OCI the consumer name is supplied using the `OCISetAttr` procedure to specify a text string as the `OCI_ATTR_CONSUMER_NAME` of an `OCI_DTYPE_AQDEQ_OPTIONS` descriptor.
- In **Oracle Objects for OLE** (OO4O), the consumer name is supplied by setting the consumer property of the `OraAQ` object.

Multiple processes or operating system threads can use the same `consumer_name` to dequeue concurrently from a queue. In that case Oracle Streams AQ provides the first unlocked message that is at the head of the queue and is intended for the consumer. Unless the message ID of a specific message is specified during dequeue, the 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.

The exception queue must also be a multiconsumer 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. Hence, from a dequeue perspective, multiconsumer exception queues act like single-consumer queues because each expired message can be dequeued only once using a `NULL` consumer name. Expired messages can be dequeued only by specifying a message ID if the multiconsumer exception queue was created in a queue table with the compatible parameter set to '8.0'.

Beginning with 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 when the messages have been completely processed and when they are physically removed from the queue.

See Also:

- "Adding a Subscriber" on page 8-26
- "Enqueuing a Message and Specifying Options" on page 10-3

Local and Remote Recipients

Consumers of a message in multiconsumer queues (either by virtue of being a subscriber to the queue or because the consumer was a recipient in the enqueuer's recipient list) can be local or remote.

- A **local consumer** dequeues the message from the same queue into which the producer enqueued the message. Local consumers have a non-`NULL` `NAME` and `NULL` `ADDRESS` and `PROTOCOL` field in the `AQ$_AGENT` type.

  See Also: "AQ Agent Type (aq$_agent)" on page 3-3

- A **remote consumer** dequeues from a queue that is different from the queue where the message was enqueued. As such, users must be familiar with and use the Oracle Streams AQ propagation feature to use remote consumers. Remote consumers can fall into one of three categories:

  a. The `ADDRESS` field 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` where `queue_name` (optionally qualified by the schema name) is the target queue. If the schema is not specified, then the schema of the current user executing the `ADD_SUBSCRIBER` procedure or the enqueue is used. Use the `DBMS_AQADM.SCHEDULE_PROPAGATION` command with a `NULL` destination (which is the default) to schedule propagation to such remote consumers.

  See Also:

  - "Adding a Subscriber" on page 8-26
  - "Enqueuing a Message" on page 10-2
  - "Scheduling a Queue Propagation" on page 8-32

  b. The `ADDRESS` field 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 executing the `ADD_SUBSCRIBER` procedure or the enqueue is used. If the database link is not a fully qualified name (does not have a domain name specified), then the default domain as specified by the `db_domain` `init.ora` parameter is used. Use the `DBMS_AQADM.SCHEDULE_PROPAGATION` procedure with the database link as the destination to schedule the propagation. Oracle Streams AQ does not support the use of synonyms to refer to queues or database links.

c. The `ADDRESS` field 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 on how to schedule propagation.

When a consumer is remote, a message is marked as `PROCESSED` in the source queue immediately after the message has been propagated, even though the consumer may not have dequeued the message at the remote queue. Similarly, when a propagated message expires at the remote queue, the message is moved to the `DEFAULT` exception queue of the remote queue's queue table, and not to the exception queue of the local queue. As can be seen in both cases, Oracle Streams AQ does not currently propagate the exceptions to the source queue. You can use the `MSGID` and the `ORIGINAL_MSGID` columns in the queue table view (`AQ$`*`queue_table`*) to chain the propagated messages. When a message with message ID m1 is propagated to a remote queue, m1 is stored in the `ORIGINAL_MSGID` column of the remote queue.

The `DELAY`, `EXPIRATION` and `PRIORITY` parameters apply identically to both local and remote consumers. Oracle Streams AQ accounts for any delay in propagation by adjusting the `DELAY` and `EXPIRATION` parameters accordingly. For example, if the `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.

Because the database handles message propagation, OO4O does not differentiate between remote and local recipients. The same sequence of calls/steps are required to dequeue a message for local and remote recipients.

Message Navigation in Dequeue

You have several options for selecting a message from a queue. You can select the first message. Alternatively, once you have selected a message and established its position in the queue (for example, as the fourth message), you can then retrieve the next message.

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.

These selections work in a slightly different way if the queue is enabled for transactional grouping.

- If FIRST_MESSAGE is requested, then the dequeue position is reset to the beginning of the queue.
- If NEXT_MESSAGE is requested, then the position is set to the next message of the same transaction
- If NEXT_TRANSACTION is requested, then the position is set to the first message of the next transaction.

The transaction grouping property is negated if a dequeue is performed in one of the following ways: dequeue by specifying a correlation identifier, dequeue by specifying a message identifier, or dequeuing some of the messages of a transaction and committing.

In navigating through the queue, if the program reaches 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.

See Also: "Dequeue Methods" on page 7-61

Scenario

The following scenario in the BooksOnLine example continues the message grouping example already discussed with regard to enqueuing.

The get_orders() procedure dequeues orders from the OE_neworders_que. Recall that each transaction refers to an order and each message corresponds to an individual book in the order. The get_orders() procedure loops through the messages to dequeue the book orders. It resets the position to the beginning of the queue using the FIRST_MESSAGE option before the first dequeues. It then uses the NEXT_MESSAGE navigation option to retrieve the next book (message) of an order (transaction). If it gets an error message indicating all messages in the current group/transaction have been fetched, then it changes the navigation option to NEXT_TRANSACTION and gets the first book of the next order. It then changes the navigation option back to `NEXT_MESSAGE` for fetching subsequent messages in the same transaction. This is repeated until all orders (transactions) have been fetched.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT boladm/boladm;

create or replace procedure get_new_orders as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
qname                    VARCHAR2(30);
no_messages              exception;
end_of_group             exception;
pragma exception_init    (no_messages, -25228);
pragma exception_init    (end_of_group, -25235);
new_orders               BOOLEAN := TRUE;

BEGIN

        dopt.wait := 1;
        dopt.navigation := DBMS_AQ.FIRST_MESSAGE;
        qname := 'OE.OE_neworders_que';
        WHILE (new_orders) LOOP
          BEGIN
            LOOP
                BEGIN
                    dbms_aq.dequeue(
                        queue_name          => qname,
                        dequeue_options     => dopt,
                        message_properties  => mprop,
                        payload             => deq_order_data,
                        msgid               => deq_msgid);

                    deq_item_data := deq_order_data.items(1);
                    deq_book_data := deq_item_data.item;
                    deq_cust_data := deq_order_data.customer;

                    IF (deq_cust_data IS NOT NULL) THEN
                      dbms_output.put_line(' **** NEXT ORDER **** ');
```

```
                    dbms_output.put_line('order_num: ' ||
                               deq_order_data.orderno);
                    dbms_output.put_line('ship_state: ' ||
                               deq_cust_data.state);
                  END IF;
                  dbms_output.put_line(' ---- next book ---- ');
                  dbms_output.put_line(' book_title: ' ||
                               deq_book_data.title ||
                               ' quantity: ' || deq_item_data.quantity);
              EXCEPTION
                  WHEN end_of_group THEN
                    dbms_output.put_line ('*** END OF ORDER ***');
                    commit;
                    dopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
              END;
          END LOOP;
        EXCEPTION
          WHEN no_messages THEN
               dbms_output.put_line (' ---- NO MORE NEW ORDERS ---- ');
               new_orders := FALSE;
        END;
      END LOOP;

END;
/

CONNECT EXECUTE ON get_new_orders to OE;

/*  Dequeue the orders: */
CONNECT OE/OE;
EXECUTE BOLADM.get_new_orders;
Visual Basic (OO4O): Example Code
Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraItemList,OraItem,OraBook,OraCustomer  as Object
Dim Msgid as String

   Set OraSession = CreateObject("OracleInProcServer.XOraSession")
   Set OraDatabase = OraSession.DbOpenDatabase("", "boladm/boladm", 0&)
   set oraAq = OraDatabase.CreateAQ("OE.OE_neworders_que")
   Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
       OraAq.wait = 1
   OraAq.Navigation = ORAAQ_DQ_FIRST_MESSAGE
```

```
private sub get_new_orders
   Dim MsgIsDequeued as Boolean
   On Error goto ErrHandler
   MsgIsDequeued = TRUE
       msgid = q.Dequeue
        if MsgIsDequeued then
      set OraOrder = OraMsg
      OraItemList = OraOrder("items")
      OraItem = OraItemList(1)
      OraBook = OraItem("item")
      OraCustomer = OraOrder("customer")

         ' Populate the textboxes with the values
      if( OraCustomer ) then
         if OraAq.Navigation <> ORAAQ_DQ_NEXT_MESSAGE then
            MsgBox " ********* NEXT ORDER *******"
         end if
         txt_book_orderno = OraOrder("orderno")
         txt_book_shipstate = OraCustomer("state")
      End if
      OraAq.Navigation = ORAAQ_DQ_NEXT_MESSAGE
      txt_book_title = OraBook("title")
      txt_book_qty  = OraItem("quantity")
    Else
      MsgBox " ********* END OF  ORDER *******"
   End if

ErrHandler:
   'Handle error case, like no message etc
   If OraDatabase.LastServerErr = 25228 then
      OraAq.Navigation = ORAAQ_DQ_NEXT_TRANSACTION
      MsgIsDequeued = FALSE
      Resume Next
   End If
   'Process other errors
end sub
```

Java (JDBC): Example Code

No example is provided with this release.

Modes of Dequeuing

A dequeue request can either view a message or delete a message.

- To view a message, you can use the browse mode or locked mode.
- To consume a message, you can use either the remove mode or remove with no data mode.

If a message is browsed, then it remains available for further processing. Similarly if a message is locked, then it remains available for further processing after the lock is released by performing a transaction commit or rollback. After a message is consumed, using either of the remove modes, it is no longer available for dequeue requests.

You can use the `REMOVE` mode to read a message and delete it. The message can be retained in the queue table based on the retention properties. When the `REMOVE` mode is specified, `DEQ_TIME`, `DEQ_USER_ID`, and `DEQ_TXN_ID` (as seen in the `AQ$`*`Queue_Table_Name`* view) are updated for the consumer that dequeued the message.[

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. In this way, you can avoid the overhead of payload retrieval that can be substantial for large payloads.

A message is retained in the queue table after it has been consumed only if a retention time is specified for a queue. Messages cannot be retained in exception queues (refer to the section on exceptions for further information). Removing a message with no data is generally used if the payload is known (from a previous browse/locked mode dequeue call), or if the message will not be used.

After a message has been browsed, there is no guarantee that the message can be dequeued again, 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.

In general, use care while using the browse mode. The dequeue position 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. Hence repeating a dequeue call in the browse mode with the `NEXT_MESSAGE` navigation option and a nonzero wait time can dequeue 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.

See Also: "Dequeuing a Message" on page 10-28

Scenario

In the following scenario from the BooksOnLine example, international orders destined to Mexico and Canada are to be processed separately due to trade policies and carrier discounts. Hence, a message is viewed in the locked mode (so no other concurrent user removes the message) and the customer country (message payload) is checked. If the customer country is Mexico or Canada, then the message is consumed (deleted from the queue) using `REMOVE_NODATA` (because the payload is already known). Otherwise, the lock on the message is released by the commit call. The remove dequeue call uses the message identifier obtained from the locked mode dequeue call. The `shipping_bookedorder_deq` (refer to the example code for the description of this procedure) call illustrates the use of the browse mode.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT boladm/boladm;

create or replace procedure get_northamerican_orders as

deq_cust_data             BOLADM.customer_typ;
deq_book_data             BOLADM.book_typ;
deq_item_data             BOLADM.orderitem_typ;
deq_msgid                 RAW(16);
dopt                      dbms_aq.dequeue_options_t;
mprop                     dbms_aq.message_properties_t;
deq_order_data            BOLADM.order_typ;
deq_order_nodata          BOLADM.order_typ;
qname                     VARCHAR2(30);
no_messages               exception;
pragma exception_init     (no_messages, -25228);
new_orders                BOOLEAN := TRUE;

begin

        dopt.consumer_name := consumer;
        dopt.wait := DBMS_AQ.NO_WAIT;
        dopt.navigation := dbms_aq.FIRST_MESSAGE;
```

```
dopt.dequeue_mode := DBMS_AQ.LOCKED;

qname := 'TS.TS_bookedorders_que';

WHILE (new_orders) LOOP
  BEGIN
    dbms_aq.dequeue(
        queue_name => qname,
        dequeue_options => dopt,
        message_properties => mprop,
        payload => deq_order_data,
        msgid => deq_msgid);

    deq_item_data := deq_order_data.items(1);
    deq_book_data := deq_item_data.item;
    deq_cust_data := deq_order_data.customer;

    IF (deq_cust_data.country = 'Canada' OR
        deq_cust_data.country = 'Mexico' ) THEN

        dopt.dequeue_mode := dbms_aq.REMOVE_NODATA;
        dopt.msgid := deq_msgid;
        dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_nodata,
                msgid => deq_msgid);
        commit;

        dbms_output.put_line(' **** next booked order **** ');
        dbms_output.put_line('order_no: ' || deq_order_data.orderno ||
                ' book_title: ' || deq_book_data.title ||
                ' quantity: ' || deq_item_data.quantity);
        dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
                ' ship_country: ' || deq_cust_data.country ||
                ' ship_order_type: ' || deq_order_data.ordertype);

    END IF;

    commit;
    dopt.dequeue_mode := DBMS_AQ.LOCKED;
    dopt.msgid := NULL;
    dopt.navigation := dbms_aq.NEXT_MESSAGE;
  EXCEPTION
```

```
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE BOOKED ORDERS ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/

CONNECT EXECUTE on get_northamerican_orders to TS;

CONNECT ES/ES;

/*  Browse all booked orders for East_Shipping: */
EXECUTE BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.BROWSE);

CONNECT TS/TS;

/*  Dequeue all international North American orders for Overseas_Shipping: */
EXECUTE BOLADM.get_northamerican_orders;
```

Visual Basic (OO4O): Example Code

OO4O supports all the modes of dequeuing described earlier. Possible values include:

- ORAAQ_DQ_BROWSE (1) - Do not lock when dequeuing
- ORAAQ_DQ_LOCKED (2) - Read and obtain a write lock on the message
- ORAAQ_DQ_REMOVE (3)(Default) -Read the message and update or delete it.

```
Dim OraSession as object
Dim OraDatabase as object
Dim OraAq as object
Dim OraMsg as Object
Dim OraOrder,OraItemList,OraItem,OraBook,OraCustomer  as Object
Dim Msgid as String

   Set OraSession = CreateObject("OracleInProcServer.XOraSession")
   Set OraDatabase = OraSession.DbOpenDatabase("", "boladm/boladm", 0&)
   set oraAq = OraDatabase.CreateAQ("OE.OE_neworders_que")
       OraAq.DequeueMode = ORAAQ_DQ_BROWSE
```

Java (JDBC): Example Code

```
public static void get_northamerican_orders(Connection db_conn)
{

    AQSession         aq_sess;
    Order             deq_order;
    Customer          deq_cust;
    String            cust_country;
    byte[]            deq_msgid;
    AQDequeueOption   deq_option;
    AQMessageProperty msg_prop;
    AQQueue           bookedorders_q;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    boolean           new_orders = true;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        deq_option = new AQDequeueOption();

        deq_option.setConsumerName("Overseas_Shipping");
        deq_option.setWaitTime(AQDequeueOption.WAIT_NONE);
        deq_option.setNavigationMode(AQDequeueOption.NAVIGATION_FIRST_MESSAGE);
        deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_LOCKED);

        bookedorders_q = aq_sess.getQueue("TS", "TS_bookedorders_que");

        while(new_orders)
        {
            try
            {
              /* Dequeue the message - browse with lock */
              message = bookedorders_q.dequeue(deq_option, Order.getFactory());

              obj_payload = message.getObjectPayload();

              deq_msgid = message.getMessageId();
              deq_order = (Order)(obj_payload.getPayloadData());

              deq_cust = deq_order.getCustomer();
```

```
                cust_country = deq_cust.getCountry();

                if(cust_country.equals("Canada") ||
                   cust_country.equals("Mexico"))
                {
                  deq_option.setDequeueMode(
                                  AQDequeueOption.DEQUEUE_REMOVE_NODATA);
                  deq_option.setMessageId(deq_msgid);


                  /* Delete the message */
                  bookedorders_q.dequeue(deq_option, Order.getFactory());

                  System.out.println("----  next booked order ------");
                  System.out.println("Order no: " + deq_order.getOrderno());
                  System.out.println("Ship state: " + deq_cust.getState());
                  System.out.println("Ship country: " + deq_cust.getCountry());
                  System.out.println("Order type: " + deq_order.getOrdertype());

                }

                db_conn.commit();

                deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_LOCKED);
                deq_option.setMessageId(null);
                deq_option.setNavigationMode(
                                      AQDequeueOption.NAVIGATION_NEXT_MESSAGE);
              }
              catch (AQException aqex)
              {
                new_orders = false;
                System.out.println("--- No more booked orders ----");
                System.out.println("Exception-1: " + aqex);
              }
          }

      }
      catch (Exception ex)
      {
          System.out.println("Exception-2: " + ex);
      }

  }
```

Optimization of Waiting for Arrival of Messages

Oracle Streams AQ allows applications to block on one or more queues waiting for the arrival of either a newly enqueued message or for a message that becomes ready. You can use the DEQUEUE operation to wait for the arrival of a message in a 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. By contrast, 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.

Applications can optionally specify a timeout of zero or more seconds to indicate the time that Oracle Streams AQ must wait for the arrival of a message. The default is to wait forever until a message arrives in the queue. This optimization is important in two ways. It 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. Applications can also perform a blocking dequeue on exception queues to wait for arrival of EXPIRED messages.

A process or thread 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. Applications cannot only wait for the arrival of a message in the queue that an enqueuer enqueues a message, but also on a remote queue, if propagation has been scheduled to the remote queue using DBMS_AQADM.SCHEDULE_PROPAGATION. In this case, the Oracle Streams AQ propagator wakes up the blocked dequeuer after a message has been propagated.

See Also:

- "Dequeuing a Message" on page 10-28
- "Listening to One or More Queues" on page 10-17

Scenario

In the BooksOnLine example, the get_rushtitles procedure discussed under dequeue methods specifies a wait time of 1 second in the dequeue_options argument for the dequeue call. Wait time can be specified in different ways as illustrated in the following code.

- If the wait time is specified as 10 seconds, then the dequeue call is blocked with a timeout of 10 seconds until a message is available in the queue. This means that if there are no messages in the queue after 10 seconds, the dequeue call returns without a message. Predefined constants can also be assigned for the wait time.
- If the wait time is specified as `DBMS_AQ.NO_WAIT`, then a wait time of 0 seconds is implemented. The dequeue call in this case returns immediately even if there are no messages in the queue.
- If the wait time is specified as `DBMS_AQ.FOREVER`, then the dequeue call is blocked without a timeout until a message is available in the queue.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* dopt is a variable of type dbms_aq.dequeue_options_t.
   Set the dequeue wait time to 10 seconds: */
dopt.wait := 10;

/* Set the dequeue wait time to 0 seconds: */
dopt.wait := DBMS_AQ.NO_WAIT;

/* Set the dequeue wait time to infinite (forever): */
dopt.wait := DBMS_AQ.FOREVER;
```

Visual Basic (OO4O): Example Code

OO4O supports asynchronous dequeuing of messages. First, the monitor is started for a particular queue. When messages that fulfil the user criteria are dequeued, the user's callback object is notified.

Java (JDBC): Example Code

AQDequeueOption deq-opt;

deq-opt = new AQDequeueOption ();

Retry with Delay Interval

If the transaction dequeuing the message from a queue fails, then it is regarded as an unsuccessful attempt to consume the message. Oracle Streams AQ 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 Streams AQ allows the application to specify, at the queue level, the maximum number of retries for messages in the queue. 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.

> **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.

Retry Delay

A bad condition can cause the transaction receiving a message to end. Oracle Streams AQ allows users to hide the bad message for a prespecified interval. A `retry_delay` can be specified along with maximum retries. This means that a message that has had a failed attempt is visible in the queue for dequeue after the `retry_delay` interval. Until then it is in the WAITING state. In the Oracle Streams AQ background process, the time manager enforces the retry delay property. The default value for maximum retries is 5. The default value for retry delay is 0. Maximum retries and retry delay are not available with 8.0-compatible multiconsumer queues.

PL/SQL (DBMS_AQADM Package): Example Code

```
/*  Create a package that enqueue with delay set to one day: /*
     CONNECT BOLADM/BOLADM
    >
     /* queue has max retries = 4 and retry delay = 12 hours */
     EXECUTE DBMS_AQADM.ALTER_QUEUE(queue_name = 'WS.WS_BOOKED_ORDERS_QUE',
     max_retr
     ies = 4,
                                    retry_delay = 3600*12);
    >
     /* processes the next order available in the booked_order_queue */
     CREATE OR REPLACE PROCEDURE  process_next_order()
     AS
       dqqopt                   dbms_aq.dequeue_options_t;
       msgprop                  dbms_aq.message_properties_t;
       deq_msgid                RAW(16);
       book                     BOLADM.book_typ;
       item                     BOLADM.orderitem_typ;
       BOLADM.order_typ         order;
     BEGIN
    >
       dqqopt.dequeue_option := DBMS_AQ.FIRST_MESSAGE;
```

```
    dbms_aq.dequeue('WS.WS_BOOKED_ORDERS_QUEUE', dqqopt, msgprop, order,
  deq_msgid
  );
 >
    /* For simplicity, assume order has a single item */
   item = order.items(1);
    book = the_orders.item;
 >
    /* assume search_inventory searches inventory for the book */
    /* if we don't find the book in the warehouse, terminate transaction */
    IF  (search_inventory(book) != TRUE)
        rollback;
    ELSE
       process_order(order);
    END IF;
 >
  END;
  /
```

Visual Basic (OO4O): Example Code

Use the dbexecutesql interface from the database for this functionality.

Java (JDBC): Example Code

```
public static void setup_queue(Connection db_conn)
{
    AQSession          aq_sess;
    AQQueue            bookedorders_q;
    AQQueueProperty    q_prop;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");

        /* Alter queue - set max retries = 4 and retry delay = 12 hours */
        q_prop = new AQQueueProperty();
        q_prop.setMaxRetries(4);

        q_prop.setRetryInterval(3600*12);  // specified in seconds

        bookedorders_q.alterQueue(q_prop);
```

```
    }
    catch (Exception ex)
    {
        System.out.println("Exception: " + ex);
    }

}
public static void process_next_order(Connection db_conn)
{
    AQSession         aq_sess;
    Order             deq_order;
    OrderItem         order_item;
    Book              book;
    AQDequeueOption   deq_option;
    AQMessageProperty msg_prop;
    AQQueue           bookedorders_q;
    AQMessage         message;
    AQObjectPayload   obj_payload;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        deq_option = new AQDequeueOption();

        deq_option.setNavigationMode(AQDequeueOption.NAVIGATION_FIRST_MESSAGE);

        bookedorders_q = aq_sess.getQueue("WS", "WS_bookedorders_que");


        /* Dequeue the message */
        message = bookedorders_q.dequeue(deq_option, Order.getFactory());

        obj_payload = message.getObjectPayload();

        deq_order = (Order)(obj_payload.getPayloadData());

        /* For simplicity, assume order has a single item */
        order_item = deq_order.getItems().getElement(0);
        book = order_item.getItem();

        /* assume search_inventory searches inventory for the book
         * if we don't find the book in the warehouse, terminate transaction
```

```
         */
        if(search_inventory(book) != true)
          db_conn.rollback();
        else
          process_order(deq_order);

    }
    catch (AQException aqex)
    {
        System.out.println("Exception-1: " + aqex);
    }
    catch (Exception ex)
    {
        System.out.println("Exception-2: " + ex);
    }
}
```

Exception Handling

Oracle Streams AQ provides four integrated mechanisms to support exception handling in applications:

- EXCEPTION_QUEUES
- EXPIRATION
- MAX_RETRIES
- RETRY_DELAY

An exception_queue is a repository for all 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 from the exception queue.

When a message has expired, it is moved to an exception queue. The exception queue for a message in a multiconsumer queue should be created in a multiconsumer queue table. However, the exception queue always acts like a single-consumer queue. You cannot add subscribers to an exception queue. The consumer name specified while dequeuing should be null.

Like any other queue, the exception queue must be enabled for dequeue using the DBMS_AQADM.START_QUEUE procedure. You get an Oracle Streams AQ error if you try to enable an exception queue for enqueue.

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. Hence, from a dequeue perspective multiconsumer exception queues act like single-consumer queues, because each expired message can be dequeued only once using a `NULL` consumer name. Messages can also be dequeued from the exception queue by specifying the message ID.

The exception queue is a message property that can be specified during enqueue time. In PL/SQL users can use the `exception_queue` attribute of the `DBMS_AQ.MESSAGE_PROPERTIES_T` record to specify the exception queue. In OCI users can use the `OCISetAttr` procedure to set the `OCI_ATTR_EXCEPTION_QUEUE` attribute of the `OCIAQMsgProperties` descriptor.

See Also: "Enqueuing a Message and Specifying Options" on page 10-3

If an exception queue is not specified, then the default exception queue is used. If the queue is created in a queue table, for example, `QTAB`, then the default exception queue is called `AQ$_QTAB_E`. The default exception queue is automatically created when the queue table is created. Messages are moved to the exception queues by Oracle Streams AQ under the following conditions:

- The message is not being dequeued within the specified expiration interval. For messages intended for more than one recipient, 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 `DBMS_AQ.NEVER`, meaning the messages does not expire.
- The message is being dequeued successfully, but the application that dequeues the message chooses to roll back the transaction because of an error that arises while processing the message. In this case, the message is returned to the queue and is available for any applications that are waiting to dequeue from the same queue. A dequeue is considered rolled back or undone if the application rolls back the entire transaction, or if it rolls back to a save point that was taken before the dequeue. 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 messages intended for multiple recipients, each message keeps a separate retry count 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 5 for single-consumer queues and

8.1-compatible multiconsumer queues. No retry limit is supported for 8.0-compatible multiconsumer queues.

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 executed by the client contains a dequeue that succeeded but the statement itself was undone later due to an exception. To understand this case, consider a PL/SQL procedure that contains a call to DBMS_AQ.DEQUEUE. If the dequeue procedure succeeds but the PL/SQL procedure raises an exception, then Oracle Streams AQ attempts to increment the RETRY_COUNT of the message returned by the dequeue procedure.
- The client program successfully dequeued a message but terminated before committing the transaction.

Messages intended for 8.1-compatible multiconsumer queues cannot be dequeued by the intended recipients once the messages have been moved to an exception queue. These messages should instead be dequeued in the REMOVE or BROWSE mode exactly once by specifying a NULL consumer name in the dequeue options. The messages can also be dequeued by their message IDs.

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

Users can associate a RETRY_DELAY with a queue. The default value for this parameter is 0, meaning that the message is available for dequeue immediately after the RETRY_COUNT is incremented. Otherwise the message is unavailable for RETRY_DELAY seconds. After RETRY_DELAY seconds, the queue monitor marks the message as READY.

For a multiconsumer queue, RETRY_DELAY is for each subscriber.

Scenario

In the BooksOnLine application, the business rule for each shipping region is that an order is placed in a back order queue if the order cannot be filled immediately. The back order application tries to fill the order once a day. If the order cannot be filled within 5 days, then it is placed in an exception queue for special processing. You can implement this process by making use of the retry and exception handling features in Oracle Streams AQ.

The following example shows how you can create a queue with specific maximum retry and retry delay interval.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Example for creating a backorder queue in Western Region which allows a
   maximum of 5 retries and 1 day delay between each retry. */
CONNECT BOLADM/BOLADM
BEGIN
  DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'WS.WS_backorders_que',
        queue_table             => 'WS.WS_orders_mqtab',
        max_retries             => 5,
        retry_delay             => 60*60*24);
END;
/

/* Create an exception queue for the backorder queue for Western Region. */
CONNECT BOLADM/BOLADM
BEGIN
  DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'WS.WS_backorders_excpt_que',
        queue_table             => 'WS.WS_orders_mqtab',
        queue_type              => DBMS_AQADM.EXCEPTION_QUEUE);
end;
/

/* Enqueue a message to WS_backorders_que and specify WS_backorders_excpt_que as
the exception queue for the message: */
CONNECT BOLADM/BOLADM
CREATE OR REPLACE PROCEDURE enqueue_WS_unfilled_order(backorder order_typ)
 AS
   back_order_queue_name    varchar2(62);
   enqopt                   dbms_aq.enqueue_options_t;
   msgprop                  dbms_aq.message_properties_t;
   enq_msgid                raw(16);
 BEGIN

   /* Set backorder queue name for this message: */
   back_order_queue_name := 'WS.WS_backorders_que';

   /* Set exception queue name for this message: */
   msgprop.exception_queue := 'WS.WS_backorders_excpt_que';

   dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
```

```
                  backorder, enq_msgid);
END;
/
```

Visual Basic (OO4O): Example Code

The exception queue is a message property that can be provided at the time of enqueuing a message. If this property is not set, then the default exception queue of the queue is used for any error conditions.

```
set oraAq = OraDatabase.CreateAQ("CBADM.deferbilling_que")
Set OraMsg = OraAq.AQMsg(ORATYPE_OBJECT, "BOLADM.order_typ")
Set OraOrder = OraDatabase.CreateOraObject("BOLADM.order_typ")
OraMsg = OraOrder
   OraMsg.delay = 15*60*60*24
   OraMsg.ExceptionQueue = "WS.WS_backorders_que"
   'Fill up the order values
   OraMsg = OraOrder 'OraOrder contains the order details
   Msgid = OraAq.enqueue
```

Java (JDBC): Example Code

```
public static void createBackOrderQueues(Connection db_conn)
{
    AQSession         aq_sess;
    AQQueue           backorders_q;
    AQQueue           backorders_excp_q;
    AQQueueProperty   q_prop;
    AQQueueProperty   q_prop2;
    AQQueueTable      mq_table;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        mq_table = aq_sess.getQueueTable("WS", "WS_orders_mqtab");


        /* Create a backorder queue in Western Region which allows a
           maximum of 5 retries and 1 day delay between each retry. */

        q_prop = new AQQueueProperty();
        q_prop.setMaxRetries(5);
        q_prop.setRetryInterval(60*24*24);
```

```
        backorders_q = aq_sess.createQueue(mq_table, "WS_backorders_que",
                                           q_prop);

        backorders_q.start(true, true);

        /* Create an exception queue for the backorder queue for
           Western Region. */
        q_prop2 = new AQQueueProperty();
        q_prop2.setQueueType(AQQueueProperty.EXCEPTION_QUEUE);

        backorders_excp_q = aq_sess.createQueue(mq_table,
                                           "WS_backorders_excpt_que", q_prop2);

    }
    catch (Exception ex)
    {
        System.out.println("Exception " + ex);
    }

}

/* Enqueue a message to WS_backorders_que and specify WS_backorders_excpt_que
   as the exception queue for the message: */
public static void enqueue_WS_unfilled_order(Connection db_conn,
                                             Order back_order)
{
    AQSession         aq_sess;
    AQQueue           back_order_q;
    AQEnqueueOption   enq_option;
    AQMessageProperty m_property;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    byte[]            enq_msg_id;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        back_order_q = aq_sess.getQueue("WS", "WS_backorders_que");

        message = back_order_q.createMessage();

        /* Set exception queue name for this message: */
        m_property = message.getMessageProperty();
```

```
            m_property.setExceptionQueue("WS.WS_backorders_excpt_que");

            obj_payload = message.getObjectPayload();
            obj_payload.setPayloadData(back_order);

            enq_option = new AQEnqueueOption();

            /* Enqueue the message */
            enq_msg_id = back_order_q.enqueue(enq_option, message);

            db_conn.commit();
        }
        catch (Exception ex)
        {
            System.out.println("Exception: " + ex);
        }

    }
```

Rule-Based Subscription

Messages can be routed to various recipients based on message properties or message content. Users define a rule-based subscription for a given queue to specify interest in receiving messages that meet particular conditions.

Rules are Boolean expressions that evaluate to `TRUE` or `FALSE`. Similar in syntax to the `WHERE` clause of a SQL query, rules are expressed in terms of the 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. This feature thus supports the notions of content-based subscriptions and content-based routing of messages.

Subscription rules can also be defined on an attribute of type XMLType using XML operators such as `ExistsNode`.

Scenario

For the BooksOnLine application, we illustrate how rule-based subscriptions are used to implement a publish/subscribe paradigm utilizing content-based subscription and content-based routing of messages. The interaction between the Order Entry application and each of the Shipping Applications is modeled as follows:

- Western Region Shipping handles orders for the Western Region of the U.S.
- Eastern Region Shipping handles orders for the Eastern Region of the U.S.
- Overseas Shipping handles all non-U.S. orders.
- Overseas Shipping checks for the `XMLType` attribute to identify special handling.
- Eastern Region Shipping also handles all U.S. rush orders.

Each shipping application subscribes to the OE booked orders queue. The following rule-based subscriptions are defined by the Order Entry user to handle the routing of booked orders from the Order Entry application to each of the Shipping applications.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT OE/OE;
```

Western Region Shipping defines an agent called '`West_Shipping`' with the `WS` booked orders queue as the agent address (destination queue where messages must be delivered). This agent subscribes to the OE booked orders queue using a rule specified on order region and `ordertype` attributes.

```
/*  Add a rule-based subscriber for West Shipping -
    West Shipping handles Western Region U.S. orders,
    Rush Western Region orders are handled by Eastern Shipping: */
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('West_Shipping', 'WS.WS_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(
                queue_name => 'OE.OE_bookedorders_que',
                subscriber => subscriber,
                rule       => 'tab.user_data.orderregion =
                    ''WESTERN'' AND tab.user_data.ordertype != ''RUSH''');
END;
/
```

Eastern Region Shipping defines an agent called `East_Shipping` with the `ES` booked orders queue as the agent address (the destination queue where messages must be delivered). This agent subscribes to the `OE` booked orders queue using a rule specified on `orderregion`, `ordertype` and customer attributes.

```
/*  Add a rule-based subscriber for Eastern Shipping -
    Eastern Shipping handles all Eastern Region orders,
    Eastern Shipping also handles all U.S. rush orders: */
DECLARE
```

```
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name => 'OE.OE_bookedorders_que',
        subscriber => subscriber,
        rule       => 'tab.user_data.orderregion = ''EASTERN'' OR
                      (tab.user_data.ordertype = ''RUSH'' AND
                       tab.user_data.customer.country = ''USA'') ');
END;
```

Overseas Shipping defines an agent called `Overseas_Shipping` with the TS booked orders queue as the agent address (destination queue to which messages must be delivered). This agent subscribes to the OE booked orders queue using a rule specified on the `orderregion` attribute. Because the representation of orders at the Overseas Shipping site is different from the representation of orders at the Order Entry site, a transformation is applied before messages are propagated from the Order Entry site to the Overseas Shipping site.

See Also: "Message Format Transformation" on page 7-7

```
/*  Add a rule-based subscriber (for Overseas Shipping) to the Booked orders
queues with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('Overseas_Shipping','TS.TS_bookedorders_que',null);

  DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name     => 'OE.OE_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.orderregion = ''INTERNATIONAL''',
        transformation => 'TS.OE2XML');
END;
```

Assume that the Overseas Shipping site has a subscriber, `Overseas_DHL`, for handling `RUSH` orders. Because `TS_bookedorders_que` has the order details represented as an `XMLType`, the rule uses XPath syntax.

```
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('Overseas_DHL', null, null);
```

```
  DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name     => 'TS.TS_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.extract(''/ORDER_TYP/ORDERTYPE/
                              text()'').getStringVal()=''RUSH''');

END;
```

Visual Basic (OO4O): Example Code

This functionality is currently not available.

Java (JDBC): Example Code

```
public static void addRuleBasedSubscribers(Connection db_conn)
{

    AQSession         aq_sess;
    AQQueue           bookedorders_q;
    String            rule;
    AQAgent           agt1, agt2, agt3;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

        bookedorders_q = aq_sess.getQueue("OE", "OE_booked_orders_que");

        /* Add a rule-based subscriber for West Shipping -
           West Shipping handles Western region U.S. orders,
           Rush Western region orders are handled by East Shipping: */

        agt1 = new AQAgent("West_Shipping", "WS.WS_bookedorders_que");

        rule = "tab.user_data.orderregion = 'WESTERN' AND " +
               "tab.user_data.ordertype != 'RUSH'";

        bookedorders_q.addSubscriber(agt1, rule);

        /*  Add a rule-based subscriber for Eastern Shipping -
            Eastern Shipping handles all Eastern Region orders,
            Eastern Shipping also handles all U.S. rush orders: */

        agt2 = new AQAgent("East_Shipping", "ES.ES_bookedorders_que");
```

```
            rule = "tab.user_data.orderregion = 'EASTERN' OR " +
                   "(tab.user_data.ordertype = 'RUSH' AND " +
                   "tab.user_data.customer.country = 'USA')";

            bookedorders_q.addSubscriber(agt2, rule);

            /*  Add a rule-based subscriber for Overseas Shipping
                Intl Shipping handles all non-U.S. orders: */

            agt3 = new AQAgent("Overseas_Shipping", "TS.TS_bookedorders_que");
            rule = "tab.user_data.orderregion = 'INTERNATIONAL'";

            bookedorders_q.addSubscriber(agt3, rule);
        }
        catch (Exception ex)
        {
            System.out.println("Exception: " + ex);
        }
    }
```

Listen Capability

Oracle Streams AQ can monitor multiple queues for messages with a single call, `LISTEN`. An application can use `LISTEN` to wait for messages for multiple subscriptions. It can also be used by gateway applications to monitor multiple queues. If the `LISTEN` call returns successfully, then a dequeue must be used to retrieve the message.

See Also: "Listening to One or More Queues" on page 10-17

Without the `LISTEN` call, an application which sought to dequeue from a set of queues would continuously poll the queues to determine if there were a message. Alternatively, you could design your application to have a separate dequeue process for each queue. However, if there are long periods with no traffic in any of the queues, then these approaches create unacceptable overhead. The `LISTEN` call is well suited for such applications.

When there are messages for multiple agents in the agent list, `LISTEN` returns with the first agent for whom there is a message. In that sense `LISTEN` is not 'fair' in monitoring the queues. The application designer must keep this in mind when using the call. To prevent one agent from 'starving' other agents for messages, the application can change the order of the agents in the agent list.

Scenario

In the customer service component of the BooksOnLine example, messages from different databases arrive in the customer service queues, indicating the state of the message. The customer service application monitors the queues and whenever there is a message about a customer order, it updates the order status in the `order_status_table`. The application uses the `LISTEN` call to monitor the different queues. Whenever there is a message in any of the queues, it dequeues the message and updates the order status accordingly.

PL/SQL (DBMS_AQADM Package): Example Code

```
CODE (in tkaqdocd.sql)

/* Update the status of the order in the order status table: */
CREATE OR REPLACE PROCEDURE update_status(
                                new_status    IN VARCHAR2,
                                order_msg     IN BOLADM.ORDER_TYP)
IS
 old_status    VARCHAR2(30);
 dummy         NUMBER;
BEGIN

  BEGIN
    /* Query old status from the table: */
    SELECT st.status INTO old_status FROM order_status_table st
       WHERE st.customer_order.orderno = order_msg.orderno;

  /* Status can be 'BOOKED_ORDER', 'SHIPPED_ORDER', 'BACK_ORDER'
     and   'BILLED_ORDER': */

   IF new_status = 'SHIPPED_ORDER' THEN
      IF old_status = 'BILLED_ORDER' THEN
        return;             /* message about a previous state */
      END IF;
   ELSIF new_status = 'BACK_ORDER' THEN
      IF old_status = 'SHIPPED_ORDER' OR old_status = 'BILLED_ORDER' THEN
        return;             /* message about a previous state */
      END IF;
   END IF;

   /* Update the order status:  */
     UPDATE order_status_table st
        SET st.customer_order = order_msg, st.status = new_status;
```

```
    COMMIT;

  EXCEPTION
  WHEN OTHERS  THEN     /* change to no data found */
    /* First update for the order: */
    INSERT INTO order_status_table(customer_order, status)
    VALUES (order_msg, new_status);
    COMMIT;

  END;
END;
/


/* Dequeues message from 'QUEUE' for 'CONSUMER': */
CREATE OR REPLACE PROCEDURE DEQUEUE_MESSAGE(
                         queue      IN   VARCHAR2,
                         consumer   IN   VARCHAR2,
                         message    OUT  BOLADM.order_typ)
IS

dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_msgid                RAW(16);
BEGIN
  dopt.dequeue_mode := dbms_aq.REMOVE;
  dopt.navigation := dbms_aq.FIRST_MESSAGE;
  dopt.consumer_name := consumer;

  dbms_aq.dequeue(
                queue_name => queue,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => message,
                msgid => deq_msgid);
  commit;
END;
/

/* Monitor the queues in the customer service databse for 'time' seconds: */
 CREATE OR REPLACE PROCEDURE MONITOR_STATUS_QUEUE(time  IN  NUMBER)
IS
  agent_w_message   aq$_agent;
  agent_list        dbms_aq.agent_list_t;
  wait_time         INTEGER := 120;
```

```
  no_message        EXCEPTION;
  pragma EXCEPTION_INIT(no_message, -25254);
  order_msg         boladm.order_typ;
  new_status        VARCHAR2(30);
  monitor           BOOLEAN := TRUE;
  begin_time        NUMBER;
  end_time          NUMBER;
BEGIN

 begin_time :=  dbms_utility.get_time;
 WHILE (monitor)
 LOOP
 BEGIN

  /* Construct the waiters list: */
  agent_list(1) := aq$_agent('BILLED_ORDER', 'CS_billedorders_que', NULL);
  agent_list(2) := aq$_agent('SHIPPED_ORDER', 'CS_shippedorders_que',
NULL);
  agent_list(3) := aq$_agent('BACK_ORDER', 'CS_backorders_que', NULL);
  agent_list(4) := aq$_agent('Booked_ORDER', 'CS_bookedorders_que', NULL);

   /* Wait for order status messages: */
   dbms_aq.listen(agent_list, wait_time, agent_w_message);

   dbms_output.put_line('Agent' || agent_w_message.name || ' Address '||
agent_w_message.address);
   /* Dequeue the message from the queue: */
   dequeue_message(agent_w_message.address, agent_w_message.name, order_msg);

   /* Update the status of the order depending on the type of the message,
    * the name of the agent contains the new state: */
   update_status(agent_w_message.name, order_msg);

  /* Exit if we have been working long enough: */
   end_time := dbms_utility.get_time;
   IF  (end_time - begin_time > time)   THEN
     EXIT;
   END IF;

  EXCEPTION
  WHEN  no_message  THEN
    dbms_output.put_line('No messages in the past 2 minutes');
       end_time := dbms_utility.get_time;
    /* Exit if we have accomplished enough work: */
    IF  (end_time - begin_time > time)   THEN
```

```
      EXIT;
    END IF;
  END;

  END LOOP;
END;
/
```

Visual Basic (OO4O): Example Code

Feature not currently available.

Java (JDBC): Example Code

```
public static void monitor_status_queue(Connection db_conn)
{
    AQSession         aq_sess;
    AQAgent[]         agt_list = null;
    AQAgent           ret_agt  = null;
    Order             deq_order;
    AQDequeueOption   deq_option;
    AQQueue           orders_q;
    AQMessage         message;
    AQObjectPayload   obj_payload;
    String            owner = null;
    String            queue_name = null;
    int               idx = 0;

    try
    {
        /* Create an AQ session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

/* Construct the waiters list: */
agt_list = new AQAgent[4];

agt_list[0] = new AQAgent("BILLED_ORDER", "CS_billedorders_que", 0);
agt_list[1] = new AQAgent("SHIPPED_ORDER", "CS_shippedorders_que", 0);
agt_list[2] = new AQAgent("BACK_ORDER", "CS_backorders_que", 0);
agt_list[3] = new AQAgent("BOOKED_ORDER", "CS_bookedorders_que", 0);

/* Wait for order status messages for 120 seconds: */
ret_agt = aq_sess.listen(agt_list, 120);

System.out.println("Message available for agent: " +
   ret_agt.getName()  + "   "  + ret_agt.getAddress());
```

```
/* Get owner, queue where message is available */
idx = ret_agt.getAddress().indexOf(".");

if(idx != -1)
{
  owner = ret_agt.getAddress().substring(0, idx);
  queue_name = ret_agt.getAddress().substring(idx + 1);

/* Dequeue the message */
deq_option = new AQDequeueOption();

deq_option.setConsumerName(ret_agt.getName());
deq_option.setWaitTime(1);

orders_q = aq_sess.getQueue(owner, queue_name);

/* Dequeue the message */
message = orders_q.dequeue(deq_option, Order.getFactory());

obj_payload = message.getObjectPayload();

deq_order = (Order)(obj_payload.getPayloadData());

   System.out.println("Order number " + deq_order.getOrderno() + " retrieved");

   }
   catch (AQException aqex)
   {
   System.out.println("Exception-1: " + aqex);
   }
   catch (Exception ex)
   {
    System.out.println("Exception-2: " + ex);
    }

}
```

Message Transformation During Dequeue

Continuing the scenario introduced in "Message Format Transformation" on page 7-7 and "Message Transformation During Enqueue" on page 7-56, the queues in the OE schema are of payload type `OE.orders_typ` and the queues in the WS schema are of payload type `WS.orders_typ_sh`.

Scenario

At dequeue time, an application can move messages from `OE_booked_orders_topic` to the `WS_booked_orders_topic` by using a selection criteria on dequeue to dequeue only orders with `order_region` "`WESTERN`" and `order_type` not equal to "`RUSH.`" At the same time, the transformation is applied and the order in the `ws.order_typ_sh` type is retrieved. Then the message is enqueued into the `WS.ws_booked_orders` queue.

PL/SQL (DBMS_AQ Package): Example Code

```
CREATE OR REPLACE PROCEDURE   fwd_message_to_ws_shipping AS

  enq_opt   dbms_aq.enqueue_options_t;
  deq_opt   dbms_aq.dequeue_options_t;
  msg_prp   dbms_aq.message_properties_t;
  booked_order WS.order_typ_sh;
BEGIN

/* First dequeue the message from OE booked orders topic: */
    deq_opt.transformation := 'OE.OE2WS';
    deq_opt.condition := 'tab.user_data.order_region = ''WESTERN'' and tab.user_
data.order_type != ''RUSH''';

    dbms_aq.dequeue('OE.oe_bookedorders_topic', deq_opt,
                     msg_prp, booked_order);

/* Enqueue the message in the WS booked orders topic */
    msg_prp.recipient_list(0) := aq$_agent('West_shipping', null, null);

    dbms_aq.enqueue('WS.ws_bookedorders_topic',
                     enq_opt, msg_prp, booked_order);

END;
```

Visual Basic (OO4O): Example Code

No example is provided with this release.

Java (JDBC): Example Code

No example is provided with this release.

Dequeue Using the Oracle Streams AQ XML Servlet

You can perform dequeue requests over the Internet using SOAP.

> **See Also:** Chapter 17, "Internet Access to Oracle Streams AQ"

In the BooksOnline scenario, assume that the Eastern Shipping application receives Oracle Streams AQ messages with a correlation identifier 'RUSH' over the Internet. The dequeue request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>ES_ES_bookedorders_que</destination>
            <consumer_name>East_Shipping</consumer_name>
            <wait_time>0</wait_time>
            <selector>
                 <correlation>RUSH</correlation>
            </selector>
          </consumer_options>

          <AQXmlCommit/>

        </AQXmlReceive>
      </Body>
</Envelope>
```

Asynchronous Notifications

This feature allows clients to receive notifications for messages of interest. It supports multiple mechanisms to receive notifications. 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.

For persistent queues, notifications contain only the message properties, except for JMS notifications. Clients explicitly dequeue to receive the message. In JMS, the dequeue is accomplished as part of the notifications and hence explicit dequeue is not required. For nonpersistent queues, the message is delivered as part of the notification.

Clients can also specify the presentation for notifications as either RAW or XML.

Scenario

In the BooksOnLine application, a customer can request Fed-Ex shipping (priority 1), priority air shipping (priority 2), or regular ground shipping (priority 3).

The shipping application then ships the orders according to the user's request. It is of interest to BooksOnLine to find out how many requests of each shipping type come in each day. The application uses asynchronous notification facility for this purpose. It registers for notification on the WS.WS_bookedorders_que. When it is notified of new message in the queue, it updates the count for the appropriate shipping type depending on the priority of the message.

Visual Basic (OO4O): Example Code

Refer to the Visual Basic online help, "Monitoring Messages".

Java (JDBC): Example Code

This feature is not supported by the Java API.

C (OCI): Example Code

This example illustrates the use of OCIRegister. At the shipping site, an OCI client program keeps track of how many orders were made for each of the shipping types, FEDEX, AIR and GROUND. The priority field of the message enables us to determine the type of shipping wanted.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#ifdef WIN32COMMON
#define sleep(x)   Sleep(1000*(x))
#endif
static text *username = (text *) "WS";
static text *password = (text *) "WS";

static OCIEnv *envhp;
static OCIServer *srvhp;
static OCIError *errhp;
static OCISvcCtx *svchp;
```

```
static void checkerr(/*_ OCIError *errhp, sword status _*/);

struct ship_data
{
  ub4  fedex;
  ub4  air;
  ub4  ground;
};

typedef struct ship_data ship_data;

int main(/*_ int argc, char *argv[] _*/);


/* Notify callback: */
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *pay;
ub4    payl;
dvoid *desc;
ub4    mode;
{
 text                *subname;
 ub4                 size;
 ship_data           *ship_stats = (ship_data *)ctx;
 text                *queue;
 text                *consumer;
 OCIRaw              *msgid;
 ub4                 priority;
 OCIAQMsgProperties  *msgprop;

 OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
                             (dvoid *)&subname, &size,
                             OCI_ATTR_SUBSCR_NAME, errhp);

 /* Extract the attributes from the AQ descriptor.
    Queue name: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
            OCI_ATTR_QUEUE_NAME, errhp);

 /* Consumer name: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &size,
            OCI_ATTR_CONSUMER_NAME, errhp);
```

```
 /* Msgid: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgid, &size,
            OCI_ATTR_NFY_MSGID, errhp);

 /* Message properties: */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
            OCI_ATTR_MSG_PROP, errhp);

 /* Get priority from message properties: */
 checkerr(errhp, OCIAttrGet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
                             (dvoid *)&priority, 0,
                             OCI_ATTR_PRIORITY, errhp));

  switch (priority)
  {
  case 1:  ship_stats->fedex++;
           break;
  case 2: ship_stats->air++;
           break;
  case 3:  ship_stats->ground++;
           break;
  default:
           printf(" Error priority %d", priority);
  }
}


int main(argc, argv)
int argc;
char *argv[];
{
  OCISession *authp = (OCISession *) 0;
  OCISubscription *subscrhp[8];
  ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;
  ship_data  ctx = {0,0,0};
  ub4 sleep_time = 0;

  printf("Initializing OCI Process\n");

  /* Initialize OCI environment with OCI_EVENTS flag set: */
  (void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                     (dvoid * (*)(dvoid *, size_t)) 0,
                     (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                     (void (*)(dvoid *, dvoid *)) 0 );
```

```
  printf("Initialization successful\n");

  printf("Initializing OCI Env\n");
  (void) OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0, (dvoid **) 0
);
  printf("Initialization successful\n");

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_
ERROR,
                   (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_
SERVER,
                   (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_
SVCCTX,
                   (size_t) 0, (dvoid **) 0));


  printf("connecting to server\n");
  checkerr(errhp, OCIServerAttach( srvhp, errhp, (text *)"inst1_alias",
          strlen("inst1_alias"), (ub4) OCI_DEFAULT));
  printf("connect successful\n");

  /* Set attribute server context in the service context: */
  checkerr(errhp, OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
                  (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp));

  checkerr(errhp, OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
                     (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0));

  /* Set username and password in the session handle: */
  checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) username, (ub4) strlen((char *)username),
                (ub4) OCI_ATTR_USERNAME, errhp));

  checkerr(errhp, OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) password, (ub4) strlen((char *)password),
                (ub4) OCI_ATTR_PASSWORD, errhp));

  /* Begin session: */
  checkerr(errhp, OCISessionBegin ( svchp,  errhp, authp, OCI_CRED_RDBMS,
                          (ub4) OCI_DEFAULT));
```

```
  (void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                    (dvoid *) authp, (ub4) 0,
                    (ub4) OCI_ATTR_SESSION, errhp);


  /* Register for notification: */
  printf("allocating subscription handle\n");
  subscrhp[0] = (OCISubscription *)0;
  (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
                         (ub4) OCI_HTYPE_SUBSCRIPTION,
                         (size_t) 0, (dvoid **) 0);

  printf("setting subscription name\n");
  (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) "WS.WS_BOOKEDORDERS_QUE:BOOKED_ORDERS",
                  (ub4) strlen("WS.WS_BOOKEDORDERS_QUE:BOOKED_ORDERS"),
                  (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

  printf("setting subscription callback\n");
  (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) notifyCB, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *)&ctx, (ub4)sizeof(ctx),
                  (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

  printf("setting subscription namespace\n");
  (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                  (dvoid *) &namespace, (ub4) 0,
                  (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

  printf("Registering \n");
  checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 1, errhp,
                                          OCI_DEFAULT));

  sleep_time = (ub4)atoi(argv[1]);
  printf ("waiting for %d s", sleep_time);
  sleep(sleep_time);

  printf("Exiting");
  exit(0);
}
```

```
void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  sb4 errcode = 0;

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
    (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    (void) printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    (void) printf("Error - OCI_NODATA\n");
    break;
  case OCI_ERROR:
    (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                        errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
    (void) printf("Error - %.*s\n", 512, errbuf);
    break;
  case OCI_INVALID_HANDLE:
    (void) printf("Error - OCI_INVALID_HANDLE\n");
    break;
  case OCI_STILL_EXECUTING:
    (void) printf("Error - OCI_STILL_EXECUTE\n");
    break;
  case OCI_CONTINUE:
    (void) printf("Error - OCI_CONTINUE\n");
    break;
  default:
    break;
  }
}
```

PL/SQL (DBMS_AQ package): Example Code

This example illustrates the use of the DBMS_AQ.REGISTER procedure.

In the BooksOnline scenario, assume that we want a PL/SQL callback `WS.notifyCB()` to be invoked when the subscriber `BOOKED_ORDER` receives a message in the `WS.WS_BOOKED_ORDERS_QUE` queue. In addition, we want to send an e-mail to `john@company.com` when an order is enqueued in the queue for subscriber `BOOKED_ORDERS`. Also assume that we want to invoke the **servlet** `http://xyz.company.com/servlets/NofifyServlet`. This can be accomplished as follows:

First define a PL/SQL procedure that is invoked on notification.

```
connect ws/ws;
set echo on;
set serveroutput on;

-- notifyCB callback
create or replace procedure notifyCB(
  context raw, reginfo sys.aq$_reg_info, descr sys.aq$_descriptor,
  payload raw, payloadl number)
AS
  dequeue_options   DBMS_AQ.dequeue_options_t;
  message_properies DBMS_AQ.message_properties_t;
  message_handle    RAW(16);
  message           BOLADM.order_typ;
BEGIN
  -- get the consumer name and msg_id from the descriptor
  dequeue_options.msgid := descr.msg_id;
  dequeue_options.consumer_name := descr.consumer_name;

  -- Dequeue the message
  DBMS_AQ.DEQUEUE(queue_name => descr.queue_name,
                  dequeue_options => dequeue_options,
                  message_properties => message_properties,
                  payload => message,
                  msgid => message_handle);

  commit;

  DBMS_OUTPUT.PUTLINE('Received Order: ' || message.orderno);

END;
/
```

The PL/SQL procedure, e-mail address, and HTTP URL can be registered as follows:

```
connect ws/ws;
set echo on;
set serveroutput on;

DECLARE
  reginfo1     sys.aq$_reg_info;
  reginfo2     sys.aq$_reg_info;
  reginfo3     sys.aq$_reg_info;
  reginfolist  sys.aq$_reg_info_list;

BEGIN
   -- register for the pl/sql procedure notifyCB to be called on notification
  reginfo1 := sys.aq$_reg_info('WS.WS_BOOKEDORDERS_QUE:BOOKED_ORDERS',
                    DBMS_AQ.NAMESPACE_AQ, 'plsql://WS.notifyCB',
                    HEXTORAW('FF'));

  -- register for an e-mail to be sent to john@company.com on notification
  reginfo2 := sys.aq$_reg_info('WS.WS_BOOKEDORDERS_QUE:BOOKED_ORDERS',
                         DBMS_AQ.NAMESPACE_AQ, 'mailto://john@company.com',
                           HEXTORAW('FF'));

  -- register for an HTTP servlet to be invoked for notification
  reginfo3 := sys.aq$_reg_info('WS.WS_BOOKEDORDERS_QUE:BOOKED_ORDERS',
                         DBMS_AQ.NAMESPACE_AQ,
                         'http://xyz.oracle.com/servlets/NotifyServlet',
                           HEXTORAW('FF'));

  -- Create the registration information list
  reginfolist := sys.aq$_reg_info_list(reginfo1);
  reginfolist.EXTEND;
  reginfolist(2) := reginfo2;

  reginfolist.EXTEND;
  reginfolist(3) := reginfo3;

-- do the registration
  sys.dbms_aq.register(reginfolist, 3);

END;
```

Registering for Notifications Using the Oracle Streams AQ XML Servlet

Clients can register for Oracle Streams AQ notifications over the Internet.

See Also: Chapter 17, "Internet Access to Oracle Streams AQ"

The register request has the following format:

```
?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>

        <AQXmlRegister xmlns = "http://ns.oracle.com/AQ/schemas/access">

          <register_options>
            <destination>WS.WS_BOOKEDORDERS_QUE</destination>
            <consumer_name>BOOKED_ORDERS</consumer_name>
            <notify_url>mailto://john@company.com</notify_url>
          </register_options>

          <AQXmlCommit/>
        </AQXmlRegister>
      </Body>
</Envelope>
```

The e-mail notification sent to `john@company.com` has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns="http://www.oracle.com/schemas/IDAP/envelope">
    <Body>
        <AQXmlNotification xmlns="http://www.oracle.com/schemas/AQ/access">
            <notification_options>
                <destination>WS.WS_BOOKEDORDERS_QUE</destination>
            </notification_options>
            <message_set>
                <message>
                    <message_header>
                       <message_id>81128B6AC46D4B15E03408002092AA15</message_id>
                       <correlation>RUSH</correlation>
                       <priority>1</priority>
                       <delivery_count>0</delivery_count>
                       <sender_id>
                            <agent_name>john</agent_name>
                       </sender_id>
                       <message_state>0</message_state>
                    </message_header>
```

```
                </message>
            </message_set>
        </AQXmlNotification>
    </Body>
</Envelope>
```

Propagation Features

In this section, the following topics are discussed:

- Propagation Overview
- Propagation Scheduling
- Scenario
- Enhanced Propagation Scheduling Capabilities
- Exception Handling During Propagation
- Message Format Transformation During Propagation

Propagation Overview

This feature allows applications to communicate with each other without being connected to the same database or to the same queue. Messages can be propagated from one queue to another. The destination queue can be located in the same database or in a remote database. Propagation is performed by job queue background processes. Propagation to the remote queue uses database links over Oracle Net Services or HTTP(S).

The propagation feature is used as follows. First one or more subscribers are defined for the queue from which messages are to be propagated. Second, a schedule is defined for each destination where messages are to be propagated from the queue. Enqueued messages are propagated and automatically available for dequeuing at the destination queues.

> **See Also:** "Subscriptions and Recipient Lists" on page 7-38

For propagation over the Internet, you must specify the remote Internet user in the database link. The remote Internet user must have privileges to enqueue in the destination queue.

Two or more `job_queue` background processes must be running to use propagation. This is in addition to the number of `job_queue` background processes needed for handling non-propagation related jobs. Also, if you want to deploy remote propagation, then you must ensure that the database link specified for the schedule is valid and have proper privileges for enqueuing into the destination queue.

> **See Also:** "Propagation Scheduling" on page 7-112 for more information about the administrative commands for managing propagation schedules

Propagation also has mechanisms for handling failure. For example, if the database link specified is invalid, then the appropriate error message is reported.

Finally, propagation provides detailed statistics about the messages propagated and the schedule itself. This information can be used to properly tune the schedules for best performance.

> **See Also:** "Enhanced Propagation Scheduling Capabilities" on page 7-118 for a discussion of the failure handling and error reporting facilities of propagation and propagation statistics

Propagation Scheduling

A propagation schedule is defined for a pair of source and destination database links. If a queue has messages to be propagated to several queues, then a schedule must be defined for each of the destination queues. 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 source database, load at 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 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 Queue Propagation" on page 8-32

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.

Scenario

In the BooksOnLine example, messages in the `OE_bookedorders_que` are propagated to different shipping sites. The following example code illustrates the various administrative calls available for specifying and managing schedules. It also shows the calls for enqueuing messages into the source queue and for dequeuing the messages at the destination site. The catalog view `USER_QUEUE_SCHEDULES` provides all information relevant to a schedule.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT OE/OE;

/* Schedule Propagation from bookedorders_que to shipping: */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION( \
   queue_name      => 'OE.OE_bookedorders_que');

/* Check if a schedule has been created: */
SELECT * FROM user_queue_schedules;

/* Enqueue some orders into OE_bookedorders_que: */
EXECUTE BOLADM.order_enq('My First   Book', 1, 1001, 'CA', 'USA', \
   'WESTERN', 'NORMAL');
EXECUTE BOLADM.order_enq('My Second  Book', 2, 1002, 'NY', 'USA', \
   'EASTERN', 'NORMAL');
EXECUTE BOLADM.order_enq('My Third   Book', 3, 1003, '',   'Canada', \
   'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Fourth  Book', 4, 1004, 'NV', 'USA', \
   'WESTERN', 'RUSH');
EXECUTE BOLADM.order_enq('My Fifth   Book', 5, 1005, 'MA', 'USA', \
   'EASTERN', 'RUSH');
EXECUTE BOLADM.order_enq('My Sixth   Book', 6, 1006, ''  , 'UK', \
   'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Seventh Book', 7, 1007, '',   'Canada', \
   'INTERNATIONAL', 'RUSH');
EXECUTE BOLADM.order_enq('My Eighth  Book', 8, 1008, '',   'Mexico', \
   'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Ninth   Book', 9, 1009, 'CA', 'USA', \
   'WESTERN', 'RUSH');
```

```
EXECUTE BOLADM.order_enq('My Tenth   Book', 8, 1010, ''  , 'UK', \
   'INTERNATIONAL', 'NORMAL');
EXECUTE BOLADM.order_enq('My Last    Book', 7, 1011, ''  , 'Mexico', \
   'INTERNATIONAL', 'NORMAL');

/* Wait for propagation to happen: */
EXECUTE dbms_lock.sleep(100);

/* Connect to shipping sites and check propagated messages: */
CONNECT WS/WS;
set serveroutput on;

/*  Dequeue all booked orders for West_Shipping: */
EXECUTE BOLADM.shipping_bookedorder_deq('West_Shipping', DBMS_AQ.REMOVE);

CONNECT ES/ES;
SET SERVEROUTPUT ON;

/* Dequeue all remaining booked orders (normal order) for East_Shipping: */
EXECUTE BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.REMOVE);

CONNECT TS/TS;
SET SERVEROUTPUT ON;

/* Dequeue all international North American orders for Overseas_Shipping: */
EXECUTE BOLADM.get_northamerican_orders('Overseas_Shipping');

/* Dequeue rest of the booked orders for Overseas_Shipping: */
EXECUTE BOLADM.shipping_bookedorder_deq('Overseas_Shipping', DBMS_AQ.REMOVE);

/* Disable propagation schedule for booked orders */
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION _SCHEDULE(   \
   queue_name   => 'OE_bookedorders_que');

/* Wait for some time for call to be effected: */
EXECUTE dbms_lock.sleep(30);

/*  Check if the schedule has been disabled: */
SELECT schedule_disabled FROM user_queue_schedules;

/* Alter propagation schedule for booked orders to run every
   15 mins (900 seconds) for a window duration of 300 seconds: */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION _SCHEDULE( \
   queue_name     => 'OE_bookedorders_que', \
   duration       => 300, \
```

```
   next_time      => 'SYSDATE + 900/86400',\
   latency        => 25);

/* Wait for some time for call to be effected: */
EXECUTE dbms_lock.sleep(30);

/*  Check if the schedule parameters have changed: */
SELECT next_time, latency, propagation_window FROM user_queue_schedules;

/* Enable propagation schedule for booked orders:
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION _SCHEDULE( \
   queue_name     => 'OE_bookedorders_que');

/* Wait for some time for call to be effected: */
EXECUTE dbms_lock.sleep(30);

/* Check if the schedule has been enabled: */
SELECT schedule_disabled FROM user_queue_schedules;

/* Unschedule propagation for booked orders: */
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(   \
   queue_name      => 'OE.OE_bookedorders_que');

/* Wait for some time for call to be effected: */
EXECUTE dbms_lock.sleep(30);

/*  Check if the schedule has been dropped
SELECT *  FROM user_queue_schedules;
```

Visual Basic (OO4O): Example Code

This functionality is currently not available.

Java (JDBC): Example Code

No example is provided with this release.

Propagation of Messages with LOB Attributes

Large Objects can be propagated using Oracle Streams AQ 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 `BLOB`s or `CLOB`s (not NCLOBs). If the attribute is a `CLOB`, then Oracle Streams AQ automatically performs any necessary characterset 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 Application Developer's Guide - Large Objects*

Scenario

In the BooksOnLine application, the company may wish to send promotional coupons along with the book orders. These coupons are generated depending on the content of the order, and other customer preferences. The coupons are images generated from some multimedia database, and are stored as `LOB`s.

When the order information is sent to the shipping warehouses, the coupon contents are also sent to the warehouses. In the following code, `order_typ` is enhanced to contain a coupon attribute of LOB type. The code demonstrates how the `LOB` contents are inserted into the message that is enqueued into `OE_bookedorders_que` when an order is placed. The message payload is first constructed with an empty `LOB`. The place holder (`LOB` locator) information is obtained from the queue table and is then used in conjunction with the `LOB` manipulation routines, such as `DBMS_LOB.WRITE()`, to fill the LOB contents. The example illustrates enqueuing and dequeuing of messages with LOBs as part the payload.

A `COMMIT` is applied only after the LOB contents are filled in with the appropriate image data. Propagation automatically takes care of moving the LOB contents along with the rest of the message contents. The following code also shows a dequeue at the destination queue for reading the LOB contents from the propagated message. The LOB contents are read into a buffer that can be sent to a printer for printing the coupon.

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Enhance the type order_typ to contain coupon field (lob field): */
CREATE OR REPLACE TYPE order_typ AS OBJECT (
        orderno         NUMBER,
        status          VARCHAR2(30),
        ordertype       VARCHAR2(30),
        orderregion     VARCHAR2(30),
        customer        customer_typ,
        paymentmethod   VARCHAR2(30),
        items           orderitemlist_vartyp,
        total           NUMBER,
        coupon          BLOB);
/

/* lob_loc is a variable of type BLOB,
   buffer is a variable of type RAW,
   length is a variable of type NUMBER. */

/* Complete the order data and perform the enqueue using the order_enq()
   procedure: */
dbms_aq.enqueue('OE.OE_bookedorders_que', enqopt, msgprop,
                OE_enq_order_data, enq_msgid);

/* Get the lob locator in the queue table after enqueue: */
SELECT t.user_data.coupon INTO lob_loc
FROM   OE.OE_orders_pr_mqtab t
WHERE  t.msgid = enq_msgid;

/* Generate a sample LOB of 100 bytes: */
buffer := hextoraw(rpad('FF',100,'FF'));

/* Fill in the lob using LOB routines in the dbms_lob package: */
dbms_lob.write(lob_loc, 90, 1, buffer);

/* Applies a commit only after filling in lob contents: */
COMMIT;
```

```
/* Sleep until propagation is complete: */

/* Perform dequeue at the Western Shipping warehouse: */
dbms_aq.dequeue(
        queue_name         => qname,
        dequeue_options    => dopt,
        message_properties => mprop,
        payload            => deq_order_data,
        msgid              => deq_msgid);

/* Get the LOB locator after dequeue: */
lob_loc := deq_order_data.coupon;

/* Get the length of the LOB: */
length := dbms_lob.getlength(lob_loc);

/* Read the LOB contents into the buffer: */
dbms_lob.read(lob_loc, length, 1, buffer);
```

Visual Basic (OO4O): Example Code

This functionality is not available currently.

Java (JDBC): Example Code

No example is provided with this release.

Enhanced Propagation Scheduling Capabilities

Detailed information about the schedules can be obtained from the catalog views defined for propagation.

Information about active schedules—such as the name of the background process handling that schedule, the SID (session, serial number) for the session handling the propagation and the Oracle Database instance handling a schedule (relevant if Real Application Clusters are being used)—can be obtained from the catalog views. The same catalog views also provide information about the previous successful execution of a schedule (last successful propagation of message) and the next execution of the schedule.

For each schedule, detailed propagation statistics are maintained:

- 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

These statistics have been designed to provide useful information to the queue administrators for tuning the schedules such that maximum efficiency can be achieved.

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 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 is made, after which the schedule is automatically disabled. 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. During a retry, if propagation is successful, the number of failures is reset to 0.

Propagation has support built-in for Oracle Real Application Clusters and 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 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. 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` 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.

Scenario

In the BooksOnLine example, the `OE_bookedorders_que` is a busy queue, because messages in it are propagated to different shipping sites. The following example code illustrates the calls supported by enhanced propagation scheduling for error checking and schedule monitoring.

PL/SQL (DBMS_AQADM Package): Example Code

```
CONNECT OE/OE;

/*  get averages
select avg_time, avg_number, avg_size from user_queue_schedules;

/*  get totals
select total_time, total_number, total_bytes from user_queue_schedules;

/*  get maximums for a window
select max_number, max_bytes from user_queue_schedules;

/*  get current status information of schedule
select process_name, session_id, instance, schedule_disabled
   from user_queue_schedules;

/*  get information about last and next execution
select last_run_date, last_run_time, next_run_date, next_run_time
   from user_queue_schedules;

/*  get last error information if any
select failures, last_error_msg, last_error_date, last_error_time
   from user_queue_schedules;
```

Visual Basic (OO4O): Example Code

This functionality is currently not available.

Java (JDBC): Example Code

No example is provided with this release.

Exception Handling During Propagation

When system errors such as a network failure occur, Oracle Streams AQ continues to attempt to propagate messages using an exponential backoff algorithm. In some situations that indicate application errors, Oracle Streams AQ marks messages as `UNDELIVERABLE` if a message propagation error occurs.

See Also: "Optimizing Propagation" on page 5-14

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. In such situations users must query the `DBA_SCHEDULES` view to determine the last error that occurred during propagation to a particular destination. The trace files in the `$ORACLE_HOME/log` directory can provide additional information about the error.

Scenario

In the BooksOnLine example, the `ES_bookedorders_que` in the Eastern Shipping Region is stopped intentionally using the stop_queue`()` call. After a short while the propagation schedule for `OE_bookedorders_que` displays an error indicating that the remote queue ES_bookedorders_que is disabled for enqueuing. When the `ES_bookedorders_que` is started using the `start_queue()` call, propagation to that queue resumes and there is no error message associated with schedule for `OE_bookedorders_que`.

PL/SQL (DBMS_AQADM Package): Example Code

```
/*  Intentionally stop the Eastern Shipping queue: */
connect BOLADM/BOLADM
EXECUTE DBMS_AQADM.STOP_QUEUE(queue_name => 'ES.ES_bookedorders_que');

/* Wait for some time before error shows up in dba_queue_schedules: */
EXECUTE dbms_lock.sleep(100);

/* This query returns an ORA-25207 enqueue failed error: */
SELECT qname, last_error_msg from dba_queue_schedules;
```

```
/* Start the Eastern Shipping queue: */
EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'ES.ES_bookedorders_que');

/* Wait for Propagation to resume for Eastern Shipping queue: */
EXECUTE dbms_lock.sleep(100);

/* This query indicates that there are no errors with propagation:
SELECT qname, last_error_msg from dba_queue_schedules;
```

Visual Basic (OO4O): Example Code

This functionality is handled by the database.

Java (JDBC): Example Code

No example is provided with this release.

Message Format Transformation During Propagation

At propagation time, a transformation can be specified when adding a rule-based subscriber to `OE_bookedorders_topic` for Western Shipping orders. The transformation is applied to the orders, transforming them to the `WS.order_typ_sh` type before propagating them to `WS_bookedorders_topic`.

PL/SQL (DBMS_AQADM Package): Example Code

```
declare
subscriber     sys.aq$_agent;
begin
  subscriber :=sys.aq$_agent('West_Shipping','WS.WS_bookedorders_topic',null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'OE.OE_bookedorders_topic',
       subscriber     => subscriber,
       rule           => 'tab.user_data.orderregion =''WESTERN''
                          AND tab.user_data.ordertype != ''RUSH''',
       transformation => 'OE.OE2WS');
end;
```

Visual Basic (OO4O): Example Code

No example is provided with this release.

Java (JDBC): Example Code

No example is provided with this release.

Propagation Using HTTP

In Oracle Database 10*g* and higher, you can set up Oracle Streams AQ propagation over HTTP and HTTPS (HTTP over SSL). HTTP propagation uses the Internet access infrastructure and requires that the Oracle Streams AQ 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.

Scenario

In the BooksOnLine example, messages in the `OE_bookedorders_que` are propagated to different shipping sites. For the purpose of this scenario, the Western Shipping application is running on another database, `'dest-db'` and we propagates to `WS_bookedorders_que`.

Propagation Setup

1. Deploy the Oracle Streams AQ Servlet.

   HTTP propagation depends on Internet access to the destination database. Create a class `AQPropServlet` that extends the AQxmlServlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.AQ.*;
import oracle.AQ.xml.*;
import java.sql.*;
import oracle.jms.*;
import javax.jms.*;
import java.io.*;
import oracle.jdbc.pool.*;

/* This is an Oracle Streams AQ Propagation Servlet. */
public class AQPropServlet extends oracle.AQ.xml.AQxmlServlet

/* getDBDrv - specify the database to which the servlet connects */
public AQxmlDataSource createAQDataSource() throws AQxmlException
{
  AQxmlDataSource  db_drv = null;
  db_drv = new AQxmlDataSource("aqadm", "aqadm", "dest-db", "dest-host",
      5521);
```

```
    return db_drv;
  }

  public void init()
  {
      try {
        AQxmlDataSource axds = this.createAQDataSource();
        setAQDataSource(axds) ;
        setSessionMaxInactiveTime(180) ;

      } catch (Exception e) {
         System.err.println("Error in init : " +e) ;
      }
  }
}
```

This servlet must connect to the destination database. The servlet must be deployed on the Web server in the path `aqserv/servlet`. In Oracle Database 10*g* and higher, the propagation servlet name and deployment path are fixed; that is, they must be `AQPropServlet` and `aqserv/servlet`, respectively.

Assume that the Web server host and port are `webdest.oracle.com` and `8081`, respectively.

2. Create the database link dba.

   - Specify HTTP as the protocol.
   - Specify the username and password that are used for authentication with the Web server/servlet runner as the host and port of the Web server running the Oracle Streams AQ servlet.

   For this example, the connect string of the database link should be as follows:

   ```
   (DESCRIPTION=(ADDRESS=(PROTOCOL=http)(HOST=webdest.oracle.com)(PORT=8081))
   ```

   If SSL is used, then specify HTTPS as the protocol in the connect string.

   Create the database link as follows:

   ```
   create public database link dba connect to john IDENTIFIED BY welcome
   using
   '(DESCRIPTION=(ADDRESS=(PROTOCOL=http)(HOST=webdest.oracle.com)(PORT=8081))'
   ;
   ```

Here `john` is the Oracle Streams AQ HTTP agent used to access the Oracle Streams AQ (propagation) servlet. `Welcome` is the password used to authenticate with the Web server.

3. Make sure that the Oracle Streams AQ HTTP agent, John, is authorized to perform Oracle Streams AQ operations. Do the following at the destination database.

   a. Register the Oracle Streams AQ agent.

      ```
      DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'John', enable_http => true);
      ```

   b. Map the Oracle Streams AQ agent to a database user.

      ```
      DBMS_AQADM.ENABLE_DB_ACCESS(agent_name =>'John', db_username =>'CBADM')'
      ```

4. Set up the remote subscription to `OE.OE_bookedorders_que`.

   ```
   EXECUTE DBMS_AQADM.ADD_SUBSCRIBER('OE.OE_bookedorders_que',
   aq$_agent(null, 'WS.WS_bookedorders_que', null));
   ```

5. Start propagation by calling `dbms_aqdm.schedule_propagation` at the source database.

   ```
   DBMS_AQADM.SCHEDULE_PROPAGATION('OE.OE_bookedorders_que', 'dba');
   ```

All other propagation administration APIs work the same for HTTP propagation. Use the propagation view, `DBA_QUEUE_SCHEDULES`, to check the propagation statistics for propagation schedules using HTTP.

Part IV

Oracle Streams AQ Administrative and Operational Interface

Part IV describes the Oracle Streams Advanced Queuing (AQ) administrative and operational interface.

This part contains the following chapters:

- Chapter 8, "Oracle Streams AQ Administrative Interface"
- Chapter 9, "Oracle Streams AQ Administrative Interface: Views"
- Chapter 10, "Oracle Streams AQ Operational Interface: Basic Operations"

8

Oracle Streams AQ Administrative Interface

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 Streams AQ Agents
- Adding an Alias to the LDAP Server
- Deleting an Alias from the LDAP Server

See Also:

- Chapter 4, "Oracle Streams AQ: Programmatic Environments" for a list of available functions in each programmatic environment
- *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

Purpose

Creates a **queue table** for messages of a predefined type.

Syntax

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table          IN      VARCHAR2,
   queue_payload_type   IN      VARCHAR2,
   [storage_clause      IN      VARCHAR2        DEFAULT NULL,]
   sort_list            IN      VARCHAR2        DEFAULT NULL,
   multiple_consumers   IN      BOOLEAN         DEFAULT FALSE,
   message_grouping     IN      BINARY_INTEGER  DEFAULT NONE,
   comment              IN      VARCHAR2        DEFAULT NULL,
   auto_commit          IN      BOOLEAN         DEFAULT TRUE,
   primary_instance     IN      BINARY_INTEGER  DEFAULT 0,
   secondary_instance   IN      BINARY_INTEGER  DEFAULT 0,
   compatible           IN      VARCHAR2        DEFAULT NULL,
   secure               IN      BOOLEAN         DEFAULT FALSE);
```

See Also:

http://otn.oracle.com/docs/products/aq/doc_library/ojms/index.html for information on Oracle Java Message Service

Usage Notes

To create a queue table, you must specify:

- 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 `"abc"."efg"` means the schema is `abc` and the name is `efg`.

- Payload type as RAW or an object type

  To specify the payload type as an object type, you must define the object type.

  CLOB, **BLOB**, and **BFILE** objects are valid in an Oracle Streams AQ **message**. You can propagate these object types using Oracle Streams AQ **propagation** with Oracle software since Oracle8*i* 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.

- Single-consumer or multiconsumer queue
- Message grouping as none (default), or transactional
- Primary instance and secondary instance

  You can specify and modify `primary_instance` and `secondary_instance` only in 8.1-compatible or higher mode. You cannot specify a secondary instance unless there is a primary instance.

- Compatible as 8.0, 8.1, or 10.0

  This parameter defaults to 8.0 if the database is in 8.0 compatible mode, 8.1 if the database is in 8.1 compatible mode, or 10.0 if the database is in 10.0 compatible mode.

- Secure as TRUE or FALSE

  This 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 perform queue operations on a secure queue, unless they are configured as secure queue users.

Further, you may optionally:

- Specify sort keys for **dequeue** ordering
- Specify the storage clause (only if you do not want to use the default tablespace)

  The `storage_clause` argument can take any text that can be used in a standard `CREATE TABLE storage_clause` argument.
- Add a table description
- Set auto-commit to true (default) or false

> **Note:** Auto-commit has been deprecated.

The sort type, if specified, can be one of the following:

- Enqueue time (default for sort time)
- Priority
- Enqueue time by priority
- Priority by enqueue time

The following objects are created at table creation time:

- `aq$_`*`queue_table_name`*`_e`, the default **exception queue** associated with the queue table
- `aq$`*`queue_table_name`*, a read-only view which is used by Oracle Streams AQ applications for querying **queue** data
- `aq$_`*`queue_table_name`*`_t`, an index for the queue monitor operations
- `aq$_`*`queue_table_name`*`_i`, an index or an **index-organized table** (IOT) in the case of multiple **consumer** queues for dequeue operations

For 8.1-compatible multiconsumer queue tables, the following additional objects are created:

- `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

> **See Also:** *Oracle Database Application Developer's Guide - Large Objects*

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.

Examples

PL/SQL (DBMS_AQADM Package): Creating a Queue Table

You must set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

Example 8–1 PL/SQL: Creating a Queue Table for Queues Containing Messages of Object Type

```
CREATE type aq.Message_typ as object (
   Subject                 VARCHAR2(30),
   Text                    VARCHAR2(80));

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table             => 'aq.ObjMsgs_qtab',
   Queue_payload_type      => 'aq.Message_typ');
```

Example 8–2 PL/SQL: Creating a Queue Table for Queues Containing Messages of RAW Type

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            => 'aq.RawMsgs_qtab',
   Queue_payload_type     => 'RAW');
```

Example 8–3 PL/SQL: Creating a Queue Table for Queues Containing Messages of XMLType

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table     => 'TS_orders_pr_mqtab',
        comment         => 'Overseas Shipping MultiConsumer Orders queue table',
        multiple_consumers  => TRUE,
        queue_payload_type  => 'SYS.XMLType',
        compatible          => '8.1');
```

Example 8–4 PL/SQL: Creating a Queue Table for Prioritized Messages

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            => 'aq.PriorityMsgs_qtab',
   Sort_list              => 'PRIORITY,ENQ_TIME',
   Queue_payload_type     => 'aq.Message_typ');
```

Example 8–5 PL/SQL: Creating a Queue Table for Multiple Consumers

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            => 'aq.MultiConsumerMsgs_qtab',
   Multiple_consumers     => TRUE,
   Queue_payload_type     => 'aq.Message_typ');
```

Example 8–6 PL/SQL: Creating a Queue Table for Multiple Consumers Compatible with 8.1

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            => 'aq.Multiconsumermsgs8_1qtab',
   Multiple_consumers     =>  TRUE,
   Compatible             => '8.1',
   Queue_payload_type     => 'aq.Message_typ');
```

Example 8–7 PL/SQL: Creating a Queue Table in a Specified Tablespace

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table        => 'aq.aq_tbsMsg_qtab',
        queue_payload_type => 'aq.Message_typ',
        storage_clause     => 'tablespace aq_tbs');
```

Example 8–8 PL/SQL: Creating a Queue Table with Freelists or Freelist Groups

```
BEGIN
DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table=> 'AQ_ADMIN.TEST',
queue_payload_type=> 'RAW',
storage_clause=> 'STORAGE (FREELISTS 4 FREELIST GROUPS 2)',
compatible => '8.1');
COMMIT;
END;
```

Altering a Queue Table

Purpose

Alters the existing properties of a queue table.

Syntax

```
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);
```

Usage Notes

To alter a queue table, you must name the queue table. You may optionally:

- Add a comment
- Specify the primary instance

  The primary instance is the instance number of the primary owner of the queue table.
- Specify the secondary instance

  The secondary instance is the instance number of the secondary owner of the queue table.

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue table is modified, then a corresponding LDAP entry is also altered.

Examples

Example 8–9 PL/SQL (DBMS_AQADM Package): Altering a Queue Table by Changing the Primary and Secondary Instances

```
EXECUTE DBMS_AQADM.ALTER_QUEUE_TABLE (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Primary_instance     => 3,
   Secondary_instance   => 2);
```

Example 8–10 PL/SQL (DBMS_AQADM Package): Altering a Queue Table by Changing the Comment

```
EXECUTE DBMS_AQADM.ALTER_QUEUE_TABLE (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Comment              => 'revised usage for queue table');
```

Example 8–11 PL/SQL (DBMS_AQADM Package): Altering a Queue Table by Changing the Comment and Using Nonrepudiation

```
EXECUTE DBMS_AQADM.ALTER_QUEUE_TABLE (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Comment              => 'revised usage for queue table');
   non_repudiation       => 'nonrepudiable_sender');
```

Dropping a Queue Table

Purpose

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 unless the `force` option is used, in which case these operations are accomplished automatically.

Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table       IN    VARCHAR2,
   force             IN    BOOLEAN DEFAULT FALSE,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Note: Parameter `auto_commit` is deprecated.

Usage Notes

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue table is dropped, then a corresponding LDAP entry is also dropped.

Examples

You must set up or drop data structures for certain examples to work.

Example 8–12 PL/SQL (DBMS_AQADM Package): Dropping a Queue Table

```
EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table        => 'aq.Objmsgs_qtab');
```

Example 8–13 PL/SQL (DBMS_AQADM Package): Dropping a Queue Table with Force Option

```
EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table        => 'aq.Objmsgs_qtab',
   force              => TRUE);
```

Purging a Queue Table

Purpose

Purges messages from a queue table.

Syntax

```
DBMS_AQADM.PURGE_QUEUE_TABLE(
   queue_table        IN   VARCHAR2,
   purge_condition    IN   VARCHAR2,
   purge_options      IN   aq$_purge_options_t);
```

Usage Notes

You can perform various purge operations on both single-consumer and multiconsumer queue tables for persistent queues. You can purge selected messages from the queue table by specifying additional parameters in the API call.

The purge condition must be in the format of a SQL `WHERE` clause, and it is case-sensitive. The condition is based on the columns of `aq$queue_table` view.

To purge all queues in a queue table, set `purge_condition` to either `NULL` (a bare null word, no quotes) or `''` (two single quotes).

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 *PL/SQL Packages and Types Reference* for more information on `DBMS_AQADM.PURGE_QUEUE_TABLE`

Examples

Example 8–14 Purging All Messages in Queue Table tkaqqtdef

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
begin
po.block := FALSE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => NULL,
     purge_options   => po);
end;
/
```

Example 8–15 Purging All Messages in Queue Table tkaqqtdef That Correspond to Queue q1def

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'queue = ''Q1DEF''',
     purge_options   => po);
end;
/
```

Example 8–16 Purging All Messages in Queue Table tkaqqtdef That Correspond to Queue q1def and Are in the PROCESSED State

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
```

```
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'queue = ''Q1DEF'' and msg_state = ''PROCESSED''',
     purge_options   => po);
end;
/
```

Example 8–17 Purging All Messages in Queue Table tkaqqtdef That Correspond to Queue q1def and Are Intended for Consumer PAYROLL_APP

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'queue = ''Q1DEF'' and consumer_name = ''PAYROLL_APP''',
     purge_options   => po);
end;
/
```

Example 8–18 Purging All Messages in Queue Table tkaqqtdef That Correspond to Sender Name PAYROLL_APP

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'sender_name = ''PAYROLL_APP''',
     purge_options   => po);
end;
/
```

Example 8–19 Purging All Messages in Queue Table tkaqqtdef Where tab.city Is BELMONT

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
```

```
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'tab.city = ''BELMONT''',
     purge_options   => po);
end;
/
```

Example 8–20 urging All Messages in Queue Table tkaqqtdef That Were Enqueued Before January 1, 2002

```
connect tkaqadmn/tkaqadmn
declare
po dbms_aqadm.aq$_purge_options_t;
begin
po.block := TRUE;
dbms_aqadm.purge_queue_table(
     queue_table     => 'tkaqqtdef',
     purge_condition => 'enq_time < ''01-JAN-2002''',
     purge_options   => po);
end;
/
```

Migrating a Queue Table

Purpose

Migrating a queue table from 8.0, 8.1, or 10.0 to 8.0, 8.1, or 10.0.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (
   queue_table   IN   VARCHAR2,
   compatible    IN   VARCHAR2);
```

Usage Notes

If a schema was created by an import of an export dump from a lower release or has Oracle Streams AQ 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.

Managing Queues

This section contains these topics:

- Creating a Queue
- Creating a Nonpersistent Queue
- Altering a Queue
- Dropping a Queue
- Starting a Queue
- Stopping a Queue

Creating a Queue

Purpose

Creates a queue in the specified queue table.

Syntax

```
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,
   comment             IN       VARCHAR2       DEFAULT NULL,
   auto_commit         IN       BOOLEAN        DEFAULT TRUE);
```

Note: Parameter `auto_commit` is deprecated.

Usage Notes

Mixed case (upper and lower case together) queue names and 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 `"abc"."efg"` means the schema is `abc` and the name is `efg`.

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.

Examples

You must set up or drop data structures for certain examples to work.

Example 8–21 PL/SQL: Creating a Queue Within a Queue Table for Messages of Object Type

```
/* Create a message type: */
CREATE type aq.Message_typ as object (
   Subject     VARCHAR2(30),
   Text        VARCHAR2(80));

/* Create a object type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table        => 'aq.ObjMsgs_qtab',
   Queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name         =>  'msg_queue',
   Queue_table        => 'aq.ObjMsgs_qtab');
```

Example 8–22 PL/SQL: Creating a Queue Within a Queue Table for Messages of RAW Type

```
/* Create a RAW type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table          => 'aq.RawMsgs_qtab',
   Queue_payload_type   => 'RAW');

/* Create queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name          => 'raw_msg_queue',
   Queue_table         => 'aq.RawMsgs_qtab');
```

Example 8–23   PL/SQL: Creating a Queue for Prioritized Messages

```
/* Create a queue table for prioritized messages: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table        => 'aq.PriorityMsgs_qtab',
   Sort_list          => 'PRIORITY,ENQ_TIME',
   Queue_payload_type => 'aq.Message_typ');
/* Create queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name         => 'priority_msg_queue',
   Queue_table        => 'aq.PriorityMsgs_qtab');
```

Example 8–24   PL/SQL: Creating a Queue Table and Queue for Multiple Consumers

```
/* Create a multiconsumer queue table: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table        => 'aq.MultiConsumerMsgs_qtab',
   Multiple_consumers => TRUE,
   Queue_payload_type => 'aq.Message_typ');

/* Create queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name         => 'MultiConsumerMsg_queue',
   Queue_table        => 'aq.MultiConsumerMsgs_qtab');
```

Example 8–25   PL/SQL: Creating a Queue Table and Queue to Demonstrate Propagation

```
/* Create queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name        => 'AnotherMsg_queue',
   queue_table       => 'aq.MultiConsumerMsgs_qtab');
```

Example 8–26   PL/SQL: Creating a Queue Table and Queue for Multiple Consumers Compatible with 8.1

```
/* Create a multiconsumer queue table compatible with Release 8.1: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table        => 'aq.MultiConsumerMsgs81_qtab',
   Multiple_consumers => TRUE,
   Compatible         => '8.1',
   Queue_payload_type => 'aq.Message_typ');
```

```
/* Create queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name         => 'MultiConsumerMsg81_queue',
   Queue_table        => 'aq.MultiConsumerMsgs81_qtab');
```

Creating a Nonpersistent Queue

Purpose

Creates a **nonpersistent** queue.

Syntax

```
DBMS_AQADM.CREATE_NP_QUEUE (
   queue_name            IN        VARCHAR2,
   multiple_consumers    IN        BOOLEAN  DEFAULT FALSE,
   comment               IN        VARCHAR2 DEFAULT NULL);
```

Usage Notes

The queue can be either single-consumer or multiconsumer. All queue names must be unique within a schema. The queues are created in a 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.

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.

See Also:

- "Registering for Notification" on page 10-39
- "Listening to One or More Queues" on page 10-17

Examples

Example 8–27 PL/SQL: Creating a Single-Consumer Nonpersistent Queue

```
EXECUTE DBMS_AQADM.CREATE_NP_QUEUE(
   Queue_name          => 'Singleconsumersmsg_npque',
   Multiple_consumers   => FALSE);
```

Example 8–28 PL/SQL: Creating a Multiconsumer Nonpersistent Queue

```
EXECUTE DBMS_AQADM.CREATE_NP_QUEUE(
   Queue_name          => 'Multiconsumersmsg_npque',
   Multiple_consumers   => TRUE);
```

Altering a Queue

Purpose

Alters existing properties of a queue.

Syntax

```
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,
   auto_commit       IN    BOOLEAN  DEFAULT TRUE,
   comment           IN    VARCHAR2 DEFAULT NULL);
```

Note: Parameter `auto_commit` is deprecated.

Usage Notes

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.

Examples

Example 8–29 PL/SQL (DBMS_AQADM): Altering a Queue

```
/* Change retention time, saving messages for 1 day after dequeuing: */
EXECUTE DBMS_AQADM.ALTER_QUEUE (
   queue_name        => 'aq.Anothermsg_queue',
   retention_time    => 86400);
```

Dropping a Queue

Purpose

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.

Syntax

```
DBMS_AQADM.DROP_QUEUE (
   queue_name        IN    VARCHAR2,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

> **Note:** Parameter `auto_commit` is deprecated.

Usage Notes

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue is dropped, then a corresponding LDAP entry is also dropped.

You must stop the queue before dropping it. A queue can be dropped only after it has been successfully stopped for enqueuing and dequeuing.

Examples

Example 8–30 PL/SQL: Dropping a Standard Queue

```
/* Stop the queue: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   Queue_name        => 'aq.Msg_queue');

/* Drop the queue: */
```

```
EXECUTE DBMS_AQADM.DROP_QUEUE (
   Queue_name         => 'aq.Msg_queue');
```

Example 8–31 PL/SQL: Dropping a Nonpersistent Queue

```
/* Stop the queue: */
EXECUTE DBMS_AQADM.DROP_QUEUE (
   Queue_name         => 'Nonpersistent_singleconsumerq1');

/* Drop the queue: */
EXECUTE DBMS_AQADM.DROP_QUEUE (
   Queue_name => 'Nonpersistent_multiconsumerq1');
```

Starting a Queue

Purpose

Enables the specified queue for enqueuing or dequeuing.

Usage Notes

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.

Syntax

```
DBMS_AQADM.START_QUEUE (
   queue_name      IN     VARCHAR2,
   enqueue         IN     BOOLEAN DEFAULT TRUE,
   dequeue         IN     BOOLEAN DEFAULT TRUE);
```

Examples

Example 8–32 PL/SQL (DBMS_AQADM Package): Starting a Queue with Both Enqueue and Dequeue Enabled

```
EXECUTE DBMS_AQADM.START_QUEUE (
   queue_name         => 'Msg_queue');
```

Example 8–33 PL/SQL (DBMS_AQADM Package): Starting a Previously Stopped Queue for Dequeue Only

```
EXECUTE DBMS_AQADM.START_QUEUE (
   queue_name          => 'aq.msg_queue',
   dequeue             => TRUE,
   enqueue             => FALSE);
```

Stopping a Queue

Purpose

Disables enqueuing, dequeuing, or both on the specified queue.

Syntax

```
DBMS_AQADM.STOP_QUEUE (
   queue_name       IN   VARCHAR2,
   enqueue          IN   BOOLEAN DEFAULT TRUE,
   dequeue          IN   BOOLEAN DEFAULT TRUE,
   wait             IN   BOOLEAN DEFAULT TRUE);
```

Usage Notes

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.

Examples

Example 8–34 PL/SQL (DBMS_AQADM): Stopping a Queue

```
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name         => 'aq.Msg_queue');
```

Managing Transformations

This section contains these topics:

- Creating a Transformation
- Modifying a Transformation
- Dropping a Transformation

Creating a Transformation

Purpose

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`.

Syntax

```
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));
```

Usage Notes

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. The transformation cannot write the database state (that is, perform **DML** operations) or commit or rollback the current transaction.

Examples

Example 8–35 PL/SQL (DBMS_AQADM): Creating a Transformation

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION(schema => 'scott',
    name           => 'test_transf', from_schema => 'scott',
    from_type      => 'type1', to_schema => 'scott',
    to_type        => 'type2',
    transformation => 'scott.trans_func(source.user_data)');
```

Or you can do the following:

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION(schema => 'scott',
    name           => 'test_transf',
    from_schema    => 'scott',
    from_type      => 'type1,
```

```
    to_schema      => 'scott',
    to_type        => 'type2',
    transformation => 'scott.type2(source.user_data.attr2,
        source.user_data.attr1)');
```

Modifying a Transformation

Purpose

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`.

Syntax

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (
    schema            VARCHAR2(30),
    name              VARCHAR2(30),
    attribute_number  INTEGER,
    transformation    VARCHAR2(4000));
```

Usage Notes

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

Purpose

Drops a transformation.

Syntax

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
    schema      VARCHAR2(30),
    name        VARCHAR2(30));
```

Usage Notes

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 System Oracle Streams AQ Privileges
- Revoking Oracle Streams AQ System Privileges
- Granting Queue Privileges
- Revoking Queue Privileges

Granting System Oracle Streams AQ Privileges

Purpose

Grants Oracle Streams AQ 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.

Syntax

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
   privilege          IN    VARCHAR2,
   grantee            IN    VARCHAR2,
   admin_option       IN    BOOLEAN := FALSE);
```

Usage Notes

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

You must set up the following data structures for this example to work:

```
CONNECT system/manager;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
```

Example 8–36 PL/SQL (DBMS_AQADM): Granting System Privilege

```
CONNECT aqadm/aqadm;
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege          =>    'ENQUEUE_ANY',
   grantee            =>    'Jones',
   admin_option       =>     FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege          =>     'DEQUEUE_ANY',
   grantee            =>     'Jones',
   admin_option       =>      FALSE);
```

Revoking Oracle Streams AQ System Privileges

Purpose

Revokes Oracle Streams AQ 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.

Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (
   privilege         IN   VARCHAR2,
   grantee           IN   VARCHAR2);
```

Usage Notes

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.

Examples

Example 8–37 PL/SQL (DBMS_AQADM): Revoking System Privilege

```
CONNECT system/manager;
   EXECUTE DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(privilege=>'DEQUEUE_ANY',
                              grantee=>'Jones');
```

Granting Queue Privileges

Purpose

Grants privileges on a queue to users and roles. The privileges are ENQUEUE or DEQUEUE. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Syntax

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege        IN    VARCHAR2,
   queue_name       IN    VARCHAR2,
   grantee          IN    VARCHAR2,
   grant_option     IN    BOOLEAN := FALSE);
```

Examples

Example 8–38 PL/SQL (DBMS_AQADM): Granting Queue Privilege

```
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege     =>     'ALL',
   queue_name    =>     'aq.multiconsumermsg81_queue',
   grantee       =>     'Jones',
   grant_option  =>      TRUE);
```

Revoking Queue Privileges

Purpose

Revokes privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE.

Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege          IN      VARCHAR2,
   queue_name         IN      VARCHAR2,
   grantee            IN      VARCHAR2);
```

Usage Notes

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 8–39.

Examples

Example 8–39 PL/SQL (DBMS_AQADM): Revoking Dequeue Privilege

```
CONNECT scott/tiger;
EXECUTE DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(
   privilege     =>     'DEQUEUE',
   queue_name    =>     'scott.ScottMsgs_queue',
   grantee       =>     'Jones');
```

Managing Subscribers

This section contains these topics:

- Adding a Subscriber
- Altering a Subscriber
- Removing a Subscriber

Adding a Subscriber

Purpose

Adds a default **subscriber** to a queue.

Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2 DEFAULT NULL,
   transformation IN    VARCHAR2 DEFAULT NULL);
```

Usage Notes

A program 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. 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'''
```

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:** *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()`.

> **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 transactions that have enqueued or dequeued messages commit and release the resources they hold.

Examples

Example 8–40 PL/SQL (DBMS_AQADM): Adding a Subscriber

```
/* Anonymous PL/SQL block for adding a subscriber at a designated queue in a
designated schema at a database link: */
DECLARE
   subscriber          sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent ('subscriber1', 'aq2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name         => 'aq.multi_queue',
      subscriber         =>  subscriber);
 END;
```

Example 8–41 PL/SQL (DBMS_AQADM): Adding a Subscriber with a Rule

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber2', 'aq2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name =>  'aq.multi_queue',
      subscriber =>   subscriber,
      rule       =>  'priority < 2');
END;
```

Example 8–42 PL/SQL: Adding a Subscriber and Specify a Transformation

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber2', 'aq2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name     => 'aq.multi_queue',
      subscriber     =>  subscriber,
      transformation => 'AQ.msg_map');
/* Where the transformation was created as */
EXECUTE DBMS_TRANSFORM.CREATE_TRANSFORMATION
 ( schema => 'AQ',
   name => 'msg_map',
   from_schema => 'AQ',
   from_type => 'purchase_order1',
   to_schema => 'AQ',
   to_type => 'purchase_order2',
```

```
   transformation => 'AQ.transform_PO(source.user_data)');
END;
```

Altering a Subscriber

Purpose

Alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

Syntax

```
DBMS_AQADM.ALTER_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2
   transformation IN    VARCHAR2);
```

Usage Notes

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.

Examples

You must set up the following data structures for the examples in this section to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table         => 'aq.multi_qtab',
   multiple_consumers  => TRUE,
   queue_payload_type  => 'aq.message_typ',
   compatible          => '8.1.5');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   queue_name          =>  'multi_queue',
   queue_table         => 'aq.multi_qtab');
```

Example 8–43   PL/SQL: Altering a Subscriber Rule

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('SUBSCRIBER1', 'aq2.msg_queue2@london', null);
```

```
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name => 'aq.msg_queue',
      subscriber =>  subscriber,
      rule       => 'priority < 2');
END;

/* Change rule for subscriber: */
DECLARE
   subscriber        sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('SUBSCRIBER1', 'aq2.msg_queue2@london', null);
   DBMS_AQADM.ALTER_SUBSCRIBER(
      queue_name => 'aq.msg_queue',
      subscriber =>  subscriber,
      rule       => 'priority = 1');
END;
```

Example 8–44   PL/SQL: Altering a Subscriber Transformation

```
EXECUTE DBMS_AQADM.ADD_SUBSCRIBER
   ('aq.msg_queue',
     aq$_agent('subscriber1',
        'aq2.msg_queue2@london',
            null),
'AQ.MSG_MAP1');

/* Alter the subscriber*/
EXECUTE DBMS_AQADM.ALTER_SUBSCRIBER
   ('aq.msg_queue',
     aq$_agent ('subscriber1',
      'aq2.msg_queue2@london',
                null),
       'AQ.MSG.MAP2');
```

Removing a Subscriber

Purpose

Removes a default subscriber from a queue.

Syntax

```
DBMS_AQADM.REMOVE_SUBSCRIBER (
   queue_name         IN         VARCHAR2,
   subscriber         IN         sys.aq$_agent);
```

Usage Notes

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.

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 transactions that have enqueued or dequeued messages commit and release the resources they hold.

Examples

Example 8–45 PL/SQL (DBMS_AQADM): Removing Subscriber

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber1','aq2.msg_queue2', NULL);
   DBMS_AQADM.REMOVE_SUBSCRIBER(
      queue_name => 'aq.multi_queue',
      subscriber => subscriber);
END;
```

Managing Propagations

This section contains these topics:

- Scheduling a Queue Propagation
- Unscheduling a Queue Propagation
- Verifying Propagation Queue Type
- Altering a Propagation Schedule
- Enabling a Propagation Schedule
- Disabling a Propagation Schedule

Scheduling a Queue Propagation

Purpose

Schedules propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
   queue_name       IN    VARCHAR2,
   destination      IN    VARCHAR2 DEFAULT NULL,
   start_time       IN    DATE     DEFAULT SYSDATE,
   duration         IN    NUMBER   DEFAULT NULL,
   next_time        IN    VARCHAR2 DEFAULT NULL,
   latency          IN    NUMBER   DEFAULT 60);
```

Usage Notes

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.

See Also: Chapter 17, "Internet Access to Oracle Streams AQ"

Examples

You must set up the following data structures for the examples in this section to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table        => 'aq.objmsgs_qtab',
   queue_payload_type => 'aq.message_typ',
   multiple_consumers => TRUE);
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   queue_name         => 'aq.q1def',
   queue_table        => 'aq.objmsgs_qtab');
```

Example 8–46 PL/SQL: Scheduling a Propagation from a Queue to other Queues in the Same Database

```
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
   Queue_name    =>    'aq.q1def');
```

Example 8–47 PL/SQL: Scheduling a Propagation from a Queue to other Queues in Another Database

```
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
   Queue_name    =>    'aq.q1def',
   Destination   =>    'another_db.world');
```

Unscheduling a Queue Propagation

Purpose

Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link.

Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
   queue_name     IN   VARCHAR2,
   destination    IN   VARCHAR2 DEFAULT NULL);
```

Examples

Example 8–48 PL/SQL: Unscheduling a Propagation from Queue to Other Queues in the Same Database

```
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(queue_name => 'aq.q1def');
```

Example 8–49 PL/SQL: Unscheduling a Propagation from a Queue to other Queues in Another Database

```
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   Queue_name    =>   'aq.q1def',
   Destination   =>   'another_db.world');
```

Verifying Propagation Queue Type

Purpose

Verifies that the source and destination queues have identical types. The result of the verification is stored in `sys.aq$_Message_types tables`, overwriting all previous output of this command.

Syntax

```
DBMS_AQADM.VERIFY_QUEUE_TYPES (
   src_queue_name    IN    VARCHAR2,
   dest_queue_name   IN    VARCHAR2,
   destination       IN    VARCHAR2 DEFAULT NULL,
   rc                OUT   BINARY_INTEGER);
```

Usage Notes

Verify that the source and destination queues have the same type. The function has the side effect of inserting/updating the entry for the source and destination queues in the dictionary table `AQ$_MESSAGE_TYPES`.

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: The `sys.aq$_message_types` table can have multiple entries for the same source queue, destination queue, and database link, but with different transformations.

Examples

You must set up the following data structures for this example to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   queue_name         => 'aq.q2def',
   queue_table        => 'aq.objmsgs_qtab');
```

Example 8–50 PL/SQL (DBMS_AQADM): Verifying a Queue Type

```
/*  Verify that the source and destination queues have the same type. */
DECLARE
rc      BINARY_INTEGER;
BEGIN
/* Verify that the queues aquser.q1def and aquser.q2def in the local database
   have the same payload type */
   DBMS_AQADM.VERIFY_QUEUE_TYPES(
      src_queue_name  => 'aq.q1def',
      dest_queue_name => 'aq.q2def',
      rc              => rc);
   DBMS_OUTPUT.PUT_LINE(rc);
END;
```

Altering a Propagation Schedule

Purpose

Alters parameters for a propagation schedule.

Syntax

```
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);
```

Examples

Example 8–51 PL/SQL: Altering a Propagation Schedule from a Queue to Other Queues in the Same Database

```
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   Queue_name    =>    'aq.q1def',
   Duration      =>    '2000',
   Next_time     =>    'SYSDATE + 3600/86400',
   Latency       =>    '32');
```

Example 8–52 PL/SQL: Altering a Propagation Schedule from a Queue to Other Queues in Another Database

```
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   Queue_name    =>    'aq.q1def',
   Destination   =>    'another_db.world',
   Duration      =>    '2000',
   Next_time     =>    'SYSDATE + 3600/86400',
   Latency       =>    '32');
```

Enabling a Propagation Schedule

Purpose

Enables a previously disabled propagation schedule.

Syntax

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
   queue_name      IN     VARCHAR2,
   destination     IN     VARCHAR2 DEFAULT NULL);
```

Examples

Example 8–53   PL/SQL: Enabling Propagation from a Queue to Other Queues in the Same Database

```
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   Queue_name   =>   'aq.q1def');
```

Example 8–54   PL/SQL: Enabling Propagation from a Queue to Queues in Another Database

```
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   Queue_name    =>    'aq.q1def',
   Destination   =>    'another_db.world');
```

Disabling a Propagation Schedule

Purpose

Disables a previously enabled propagation schedule.

Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
   queue_name      IN     VARCHAR2,
   destination     IN     VARCHAR2 DEFAULT NULL);
```

Examples

Example 8–55   PL/SQL: Disabling Propagation from a Queue to Other Queues in the Same Database

```
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   Queue_name   =>   'aq.q1def');
```

Example 8–56 PL/SQL: Disabling Propagation from a Queue to Queues in Another Database

```
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   Queue_name     =>    'aq.q1def',
   Destination    =>    'another_db.world');
```

Managing Oracle Streams AQ Agents

This section contains these topics:

- Creating an Oracle Streams AQ Agent
- Altering an Oracle Streams AQ Agent
- Dropping an Oracle Streams AQ Agent
- Enabling Database Access
- Disabling Database Access

Creating an Oracle Streams AQ Agent

Purpose

Registers an agent for Oracle Streams AQ Internet access using HTTP protocols.

Syntax

```
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 )
```

Usage Notes

The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle Streams AQ 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 Streams AQ Agent

Purpose

Alters an agent registered for Oracle Streams AQ Internet access.

Syntax

```
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 )
```

Usage Notes

When an Oracle Streams AQ 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 Streams AQ Agent

Purpose

Drops an agent that was previously registered for Oracle Streams AQ Internet access.

Syntax

```
DBMS_AQADM.DROP_AQ_AGENT (
  agent_name     IN VARCHAR2)
```

Usage Notes

When an Oracle Streams AQ 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

Purpose

Grants an Oracle Streams AQ Internet agent the privileges of a specific database user. The agent should have been previously created using the CREATE_AQ_AGENT procedure.

Syntax

```
DBMS_AQADM.ENABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Streams AQ Internet agents and the names of the database users whose privileges are granted to them.

Disabling Database Access

Purpose

Revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent. The agent should have been previously granted those privileges using the ENABLE_DB_ACCESS procedure.

Syntax

```
DBMS_AQADM.DISABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Adding an Alias to the LDAP Server

Purpose

Adds an alias to the LDAP server.

Syntax

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP(
   alias          IN VARCHAR2,
   obj_location   IN VARCHAR2);
```

> **See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

Usage Notes

This call takes the name of an alias and the distinguished name of an Oracle Streams AQ object in LDAP, and creates the alias that points to the Oracle Streams AQ 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 **connection factory**.

Deleting an Alias from the LDAP Server

Purpose

Removes an alias from the LDAP server.

Syntax

```
DBMS_AQ.DEL_ALIAS_FROM_LDAP(
   alias IN VARCHAR2);
```

Usage Notes

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.

9

Oracle Streams AQ Administrative Interface: Views

This chapter describes the Oracle Streams Advanced Queuing (AQ) administrative interface views.

This chapter contains these topics:

- All Queue Tables in Database View
- User Queue Tables View
- All Queues in Database View
- All Propagation Schedules View
- Queues for Which User Has Any Privilege View
- Queues for Which User Has Queue Privilege View
- Messages in Queue Table View
- Queue Tables in User Schema View
- Queues In User Schema View
- Propagation Schedules in User Schema View
- Queue Subscribers View
- Queue Subscribers and Their Rules View
- Number of Messages in Different States for the Whole Database View
- Number of Messages in Different States for Specific Instances View
- Oracle Streams AQ Agents Registered for Internet Access View
- All Transformations View

- All Transformation Functions View
- User Transformations View
- User Transformation Functions View

All Queue Tables in Database View

Name of View

DBA_QUEUE_TABLES

Purpose

Describes the names and types of all queue tables created in the database.

Table 9–1 DBA_QUEUE_TABLES View

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	-	Queue table **schema**
QUEUE_TABLE	VARCHAR2(30)	-	Queue table name
TYPE	VARCHAR2(7)	-	Payload type
OBJECT_TYPE	VARCHAR2(61)	-	Name of **object type**, if any
SORT_ORDER	VARCHAR2(22)	-	User-specified sort order
RECIPIENTS	VARCHAR2(8)	-	SINGLE or MULTIPLE
MESSAGE_GROUPING	VARCHAR2(13)	-	NONE or TRANSACTIONAL
COMPATIBLE	VARCHAR2(5)	-	Indicates the lowest version with which the **queue table** is compatible
PRIMARY_INSTANCE	NUMBER	-	Indicates which instance is the primary owner of the queue table, or no primary owner if 0
SECONDARY_INSTANCE	NUMBER	-	Indicates which instance is the secondary owner of the queue table. This instance becomes the owner of the queue table if the primary owner is not up. A value of 0 indicates that there is no secondary owner.

Table 9–1 (Cont.) (Cont.) DBA_QUEUE_TABLES View

Column	Datatype	NULL	Description
OWNER_INSTANCE	NUMBER	-	Indicates which instance currently owns the queue table
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue table
SECURE	VARCHAR2(3)	-	Indicates whether this queue table is secure (YES) or not (NO). 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 perform queue operations on a secure queue, unless they are configured as secure queue users.

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

User Queue Tables View

Name of View

ALL_QUEUE_TABLES

Purpose

Describes queue tables accessible to a user.

Table 9–2 ALL_QUEUE_TABLES View

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	-	Owner of the queue table
QUEUE_TABLE	VARCHAR2(30)	-	Queue table name
TYPE	VARCHAR2(7)	-	Payload type
OBJECT_TYPE	VARCHAR2(61)	-	Name of object type, if any
SORT_ORDER	VARCHAR2(22)	-	User-specified sort order
RECIPIENTS	VARCHAR2(8)	-	SINGLE or MULTIPLE
MESSAGE_GROUPING	VARCHAR2(13)	-	NONE or TRANSACTIONAL

Table 9–2 (Cont.) ALL_QUEUE_TABLES View

Column	Datatype	NULL	Description
COMPATIBLE	VARCHAR2(5)	-	Indicates the lowest version with which the queue table is compatible
PRIMARY_INSTANCE	NUMBER	-	Indicates which instance is the primary owner of the queue table, or no primary owner if 0
SECONDARY_INSTANCE	NUMBER	-	Indicates which instance is the secondary owner of the queue table. This instance becomes the owner of the queue table if the primary owner is not up. A value of 0 indicates that there is no secondary owner.
OWNER_INSTANCE	NUMBER	-	Indicates which instance currently owns the queue table
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue table
SECURE	VARCHAR2(3)	-	Indicates whether this queue table is secure (YES) or not (NO). 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 perform queue operations on a secure queue, unless they are configured as secure queue users.

All Queues in Database View

Name of View

DBA_QUEUES

Purpose

Specifies operational characteristics for individual queues. The DBA_QUEUES view displays these characteristics for every **queue** in a database.

Table 9–3 DBA_QUEUES View

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Queue schema name
NAME	VARCHAR2(30)	NOT NULL	Queue name
QUEUE_TABLE	VARCHAR2(30)	NOT NULL	Queue table where this queue resides
QID	NUMBER	NOT NULL	Unique queue identifier
QUEUE_TYPE	VARCHAR2(20)	-	Queue type
MAX_RETRIES	NUMBER	-	Number of **dequeue** attempts allowed
RETRY_DELAY	NUMBER	-	Number of seconds before retry can be attempted
ENQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO
DEQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO
RETENTION	VARCHAR2(40)	-	Number of seconds **message** is retained after dequeue
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue

Note: A message is moved to an **exception queue** if RETRY_COUNT is greater than MAX_RETRIES. 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.

All Propagation Schedules View

Name of View

DBA_QUEUE_SCHEDULES

Purpose

Describes the current schedules for propagating messages.

Table 9–4 DBA_QUEUE_SCHEDULES View

Column	Datatype	NULL	Description
SCHEMA	VARCHAR2(30)	NOT NULL	Schema name for the source queue
QNAME	VARCHAR2(30)	NOT NULL	Source queue name
DESTINATION	VARCHAR2(128)	NOT NULL	Destination name, currently limited to be a database link name
START_DATE	DATE	-	Date to start **propagation** in the default date format
START_TIME	VARCHAR2(8)	-	Time of day to start propagation in HH:MI:SS format
PROPAGATION_WINDOW	NUMBER	-	Duration in seconds for the propagation window
NEXT_TIME	VARCHAR2(200)	-	Function to compute the start of the next propagation window
LATENCY	NUMBER	-	Maximum wait time to propagate a message during the propagation window
SCHEDULE_DISABLED	VARCHAR(1)	-	N if enabled; Y if disabled (schedule will not be executed)
PROCESS_NAME	VARCHAR2(8)	-	Name of **Jnnn** background process executing this schedule; NULL if not currently executing
SESSION_ID	VARCHAR2(82)	-	Session ID (SID, SERIAL#) of the job executing this schedule; NULL if not currently executing
INSTANCE	NUMBER	-	Real Application Clusters instance number executing this schedule
LAST_RUN_DATE	DATE	-	Date of the last successful execution
LAST_RUN_TIME	VARCHAR2(8)	-	Time of the last successful execution in HH:MI:SS format
CURRENT_START_DATE	DATE	-	Date the current window of this schedule was started
CURRENT_START_TIME	VARCHAR2(8)	-	Time the current window of this schedule was started in HH:MI:SS format
NEXT_RUN_DATE	DATE	-	Date the next window of this schedule will be started
NEXT_RUN_TIME	VARCHAR2(8)	-	Time the next window of this schedule will be started in HH:MI:SS format
TOTAL_TIME	NUMBER	-	Total time in seconds spent in propagating messages from the schedule

Table 9–4 (Cont.) DBA_QUEUE_SCHEDULES View

Column	Datatype	NULL	Description
TOTAL_NUMBER	NUMBER	-	Total number of messages propagated in this schedule
TOTAL_BYTES	NUMBER	-	Total number of bytes propagated in this schedule
MAX_NUMBER	NUMBER	-	Maximum number of messages propagated in a propagation window
MAX_BYTES	NUMBER	-	Maximum number of bytes propagated in a propagation window
AVG_NUMBER	NUMBER	-	Average number of messages propagated in a propagation window
AVG_SIZE	NUMBER	-	Average size of propagated messages in bytes
AVG_TIME	NUMBER	-	Average time to propagate a message in seconds
FAILURES	NUMBER	-	Number of times execution failed. If it reaches 16, then the schedule is disabled.
LAST_ERROR_DATE	DATE	-	Date of the last unsuccessful execution
LAST_ERROR_TIME	VARCHAR2(8)	-	Time of the last unsuccessful execution in HH:MI:SS format
LAST_ERROR_MSG	VARCHAR2(4000)	-	Error number and error message text of the last unsuccessful execution

Queues for Which User Has Any Privilege View

Name of View

ALL_QUEUES

Purpose

Describes all queues accessible to the user.

Table 9–5 ALL_QUEUES View

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue
NAME	VARCHAR2(30)	NOT NULL	Name of the queue
QUEUE_TABLE	VARCHAR2(30)	NOT NULL	Queue table name
QID	NUMBER	NOT NULL	Unique queue identifier
QUEUE_TYPE	VARCHAR2(15)	-	Queue type
MAX_RETRIES	NUMBER	-	Number of dequeue attempts allowed
RETRY_DELAY	NUMBER	-	Number of seconds before retry can be attempted
ENQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO
DEQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO
RETENTION	VARCHAR2(40)	-	Number of seconds message is retained after dequeue
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue

Note: A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. 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.

Queues for Which User Has Queue Privilege View

Name of View

QUEUE_PRIVILEGES

Purpose

Describes queues for which the user is the grantor, or grantee, or owner, or an enabled role or the queue is granted to PUBLIC.

Table 9–6 QUEUE_PRIVILEGES View

Column	Datatype	NULL	Description
GRANTEE	VARCHAR2(30)	NOT NULL	Name of the user to whom access was granted
OWNER	VARCHAR2(30)	NOT NULL	Owner of the queue
NAME	VARCHAR2(30)	NOT NULL	Name of the queue
GRANTOR	VARCHAR2(30)	NOT NULL	Name of the user who performed the grant
ENQUEUE_PRIVILEGE	NUMBER	-	Permission to **enqueue** to queue (1 if granted, 0 if not)
DEQUEUE_PRIVILEGE	NUMBER	-	Permission to dequeue from queue (1 if granted, 0 if not)

Messages in Queue Table View

Name of View

AQ$*Queue_Table_Name*

Purpose

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.

Beginning with Oracle Database 10*g*, AQ$*Queue_Table_Name* includes buffered messages. For buffered messages, the value of MSG_STATE is one of the following:

- SPILLED
- IN MEMORY
- DEFERRED
- DEFERRED SPILLED

Table 9–7 AQ$Queue_Table_Name View

Column	Datatype	NULL	Description
QUEUE	VARCHAR2(30)	-	Queue name
MSG_ID	RAW(16)	NOT NULL	Unique identifier of the message
CORR_ID	VARCHAR2(128)	-	User-provided correlation identifier
MSG_PRIORITY	NUMBER	-	Message priority
MSG_STATE	VARCHAR2(16)	-	Message state
DELAY	DATE	-	Number of seconds the message is delayed
DELAY_TIMESTAMP	TIMESTAMP	-	Number of seconds the message is delayed
EXPIRATION	NUMBER	-	Number of seconds in which the message expires after being READY
ENQ_TIME	DATE	-	Enqueue time
ENQ_TIMESTAMP	TIMESTAMP	-	Enqueue time
ENQ_USER_ID (8.0.4 or 8.1.3 queue tables)	NUMBER	-	Enqueue user ID
ENQ_USER_ID (10.1 queue tables)	VARCHAR2(30)	-	Enqueue user ID
ENQ_TXN_ID	VARCHAR2(30)	-	Enqueue transaction ID
DEQ_TIME	DATE	-	Dequeue time
DEQ_TIMESTAMP	TIMESTAMP	-	Dequeue time
DEQ_USER_ID (8.0.4 or 8.1.3 queue tables)	NUMBER	-	Dequeue user ID
DEQ_USER_ID (10.1 queue tables)	VARCHAR2(30)	-	Dequeue user ID
DEQ_TXN_ID	VARCHAR2(30)	-	Dequeue transaction ID
RETRY_COUNT	NUMBER	-	Number of retries
EXCEPTION_QUEUE_OWNER	VARCHAR2(30)	-	Exception queue schema
EXCEPTION_QUEUE	VARCHAR2(30)	-	Exception queue name
USER_DATA	-	-	User data

Table 9–7 (Cont.) AQ$Queue_Table_Name View

Column	Datatype	NULL	Description
SENDER_NAME	VARCHAR2(30)	-	Name of the agent enqueuing the message (valid only for 8.1-compatible queue tables)
SENDER_ADDRESS	VARCHAR2(1024)	-	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.
SENDER_PROTOCOL	NUMBER	-	Protocol for sender address (reserved for future use and valid only for 8.1-compatible queue tables)
ORIGINAL_MSGID	RAW(16)	-	Message ID of the message in the source queue (valid only for 8.1-compatible queue tables)
CONSUMER_NAME	VARCHAR2(30)	-	Name of the agent receiving the message (valid only for 8.1-compatible multiconsumer queue tables)
ADDRESS	VARCHAR2(1024)	-	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)
PROTOCOL	NUMBER	-	Protocol for address of receiving agent (valid only for 8.1-compatible queue tables)
PROPAGATED_MSGID	RAW(16)	-	Message ID of the message in the queue of the receiving agent (valid only for 8.1-compatible queue tables)
ORIGINAL_QUEUE_NAME	VARCHAR2(30)	-	Name of the queue the message came from
ORIGINAL_QUEUE_OWNER	VARCHAR2(30)	-	Owner of the queue the message came from
EXPIRATION_REASON	VARCHAR2(19)	-	Reason the message came into exception queue. Possible values are TIME_EXPIRATION (message expired after the specified expired time), MAX_RETRY_EXCEEDED (maximum retry count exceeded), and PROPAGATION_FAILURE (message became undeliverable during propagation).

Note: A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. 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.

Queue Tables in User Schema View

Name of View

USER_QUEUE_TABLES

Syntax

This 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.

Table 9–8 USER_QUEUE_TABLES View

Column	Datatype	NULL	Description
QUEUE_TABLE	VARCHAR2(30)	-	Queue table name
TYPE	VARCHAR2(7)	-	Payload type
OBJECT_TYPE	VARCHAR2(61)	-	Name of object type, if any
SORT_ORDER	VARCHAR2(22)	-	User-specified sort order
RECIPIENTS	VARCHAR2(8)	-	SINGLE or MULTIPLE
MESSAGE_GROUPING	VARCHAR2(13)	-	NONE or TRANSACTIONAL
COMPATIBLE	VARCHAR2(5)	-	Indicates the lowest version with which the queue table is compatible
PRIMARY_INSTANCE	NUMBER	-	Indicates which instance is the primary owner of the queue table, or no primary owner if 0
SECONDARY_INSTANCE	NUMBER	-	Indicates which instance is the secondary owner of the queue table. This instance becomes the owner of the queue table if the primary owner is not up. A value of 0 indicates that there is no secondary owner.

Table 9–8 (Cont.) USER_QUEUE_TABLES View

Column	Datatype	NULL	Description
OWNER_INSTANCE	NUMBER	-	Indicates which instance currently owns the queue table
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue table
SECURE	VARCHAR2(3)	-	Indicates whether this queue table is secure (YES) or not (NO). 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 perform queue operations on a secure queue, unless they are configured as secure queue users.

Queues In User Schema View

Name of View

USER_QUEUES

Purpose

This view is the same as DBA_QUEUES with the exception that it only shows queues in the user's schema.

Table 9–9 USER_QUEUES View

Column	Datatype	NULL	Description
NAME	VARCHAR2(30)	NOT NULL	Queue name
QUEUE_TABLE	VARCHAR2(30)	NOT NULL	Queue table where this queue resides
QID	NUMBER	NOT NULL	Unique queue identifier
QUEUE_TYPE	VARCHAR2(20)	-	Queue type
MAX_RETRIES	NUMBER	-	Number of dequeue attempts allowed
RETRY_DELAY	NUMBER	-	Number of seconds before retry can be attempted
ENQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO

Table 9–9 (Cont.) USER_QUEUES View

Column	Datatype	NULL	Description
DEQUEUE_ENABLED	VARCHAR2(7)	-	YES or NO
RETENTION	VARCHAR2(40)	-	Number of seconds message is retained after dequeue
USER_COMMENT	VARCHAR2(50)	-	User comment for the queue

Note: A message is moved to an exception queue if RETRY_COUNT is greater than MAX_RETRIES. 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.

Propagation Schedules in User Schema View

Name of View

USER_QUEUE_SCHEDULES

Purpose

This view is the same as DBA_QUEUE_SCHEDULES with the exception that it only shows queue schedules in the user's schema.

Table 9–10 DBA_QUEUE_SCHEDULES View

Column	Datatype	NULL	Description
QNAME	VARCHAR2(30)	NOT NULL	Source queue name
DESTINATION	VARCHAR2(128)	NOT NULL	Destination name, currently limited to be a database link name
START_DATE	DATE	-	Date to start propagation in the default date format
START_TIME	VARCHAR2(8)	-	Time of day to start propagation in HH:MI:SS format
PROPAGATION_WINDOW	NUMBER	-	Duration in seconds for the propagation window
NEXT_TIME	VARCHAR2(200)	-	Function to compute the start of the next propagation window

Table 9–10 (Cont.) DBA_QUEUE_SCHEDULES View

Column	Datatype	NULL	Description
LATENCY	NUMBER	-	Maximum wait time to propagate a message during the propagation window
SCHEDULE_DISABLED	VARCHAR(1)	-	N if enabled; Y if disabled (schedule will not be executed)
PROCESS_NAME	VARCHAR2(8)	-	Name of Jnnn background process executing this schedule; NULL if not currently executing
SESSION_ID	VARCHAR2(82)	-	Session ID (SID, SERIAL#) of the job executing this schedule; NULL if not currently executing
INSTANCE	NUMBER	-	Real Application Clusters instance number executing this schedule
LAST_RUN_DATE	DATE	-	Date of the last successful execution
LAST_RUN_TIME	VARCHAR2(8)	-	Time of the last successful execution in HH:MI:SS format
CURRENT_START_DATE	DATE	-	Date the current window of this schedule was started
CURRENT_START_TIME	VARCHAR2(8)	-	Time the current window of this schedule was started in HH:MI:SS format
NEXT_RUN_DATE	DATE	-	Date the next window of this schedule will be started
NEXT_RUN_TIME	VARCHAR2(8)	-	Time the next window of this schedule will be started in HH:MI:SS format
TOTAL_TIME	NUMBER	-	Total time in seconds spent in propagating messages from the schedule
TOTAL_NUMBER	NUMBER	-	Total number of messages propagated in this schedule
TOTAL_BYTES	NUMBER	-	Total number of bytes propagated in this schedule
MAX_NUMBER	NUMBER	-	Maximum number of messages propagated in a propagation window
MAX_BYTES	NUMBER	-	Maximum number of bytes propagated in a propagation window
AVG_NUMBER	NUMBER	-	Average number of messages propagated in a propagation window
AVG_SIZE	NUMBER	-	Average size of propagated messages in bytes
AVG_TIME	NUMBER	-	Average time to propagate a message in seconds

Table 9–10 (Cont.) DBA_QUEUE_SCHEDULES View

Column	Datatype	NULL	Description
FAILURES	NUMBER	-	Number of times execution failed. If it reaches 16, then the schedule is disabled.
LAST_ERROR_DATE	DATE	-	Date of the last unsuccessful execution
LAST_ERROR_TIME	VARCHAR2(8)	-	Time of the last unsuccessful execution in HH:MI:SS format
LAST_ERROR_MSG	VARCHAR2(4000)	-	Error number and error message text of the last unsuccessful execution

Queue Subscribers View

Name of View

AQ$*Queue_Table_Name*_S

Purpose

This is a view of subscribers for all the queues in any given queue table. The subscriber view 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 is only created for 8.1-compatible queue tables.

Table 9–11 AQ$Queue_Table_Name_S View

Column	Datatype	NULL	Description
QUEUE	VARCHAR2(30)	NOT NULL	Name of queue for which subscriber is defined
NAME	VARCHAR2(30)	-	Name of agent
ADDRESS	VARCHAR2(1024)	-	Address of agent
PROTOCOL	NUMBER	-	Protocol of agent
TRANSFORMATION	VARCHAR2(61)	-	Name of the transformation (can be null)

Usage Notes

For queues created in 8.1-compatible queue tables, 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.

Queue Subscribers and Their Rules View

Name of View

AQ$*Queue_Table_Name*_R

Purpose

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 only created for 8.1-compatible queue tables.

Table 9–12 AQ$Queue_Table_Name_R View

Column	Datatype	NULL	Description
QUEUE	VARCHAR2(30)	NOT NULL	Name of queue for which subscriber is defined
NAME	VARCHAR2(30)	-	Name of agent
ADDRESS	VARCHAR2(1024)	-	Address of agent
PROTOCOL	NUMBER	-	Protocol of agent
RULE	CLOB	-	Text of defined rule
RULE_SET	VARCHAR2(65)	-	Set of rules
TRANSFORMATION	VARCHAR2(61)	-	Name of the transformation (can be null)

Number of Messages in Different States for the Whole Database View

Name of View

GV$AQ

Purpose

Provides information about the number of messages in different states for the whole database.

Table 9–13 GV$AQ View

Column	Datatype	NULL	Description
QID	NUMBER	-	Identity of the queue (same as QID in user_queues and dba_queues)
WAITING	NUMBER	-	Number of messages in the state WAITING
READY	NUMBER	-	Number of messages in state READY
EXPIRED	NUMBER	-	Number of messages in state EXPIRED
TOTAL_WAIT	NUMBER	-	Number of seconds messages in the queue have been waiting in state READY
AVERAGE_WAIT	NUMBER	-	Average number of seconds messages in state READY have been waiting to be dequeued

Number of Messages in Different States for Specific Instances View

Name of View

V$AQ

Purpose

Provides information about the number of messages in different states for specific instances.

Table 9–14 V$AQ View

Column	Datatype	NULL	Description
QID	NUMBER	-	Identity of the queue (same as QID in user_queues and dba_queues)
WAITING	NUMBER	-	Number of messages in the state WAITING
READY	NUMBER	-	Number of messages in state READY

Table 9–14 (Cont.) V$AQ View

Column	Datatype	NULL	Description
EXPIRED	NUMBER	-	Number of messages in state EXPIRED
TOTAL_WAIT	NUMBER	-	Number of seconds messages in the queue have been waiting in state READY
AVERAGE_WAIT	NUMBER	-	Average number of seconds messages in state READY have been waiting to be dequeued

Oracle Streams AQ Agents Registered for Internet Access View

Name of View

AQ$INTERNET_USERS

Purpose

Provides information about the agents registered for Internet access to Oracle Streams AQ. It also provides the list of database users that each Internet agent maps to.

Table 9–15 AQ$INTERNET_USERS View

Column	Datatype	NULL	Description
AGENT_NAME	VARCHAR2(30)	-	Name of the Oracle Streams AQ Internet agent
DB_USERNAME	VARCHAR2(30)	-	Name of database user that this Internet agent maps to
HTTP_ENABLED	VARCHAR2(4)	-	Indicates whether this agent is allowed to access Oracle Streams AQ through HTTP (YES or NO)
FTP_ENABLED	VARCHAR2(4)	-	Indicates whether this agent is allowed to access Oracle Streams AQ through FTP (always NO in current release)

All Transformations View

Name of View

DBA_TRANSFORMATIONS

Purpose

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.

Table 9–16 DBA_TRANSFORMATIONS View

Column	Datatype	NULL	Description
TRANSFORMATION_ID	NUMBER	NOT NULL	Unique ID for the transformation
OWNER	VARCHAR2(30)	NOT NULL	Owning user of the transformation
NAME	VARCHAR2(30)	NOT NULL	Transformation name
FROM_TYPE	VARCHAR2(61)	-	Source type name
TO_TYPE	VARCHAR2(91)	-	Target type name

All Transformation Functions View

Name of View

DBA_ATTRIBUTE_TRANSFORMATIONS

Purpose

Displays the transformation functions for all the transformations in the database.

Table 9–17 DBA_ATTRIBUTE_TRANSFORMATIONS View

Column	Datatype	NULL	Description
TRANSFORMATION_ID	NUMBER	NOT NULL	Unique ID for the transformation
OWNER	VARCHAR2(30)	NOT NULL	Transformation owner
NAME	VARCHAR2(30)	NOT NULL	Transformation name
FROM_TYPE	VARCHAR2(61)	-	Source type name

Table 9–17 (Cont.) DBA_ATTRIBUTE_TRANSFORMATIONS View

Column	Datatype	NULL	Description
TO_TYPE	VARCHAR2(91)	-	Target type name
ATTRIBUTE	NUMBER	NOT NULL	Target type attribute number
ATTRIBUTE_TRANSFORMATION	VARCHAR2(4000)	-	Transformation function for the attribute

User Transformations View

Name of View

USER_TRANSFORMATIONS

Purpose

Displays all the transformations owned by the user. To view the transformation definition, query USER_ATTRIBUTE_TRANSFORMATIONS.

Table 9–18 USER_TRANSFORMATIONS View

Column	Datatype	NULL	Description
TRANSFORMATION_ID	NUMBER	NOT NULL	Unique ID for the transformation
NAME	VARCHAR2(30)	NOT NULL	Transformation name
FROM_TYPE	VARCHAR2(61)	-	Source type name
TO_TYPE	VARCHAR2(91)	-	Target type name

User Transformation Functions View

Name of View

USER_ATTRIBUTE_TRANSFORMATIONS

Purpose

Displays the transformation functions for all the transformations of the user.

Table 9–19 USER_ATTRIBUTE_TRANSFORMATIONS View

Column	Datatype	NULL	Description
TRANSFORMATION_ID	NUMBER	NOT NULL	Unique ID for the transformation
NAME	VARCHAR2(30)	NOT NULL	Transformation name
FROM_TYPE	VARCHAR2(61)	-	Source type name
TO_TYPE	VARCHAR2(91)	-	Target type name
ATTRIBUTE	NUMBER	NOT NULL	Target type attribute number
ATTRIBUTE_ TRANSFORMATION	VARCHAR2(4000)	-	Transformation function for the attribute

10

Oracle Streams AQ Operational Interface: Basic Operations

This chapter describes the Oracle Streams Advanced Queuing (AQ) basic operational interface.

This chapter contains these topics:

- Enqueuing a Message
- Enqueuing an Array of Messages
- Listening to One or More Queues
- Dequeuing a Message
- 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 4, "Oracle Streams AQ: Programmatic Environments" for a list of available functions in each programmatic environment
- "DBMS_AQ" in *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)

Enqueuing a Message

This section contains these topics:

- Enqueuing a Message and Specifying Options
- Enqueuing a Message and Specifying Message Properties
- Enqueuing a Message and Specifying Sender ID
- Enqueuing a Message and Adding Payload

Purpose

Adds a **message** to the specified **queue**.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name          IN      VARCHAR2,
   payload             IN      "type_name",
   msgid               OUT     RAW);
```

Usage Notes

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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Enqueue a Single Message and Specify the Queue Name and Payload on page 10-6
- PL/SQL: Enqueue a Single Message and Specify the Priority on page 10-7
- PL/SQL: Enqueue a Single Message and Specify a Transformation on page 10-7
- Java (JDBC): Enqueue a Message and Add Payload on page 10-8
- Visual Basic (OO4O): Enqueue a message on page 10-11

Enqueuing a Message and Specifying Options

Purpose

Specifies options available for the **enqueue** operation.

Syntax

```
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);
```

Usage Notes

Do not use the `immediate` option when you want to use **LOB** locators. LOB locators are valid only for the duration of the transaction. Your locator will not be valid, because the `immediate` option automatically commits the transaction.

The `sequence deviation` parameter in enqueue options can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the enqueue options parameter relative msgid. The relationship is identified by the sequence deviation parameter.

Specifying 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.

The visibility option must be immediate for **nonpersistent** queues.

Only local recipients are supported for nonpersistent queues.

If a **transformation** is specified, then it is applied to the message before enqueuing it to the queue. The transformation must map the message into an object whose type is the Oracle **object type** of the queue.

Using Secure Queues

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter. See "MESSAGE_PROPERTIES_T Type" in *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 Streams AQ 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 Streams AQ Agent" on page 8-37
- "Enabling Database Access" on page 8-39
- *Oracle Streams Concepts and Administration* for information about secure queues

Enqueuing a Message and Specifying Message Properties

Purpose

Specifies message properties for the enqueue operation.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name          IN      VARCHAR2,
   message_properties  IN      message_properties_t,
   payload             IN      "type_name",
   msgid               OUT     RAW);
```

Usage Notes

Oracle Streams AQ uses message properties to manage individual messages. They are set when a message is enqueued, and their values are returned when the message is dequeued. To view messages in a waiting or processed state, you can either **dequeue** or browse by message ID, or use `SELECT` statements.

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 Streams AQ.

Enqueuing a Message and Specifying Sender ID

Purpose

Identifies the **producer** of a message.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name          IN      VARCHAR2,
   payload             IN      "type_name",
   msgid               OUT     RAW);
```

> **See Also:** "AQ Agent Type (aq$_agent)" on page 3-3 for more information on Agent

Enqueuing a Message and Adding Payload

To store a payload of type `RAW`, Oracle Streams AQ creates a **queue table** with `LOB` column as the payload repository. The maximum size of the payload is determined by which programmatic environment you use to access Oracle Streams AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G.

Examples

You must set up the following data structures for certain examples to work:

```
CONNECT system/manager
CREATE USER aq IDENTIFIED BY aq;
GRANT Aq_administrator_role TO aq;
***** CREATE TYPE ******

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            =>  'aq.objsgs_qtab',
   Queue_payload_type     =>  'aq.message_typ');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name            =>  'aq.msg_queue',
   Queue_table           =>  'aq.objmsgs_qtab');
EXECUTE DBMS_AQADM.START_QUEUE (
   Queue_name         => 'aq.msg_queue',
   Enqueue            => TRUE);
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            => 'aq.prioritymsgs_qtab',
   Sort_list              => 'PRIORITY,ENQ_TIME',
   Queue_payload_type     => 'aq.message_typ');
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   Queue_name             => 'aq.priority_msg_queue',
   Queue_table            => 'aq.prioritymsgs_qtab');
EXECUTE DBMS_AQADM.START_QUEUE (
   Queue_name             => 'aq.priority_msg_queue',
   Enqueue                => TRUE);
```

Example 10–1 PL/SQL: Enqueue a Single Message and Specify the Queue Name and Payload

```
/* Enqueue to msg_queue: */
DECLARE
   Enqueue_options     DBMS_AQ.enqueue_options_t;
   Message_properties  DBMS_AQ.message_properties_t;
   Message_handle      RAW(16);
   Message             aq.message_typ;
```

```
BEGIN
   Message := aq.message_typ('NORMAL MESSAGE',
      'enqueued to msg_queue first.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
   Enqueue_options            => enqueue_options,
   Message_properties         => message_properties,
   Payload                    => message,
   Msgid                      => message_handle);

   COMMIT;
END;
```

Example 10–2 PL/SQL: Enqueue a Single Message and Specify the Priority

```
/* The queue name priority_msg_queue is defined as an object type queue table.
   The payload object type is message. The schema of the queue is aq.  */

 /* Enqueue a message with priority 30: */
DECLARE
   Enqueue_options       dbms_aq.enqueue_options_t;
   Message_properties    dbms_aq.message_properties_t;
   Message_handle        RAW(16);
   Message               aq.Message_typ;

BEGIN
   Message := Message_typ('PRIORITY MESSAGE', 'enqued at priority 30.');

   message_properties.priority := 30;

   DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
   enqueue_options            => enqueue_options,
   message_properties         => message_properties,
   payload                    => message,
   msgid                      => message_handle);

   COMMIT;
END;
```

Example 10–3 PL/SQL: Enqueue a Single Message and Specify a Transformation

```
/* Enqueue to msg_queue: */
DECLARE
   Enqueue_options     DBMS_AQ.enqueue_options_t;
   Message_properties  DBMS_AQ.message_properties_t;
```

```
   Message_handle      RAW(16);
   Message             aq.message_typ;

BEGIN
   Message := aq.message_typ('NORMAL MESSAGE',
      'enqueued to msg_queue first.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
   Enqueue_options            => enqueue_options,
   Message_properties         => message_properties,
   transformation             => 'AQ.MSG_MAP',
   Payload                    => message,
   Msgid                      => message_handle);

   COMMIT;
END;
```

Where `MSG_MAP` was created as follows:

```
BEGIN
   DBMS.TRANSFORM.CREATE_TRANSFORMATION
   (
      schema => 'AQ',
      name =>   'MSG_MAP',
      from_schema => 'AQ',
      from_type => 'PO_ORDER1',
      to_schema => 'AQ',
      to_type => 'PO_ORDER2',
      transformation => 'AQ.MAP_PO_ORDER (source.user_data)'),
END;
```

Example 10–4   Java (JDBC): Enqueue a Message and Add Payload

```
/* Setup  */
connect system/manager
CREATE USER aq IDENTIFIED BY aq;
grant aq_administrator_role to aq;

public static void setup(AQSession aq_sess) throws AQException
{
     AQQueueTableProperty    qtable_prop;
     AQQueueProperty         queue_prop;
     AQQueueTable            q_table;
     AQQueue                 queue;
     AQAgent                 agent;
```

```
        qtable_prop = new AQQueueTableProperty("RAW");

        q_table = aq_sess.createQueueTable ("aq", "rawmsgs_qtab", qtable_prop);

        queue_prop = new AQQueueProperty();
        queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);

        queue.start();

        qtable_prop = new AQQueueTableProperty("RAW");
        qtable_prop.setMultiConsumer(true);

        qtable_prop.setSortOrder("priority,enq_time");
        q_table = aq_sess.createQueueTable ("aq", "rawmsgs_qtab2",
        qtable_prop);

        queue_prop = new AQQueueProperty();
        queue = aq_sess.createQueue (q_table, "priority_msg_queue", queue_prop);

        queue.start();

        agent = new AQAgent("subscriber1", null);

        queue.addSubscriber(agent, null);
    }


    /* Enqueue a message */
    public static void example(AQSession aq_sess) throws AQException, SQLException
    {
        AQQueue                  queue;
        AQMessage                message;
        AQRawPayload             raw_payload;
        AQEnqueueOption          enq_option;
        String                   test_data = "new message";
        byte[]                   b_array;
        Connection               db_conn;

        db_conn = ((AQOracleSession)aq_sess).getDBConnection();

        /* Get a handle to the queue */
        queue = aq_sess.getQueue ("aq", "msg_queue");

        /* Create a message to contain raw payload: */
        message = queue.createMessage();
```

```
        /* Get handle to the AQRawPayload object and populate it with raw data: */
        b_array = test_data.getBytes();

        raw_payload = message.getRawPayload();

        raw_payload.setStream(b_array, b_array.length);

        /* Create a AQEnqueueOption object with default options: */
        enq_option = new AQEnqueueOption();

        /* Enqueue the message: */
        queue.enqueue(enq_option, message);

        db_conn.commit();
   }


   /* Enqueue a message with priority = 5 */
   public static void example(AQSession aq_sess) throws AQException, SQLException
   {
        AQQueue                  queue;
        AQMessage                message;
        AQMessageProperty        msg_prop;
        AQRawPayload             raw_payload;
        AQEnqueueOption          enq_option;
        String                   test_data = "priority message";
        byte[]                   b_array;
        Connection               db_conn;

        db_conn = ((AQOracleSession)aq_sess).getDBConnection();

        /* Get a handle to the queue */
        queue = aq_sess.getQueue ("aq", "msg_queue");

        /* Create a message to contain raw payload: */
        message = queue.createMessage();

        /* Get Message property */
        msg_prop = message.getMessageProperty();

        /* Set priority */
        msg_prop.setPriority(5);

        /* Get handle to the AQRawPayload object and populate it with raw data: */
```

```
        b_array = test_data.getBytes();

        raw_payload = message.getRawPayload();

        raw_payload.setStream(b_array, b_array.length);

        /* Create a AQEnqueueOption object with default options: */
        enq_option = new AQEnqueueOption();

        /* Enqueue the message: */
        queue.enqueue(enq_option, message);

        db_conn.commit();
   }
```

Example 10–5 Visual Basic (OO4O): Enqueue a message

Enqueuing messages of type objects

```
'Prepare the message. MESSAGE_TYPE is a user-defined type
' in the "AQ" schema
Set OraMsg = Q.AQMsg(1, "MESSAGE_TYPE")
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")

OraObj("subject").Value = "Greetings from OO4O"
OraObj("text").Value = "Text of a message originated from OO4O"

Msgid = Q.Enqueue
```

Enqueuing messages of type RAW

```
'Create an OraAQ object for the queue "DBQ"
Dim Q as object
Dim Msg as object
Dim OraSession as object
Dim DB as object

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase(mydb, "scott/tiger" 0&)
Set Q = DB.CreateAQ("DBQ")

'Get a reference to the AQMsg object
Set Msg = Q.AQMsg
Msg.Value = "Enqueue the first message to a RAW queue."

'Enqueue the message
```

```
Q.Enqueue()

'Enqueue another message.

Msg.Value = "Another message"
Q.Enqueue()

'Enqueue a message with nondefault properties.
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Msg.Delay = 5
Msg.Value = "Urgent message"
Q.Enqueue()
Msg.Value = "The visibility option used in the enqueue call is
             ORAAQ_ENQ_IMMEDIATE"
Q.Visible = ORAAQ_ENQ_IMMEDIATE
Msgid = Q.Enqueue

'Enqueue Ahead of message Msgid_1
Msg.Value = "First Message to test Relative Message id"
Msg.Correlation = "RELATIVE_MESSAGE_ID"

Msgid_1 = Q.Enqueue
Msg.Value = "Second message to test RELATIVE_MESSAGE_ID is queued
             ahead of the First Message "
OraAq.relmsgid = Msgid_1
Msgid = Q.Enqueue
```

Enqueuing an Array of Messages

Purpose

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 IDs of the enqueued messages. The function returns the number of messages successfully enqueued.

Syntax

```
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;
```

Usage Notes

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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Array Enqueuing into a Queue of Type Message on page 10-13
- C(OCI): Array Enqueuing into a Queue of Type Message on page 10-14

Example 10–6   PL/SQL: Array Enqueuing into a Queue of Type Message

```
CREATE OR REPLACE TYPE message as OBJECT (
data VARCHAR2(10)) ;
/

CREATE OR REPLACE TYPE message_tbl AS TABLE OF message;
/

....

DECLARE
 enqopt dbms_aq.enqueue_options_t;
 msgproparr dbms_aq.message_properties_array_t;
 msgprop dbms_aq.message_properties_t;
 payloadarr  message_tbl;
 msgidarr dbms_aq.msgid_array_t;
 retval pls_integer;

BEGIN
  payloadarr := message_tbl(message('Oracle') ,message('Corp')) ;
  msgproparr := dbms_aq.message_properties_array_t(msgprop, msgprop);

  retval := dbms_aq.enqueue_array( queue_name => 'AQUSER.MY_QUEUE',
                 enqueue_options => enqopt ,
```

```
                 array_size => 2,
                 message_properties_array => msgproparr,
                 payload_array => payloadarr,
                 msgid_array => msgidarr ) ;
  commit;
END;
/
```

Example 10–7 C(OCI): Array Enqueuing into a Queue of Type Message

```
struct message
{

  OCIString   *data;
};
typedef struct message message;

struct null_message
{

  OCIInd null_adt;
  OCIInd null_data;
};
typedef struct null_message null_message;

int main( argc, argv)
int argc ;
char **argv ;
{

  OCIEnv              *envhp;
  OCIServer           *srvhp;
  OCIError            *errhp;
  OCISvcCtx           *svchp;
  OCISession          *usrhp;
  dvoid               *tmp;
  OCIType             *mesg_tdo = (OCIType *) 0;
  message              mesg[NMESGS];
  message             *mesgp[NMESGS];
  null_message         nmesg[NMESGS];
  null_message        *nmesgp[NMESGS];
  int                  i, j, k;
  OCIInd               ind[NMESGS];
  dvoid               *indptr[NMESGS];
  ub4                  priority;
```

```
OCIAQEnqOptions     *enqopt = (OCIAQEnqOptions *)0;
OCIAQMsgProperties  *msgprop= (OCIAQMsgProperties *)0;
ub4                  wait = 1;
ub4                  navigation = OCI_DEQ_NEXT_MSG;
ub4                  iters = 2;
text                *qname ;
text                 mesgdata[30];
ub4                  payload_size = 5;
text                *payload = (text *)0;
ub4                  batch_size = 2;
ub4                  enq_size = 2;

printf("session start\n");
/* establish a session */
OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
      (dvoid * (*)()) 0,  (void (*)()) 0 );

OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
         52, (dvoid **) &tmp);

OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
         52, (dvoid **) &tmp);
OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
         52, (dvoid **) &tmp);

printf("server attach\n");
OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
         52, (dvoid **) &tmp);

/* set attribute server context in the service context */
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
         (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* allocate a user context handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
         (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
         (dvoid *)"AQUSER", (ub4)strlen("AQUSER"),
           OCI_ATTR_USERNAME, errhp);
```

```
OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
         (dvoid *)"AQUSER", (ub4)strlen("AQUSER"),
         OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                 OCI_DEFAULT));

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
         (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* get descriptor for enqueue options */
checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&enqopt,
                                       OCI_DTYPE_AQENQ_OPTIONS, 0,
                                       (dvoid **)0));

printf("enq options set\n");
/* set enqueue options - for consumer name, wait and navigation */

/* construct null terminated payload string */
payload = (text *)malloc(payload_size+1);
for (k=0 ; k < payload_size ; k++)
  payload[k] = 'a';
payload[payload_size] = '\0';

for (k=0 ; k < batch_size ; k++)
{
  indptr[k] = &ind[k];
  mesgp[k] = &mesg[k];
  nmesgp[k] = &nmesg[k];
  nmesg[k].null_adt = nmesg[k].null_data = OCI_IND_NOTNULL;
  mesg[k].data = (OCIString *)0;
  OCIStringAssignText(envhp, errhp, (const unsigned char *)payload,
                      strlen((const char *)payload), &(mesg[k].data));
}

printf("check message tdo\n");
checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
 (CONST text *)"AQUSER", strlen("AQUSER"),
 (CONST text *)"MESSAGE", strlen("MESSAGE"), (text *)0, 0,
 OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo));
k=0;

while (k < iters)
{
  enq_size = batch_size;
```

```
    checkerr(errhp, OCIAQEnqArray(svchp, errhp,
                                  (dvoid *)"AQUSER.MY_QUEUE",
                                  (OCIAQEnqOptions *)0, &enq_size,
                                  0, mesg_tdo,
                                  (dvoid **)&mesgp,
                                  (dvoid **)&nmesgp, 0, 0, 0, 0));
    k+=batch_size;
  }

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

  checkerr(errhp, OCIServerDetach( srvhp, errhp, (ub4) OCI_DEFAULT));

  return 0;
}
```

Listening to One or More Queues

Purpose

Specifies which queue or queues to monitor

Syntax

```
DBMS_AQ.LISTEN (
   agent_list IN    aq$_agent_list_t,
   wait       IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
   agent      OUT   sys.aq$_agent);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

Usage Notes

The call takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. 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**.

This is a blocking call that returns 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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Listen to Single-Consumer Queue (Timeout of Zero) on page 10-18
- Java (JDBC): Listen to Queues on page 10-19
- C (OCI): Listening for Single-Consumer Queues with Zero Timeout on page 10-20

Example 10–8 PL/SQL: Listen to Single-Consumer Queue (Timeout of Zero)

```
/* The listen call monitors a list of queues for messages for
   specific agents. You must have dequeue privileges for all the queues
   you wish to monitor. */
DECLARE
   Agent_w_msg      aq$_agent;
   My_agent_list    dbms_aq.agent_list_t;

BEGIN
   /* NOTE:  MCQ1, MCQ2, MCQ3 are multiconsumer queues  in SCOTT's schema
   *        SCQ1, SCQ2, SCQ3 are single-consumer queues in SCOTT's schema
   */

   Qlist(1):= aq$_agent(NULL, 'scott.SCQ1',  NULL);
   Qlist(2):= aq$_agent(NULL, 'SCQ2', NULL);
   Qlist(3):= aq$_agent(NULL, 'SCQ3', NULL);

   /* Listen with a timeout of zero: */
   DBMS_AQ.LISTEN(
      Agent_list   =>   My_agent_list,
      Wait         =>   0,
      Agent        =>   agent_w_msg);
```

```
   DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' ||  agent_w_msg.address);
   DBMS_OUTPUT.PUT_LINE('');
END;
```

Example 10–9 Java (JDBC): Listen to Queues

```
public static void monitor_status_queue(Connection db_conn)
{
    AQSession         aq_sess;
    AQAgent[]         agt_list = null;
    AQAgent           ret_agt  = null;

    try
    {
        /* Create an AQ Session: */
        aq_sess = AQDriverManager.createAQSession(db_conn);

/* Construct the waiters list: */
agt_list = new AQAgent[3];

agt_list[0] = new AQAgent(null, "scott.SCQ1",0);
agt_list[1] = new AQAgent (null, "SCQ2",0);
agt_list[2] = new AQAgent (null, "SCQ3",0);

/* Wait for order status messages for 120 seconds: */
ret_agt = aq_sess.listen(agt_list, 120);

System.out.println("Message available for agent: " +
   ret_agt.getName()  + "   "  + ret_agt.getAddress());

    }
    catch (AQException aqex)
    {
System.out.println("Exception-1: " + aqex);
    }
    catch (Exception ex)
    {
        System.out.println("Exception-2: " + ex);
    }

}
```

Example 10–10 C (OCI): Listening for Single-Consumer Queues with Zero Timeout

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    switch (status)
    {
   case OCI_SUCCESS:
       break;
   case OCI_SUCCESS_WITH_INFO:
       printf("Error - OCI_SUCCESS_WITH_INFO\n");
       break;
   case OCI_NEED_DATA:
       printf("Error - OCI_NEED_DATA\n");
       break;
   case OCI_NO_DATA:
       printf("Error - OCI_NO_DATA\n");
       break;
   case OCI_ERROR:
       OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
       errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
       printf("Error - %s\n", errbuf);
       break;
   case OCI_INVALID_HANDLE:
       printf("Error - OCI_INVALID_HANDLE\n");
       break;
   case OCI_STILL_EXECUTING:
       printf("Error - OCI_STILL_EXECUTE\n");
       break;
   case OCI_CONTINUE:
       printf("Error - OCI_CONTINUE\n");
       break;
   default:
   break;
    }
}
```

```
/* set agent into descriptor */
void SetAgent(agent, appname, queue,errhp)

OCIAQAgent  *agent;
text        *appname;
text        *queue;
OCIError    *errhp;
{

  OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
     appname ? (dvoid *)appname : (dvoid *)"",
     appname ? strlen((const char *)appname) : 0,
        OCI_ATTR_AGENT_NAME, errhp);

  OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
     queue ? (dvoid *)queue : (dvoid *)"",
     queue ? strlen((const char *)queue) : 0,
        OCI_ATTR_AGENT_ADDRESS, errhp);

  printf("Set agent name to %s\n", appname ? (char *)appname : "NULL");
  printf("Set agent address to %s\n", queue ? (char *)queue : "NULL");
}

/* get agent from descriptor */
void GetAgent(agent, errhp)
OCIAQAgent *agent;
OCIError   *errhp;
{
text      *appname;
text      *queue;
ub4       appsz;
ub4       queuesz;

  if (!agent )
  {
    printf("agent was NULL \n");
    return;
  }
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
  if (!appsz)
     printf("agent name: NULL\n");
```

```
  else printf("agent name: %.*s\n", appsz, (char *)appname);
  if (!queuesz)
     printf("agent address: NULL\n");
  else printf("agent address: %.*s\n", queuesz, (char *)queue);
}

int main()
{
  OCIEnv *envhp;
  OCIServer *srvhp;
  OCIError *errhp;
  OCISvcCtx *svchp;
  OCISession *usrhp;
  OCIAQAgent *agent_list[3];
  OCIAQAgent *agent = (OCIAQAgent *)0;
  /* added next 2 121598 */
  int i;

 /* Standard OCI Initialization */

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
     (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp,
            (ub4) OCI_HTYPE_ENV, 0, (dvoid **) 0);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **) 0);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
     0, (dvoid **) 0);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
     0, (dvoid **) 0);

  OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
     0, (dvoid **) 0);

  /* set attribute server context in the service context */
  OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
     (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  /* allocate a user context handle */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
```

```
        (size_t) 0, (dvoid **) 0);

    /* allocate a user context handle */
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
       (size_t) 0, (dvoid **) 0);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
       (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

    OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
        (dvoid *) "tiger", (ub4) strlen("tiger"),
        (ub4) OCI_ATTR_PASSWORD, errhp);

    OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, OCI_DEFAULT);

    OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
       (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

    /* AQ LISTEN Initialization - allocate agent handles */
    for (i = 0; i < 3; i++)
    {
       agent_list[i] = (OCIAQAgent *)0;
       OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
           OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
    }

    /*
     *   SCQ1, SCQ2, SCQ3 are single-consumer queues in SCOTT's schema
     */

    SetAgent(agent_list[0], (text *)0, "SCOTT.SCQ1", errhp);
    SetAgent(agent_list[1], (text *)0, "SCOTT.SCQ2", errhp);
    SetAgent(agent_list[2], (text *)0, "SCOTT.SCQ3", errhp);

    checkerr(errhp,OCIAQListen(svchp, errhp, agent_list, 3, 0, &agent, 0));

    printf("MESSAGE for :- \n");
    GetAgent(agent, errhp);
    printf("\n");

  }
```

Example 10–11 C (OCI): Listening for Single-Consumer Queues with Timeout of 120 Seconds

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    ub4 buflen;
    sb4 errcode;

    switch (status)
    {
   case OCI_SUCCESS:
       break;
   case OCI_SUCCESS_WITH_INFO:
       printf("Error - OCI_SUCCESS_WITH_INFO\n");
       break;
   case OCI_NEED_DATA:
       printf("Error - OCI_NEED_DATA\n");
       break;
   case OCI_NO_DATA:
       printf("Error - OCI_NO_DATA\n");
       break;
   case OCI_ERROR:
       OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
       errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
       printf("Error - %s\n", errbuf);
       break;
   case OCI_INVALID_HANDLE:
       printf("Error - OCI_INVALID_HANDLE\n");
       break;
   case OCI_STILL_EXECUTING:
       printf("Error - OCI_STILL_EXECUTE\n");
       break;
   case OCI_CONTINUE:
       printf("Error - OCI_CONTINUE\n");
       break;
   default:
   break;
    }
```

```
}

/* set agent into descriptor */
/* void SetAgent(agent, appname, queue) */
void SetAgent(agent, appname, queue,errhp)

OCIAQAgent  *agent;
text        *appname;
text        *queue;
OCIError    *errhp;
{

  OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
     appname ? (dvoid *)appname : (dvoid *)"",
     appname ? strlen((const char *)appname) : 0,
        OCI_ATTR_AGENT_NAME, errhp);

  OCIAttrSet(agent, OCI_DTYPE_AQAGENT,
     queue ? (dvoid *)queue : (dvoid *)"",
     queue ? strlen((const char *)queue) : 0,
        OCI_ATTR_AGENT_ADDRESS, errhp);

  printf("Set agent name to %s\n", appname ? (char *)appname : "NULL");
  printf("Set agent address to %s\n", queue ? (char *)queue : "NULL");
}

/* get agent from descriptor */
void GetAgent(agent, errhp)
OCIAQAgent *agent;
OCIError   *errhp;
{
text      *appname;
text      *queue;
ub4       appsz;
ub4       queuesz;

  if (!agent )
  {
    printf("agent was NULL \n");
    return;
  }
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
```

```
  if (!appsz)
     printf("agent name: NULL\n");
  else printf("agent name: %.*s\n", appsz, (char *)appname);
  if (!queuesz)
     printf("agent address: NULL\n");
  else printf("agent address: %.*s\n", queuesz, (char *)queue);
}

int main()
{
  OCIEnv *envhp;
  OCIServer *srvhp;
  OCIError *errhp;
  OCISvcCtx *svchp;
  OCISession *usrhp;
  OCIAQAgent *agent_list[3];
  OCIAQAgent *agent = (OCIAQAgent *)0;
  /* added next 2 121598 */
  int i;

 /* Standard OCI Initialization */

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
     (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp,
             (ub4) OCI_HTYPE_ENV, 0, (dvoid **) 0);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 0, (dvoid **) 0);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
     0, (dvoid **) 0);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
     0, (dvoid **) 0);

  OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
     0, (dvoid **) 0);

  /* set attribute server context in the service context */
  OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
     (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);
```

```
/* allocate a user context handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
    (size_t) 0, (dvoid **) 0);

/* allocate a user context handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
   (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
   (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION,
    (dvoid *) "tiger", (ub4) strlen("tiger"),
    (ub4) OCI_ATTR_PASSWORD, errhp);

OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS, OCI_DEFAULT);

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
   (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* AQ LISTEN Initialization - allocate agent handles */
for (i = 0; i < 3; i++)
{
   agent_list[i] = (OCIAQAgent *)0;
   OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
       OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
}

/*
 *   SCQ1, SCQ2, SCQ3 are single-consumer queues in SCOTT's schema
 */

SetAgent(agent_list[0], (text *)0, "SCOTT.SCQ1", errhp);
SetAgent(agent_list[1], (text *)0, "SCOTT.SCQ2", errhp);
SetAgent(agent_list[2], (text *)0, "SCOTT.SCQ3", errhp);

checkerr(errhp,OCIAQListen(svchp, errhp, agent_list, 3, 120, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");

}
```

Dequeuing a Message

This section contains these topics:

- Dequeuing a Message from a Single-Consumer Queue and Specifying Options
- Dequeuing a Message from a Multiconsumer Queue and Specifying Options

Purpose

Dequeues a message from the specified queue.

Syntax

```
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);
```

Usage Notes

The search criteria for messages to be dequeued is determined by the **consumer** name, msgid and correlation parameters in the dequeue options. Parameter msgid uniquely identifies the message to be dequeued. Only messages in the READY state are dequeued unless msgid is specified. Correlation identifiers are application-defined identifiers that are not interpreted by Oracle Streams AQ.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the message ID and correlation ID 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.

The default NAVIGATION parameter during dequeue is NEXT MESSAGE. This means that subsequent dequeues retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command is processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the FIRST_MESSAGE navigation option when the first message in the queue must be processed by every dequeue command. This usually becomes necessary when a higher priority message arrives in the queue while messages already enqueued are being processed.

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 Streams AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the FIRST_MESSAGE option is specified, then Oracle Streams AQ 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 the NEXT TRANSACTION to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue times out after the specified WAIT period.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Dequeue of Object Type Messages on page 10-30
- Java (JDBC): Dequeue a message from a single-consumer queue (specify options) on page 10-30
- Visual Basic (OO4O): Dequeue a message on page 10-31

Dequeuing a Message from a Single-Consumer Queue and Specifying Options

Purpose

Specifies the options available for the dequeue operation.

Usage Notes

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.

The transformation, if specified, is applied before returning the message to the caller. The transformation should be defined to map the queue Oracle object type to the return type wanted by the caller.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Dequeue of Object Type Messages on page 10-30
- Java (JDBC): Dequeue a message from a single-consumer queue (specify options) on page 10-30
- Visual Basic (OO4O): Dequeue a message on page 10-31

Example 10–12 PL/SQL: Dequeue of Object Type Messages

```
/* Dequeue from msg_queue: */
DECLARE
dequeue_options     dbms_aq.dequeue_options_t;
message_properties  dbms_aq.message_properties_t;
message_handle      RAW(16);
message             aq.message_typ;

BEGIN
   DBMS_AQ.DEQUEUE(
      queue_name          =>     'msg_queue',
      dequeue_options      =>    dequeue_options,
      message_properties  =>     message_properties,
      payload             =>     message,
      msgid               =>     message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                      ' ... ' || message.text );
   COMMIT;
END;
```

Example 10–13 Java (JDBC): Dequeue a message from a single-consumer queue (specify options)

```
/* Dequeue a message with correlation ID = 'RUSH' */
```

```
public static void example(AQSession aq_sess) throws AQException, SQLException
{
     AQQueue                  queue;
     AQMessage                message;
     AQRawPayload             raw_payload;
     AQDequeueOption          deq_option;
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

     queue = aq_sess.getQueue ("aq", "msg_queue");

     /* Create a AQDequeueOption object with default options: */
     deq_option = new AQDequeueOption();

     deq_option.setCorrelation("RUSH");

     /* Dequeue a message */
     message = queue.dequeue(deq_option);

     System.out.println("Successful dequeue");

     /* Retrieve raw data from the message: */
     raw_payload = message.getRawPayload();

     b_array = raw_payload.getBytes();

     db_conn.commit();
}
```

Example 10–14 Visual Basic (OO4O): Dequeue a message

Dequeuing messages of RAW type

```
 'Dequeue the first message available
Q.Dequeue()
Set Msg = Q.QMsg

'Display the message content
MsgBox Msg.Value

'Dequeue the first message available without removing it
' from the queue
Q.DequeueMode = ORAAQ_DEQ_BROWSE
```

```
'Dequeue the first message with the correlation identifier
' equal to "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_FIRST_MSG
Q.correlate = "RELATIVE_MESSAGE_ID"
Q.Dequeue

'Dequeue the next message with the correlation identifier

' of "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_NEXT_MSG
Q.Dequeue()

'Dequeue the first high priority message
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Q.Dequeue()

'Dequeue the message enqueued with message ID of Msgid_1
Q.DequeueMsgid = Msgid_1
Q.Dequeue()

'Dequeue the message meant for "ANDY"
Q.consumer = "ANDY"
Q.Dequeue()

'Return immediately if there is no message on the queue
Q.wait = ORAAQ_DQ_NOWAIT
Q.Dequeue()
```

Dequeuing messages of Oracle object type

```
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")
Set QMsg = Q.AQMsg(1, "MESSAGE_TYPE")

'Dequeue the first message available without removing it
Q.Dequeue()
OraObj = QMsg.Value

'Display the subject and data
MsgBox OraObj!subject & OraObj!Data
```

Dequeuing a Message from a Multiconsumer Queue and Specifying Options

Purpose

Specifies the options available for the dequeue operation.

Usage Notes

See "Dequeuing a Message from a Single-Consumer Queue and Specifying Options" on page 10-29.

Examples

Examples are provided in the following programmatic environments:

- Java (JDBC): Dequeue a message from a multiconsumer queue (specify options) on page 10-33

Example 10–15 Java (JDBC): Dequeue a message from a multiconsumer queue (specify options)

```
/* Dequeue a message for subscriber1 in browse mode*/
public static void example(AQSession aq_sess) throws AQException, SQLException
{
     AQQueue                  queue;
     AQMessage                message;
     AQRawPayload             raw_payload;
     AQDequeueOption          deq_option;
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

     queue = aq_sess.getQueue ("aq", "priority_msg_queue");

     /* Create a AQDequeueOption object with default options: */
     deq_option = new AQDequeueOption();

     /* Set dequeue mode to BROWSE */
     deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_BROWSE);

     /* Dequeue messages for subscriber1 */
     deq_option.setConsumerName("subscriber1");

     /* Dequeue a message: */
```

```
        message = queue.dequeue(deq_option);

        System.out.println("Successful dequeue");

        /* Retrieve raw data from the message: */
        raw_payload = message.getRawPayload();

        b_array = raw_payload.getBytes();

        db_conn.commit();
    }
```

Dequeuing an Array of Messages

Purpose

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.

Syntax

```
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;
```

Usage Notes

A nonzero wait time, as specified in `dequeue_options`, is recognized only when there are no messages in the queue. If the queue contains messages that are eligible for dequeue, then the `DEQUEUE_ARRAY` function will dequeue up to `array_size` messages and return immediately.

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`. 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.

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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL: Array Dequeuing from a Queue of Type Message on page 10-36
- C(OCI): Array Dequeuing from a Queue of Type Message on page 10-36

Example 10–16 PL/SQL: Array Dequeuing from a Queue of Type Message

```
CREATE OR REPLACE TYPE message as OBJECT (data VARCHAR2(10));
/

CREATE OR REPLACE TYPE message_arr AS VARRAY(2000) OF message;
/

....

DECLARE
 deqopt dbms_aq.dequeue_options_t ;
 msgproparr dbms_aq.message_properties_array_t :=
              dbms_aq.message_properties_array_t();
 payloadarr  message_arr := message_arr() ;
 msgidarr dbms_aq.msgid_array_t ;
 retval pls_integer ;
BEGIN
  payloadarr.extend(2);
  msgproparr.extend(2);
  deqopt.consumer_name := 'SUB1';
  retval := dbms_aq.dequeue_array( queue_name => 'AQUSER.MY_QUEUE',
                dequeue_options => deqopt ,
                array_size => payloadarr.count,
                message_properties_array => msgproparr,
                payload_array => payloadarr,
                msgid_array => msgidarr ) ;
END;
/
```

Example 10–17 C(OCI): Array Dequeuing from a Queue of Type Message

```
struct message
{

  OCIString   *data;
};
typedef struct message message;

struct null_message
{

  OCIInd null_adt;
  OCIInd null_data;
};
```

```
typedef struct null_message null_message;

int main(argc, argv)
 int argc;
 char **argv;
{

  OCIEnv                *envhp;
  OCIServer             *srvhp;
  OCIError              *errhp;
  OCISvcCtx             *svchp;
  OCISession            *usrhp;
  dvoid                 *tmp;
  message               *mesgp[NMESGS];
  int                    i, j, k;
  null_message          *nmesgp[NMESGS];
  ub4                    priority = 0;
  OCIAQDeqOptions       *deqopt = (OCIAQDeqOptions *)0;
  ub4                    iters = 2;
  OCIType               *mesg_tdo = (OCIType *) 0;
  ub4                    batch_size = 2;
  ub4                    deq_size = batch_size;

 printf("session start\n");
  /* establish a session */
  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
        (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc( (dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
           52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
           52, (dvoid **) &tmp);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
           52, (dvoid **) &tmp);

  printf("server attach\n");
  OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
           52, (dvoid **) &tmp);
```

```
/* set attribute server context in the service context */
OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
         (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

/* allocate a user context handle */
OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
         (size_t) 0, (dvoid **) 0);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
         (dvoid *)"AQUSER", (ub4)strlen("AQUSER"),
           OCI_ATTR_USERNAME, errhp);

OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
         (dvoid *)"AQUSER", (ub4)strlen("AQUSER"),
         OCI_ATTR_PASSWORD, errhp);

checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                 OCI_DEFAULT));

OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
         (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

/* get descriptor for dequeue options */
checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
                                       OCI_DTYPE_AQDEQ_OPTIONS, 0,
                                       (dvoid **)0));

printf("deq options set\n");
/* set dequeue options - for consumer name, wait and navigation */
checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
                               (dvoid *)"SUB1",
                               (ub4)strlen("SUB1"),
                               OCI_ATTR_CONSUMER_NAME, errhp));

for (k=0 ; k < NMESGS ; k++)
{
  mesgp[k] = 0;
  nmesgp[k] = 0;
}

printf("check message tdo\n");
checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
      (CONST text *)"AQUSER", strlen("AQUSER"),
      (CONST text *)"MESSAGE", strlen("MESSAGE"), (text *)0, 0,
      OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo));
```

```
 k=0;

  while (k < iters)
  {
    deq_size = batch_size;
    checkerr(errhp, OCIAQDeqArray(svchp, errhp,
                                  (text *)"AQUSER.MY_QUEUE",
                                  (OCIAQDeqOptions *)deqopt,
                                  &deq_size, 0, mesg_tdo,
                                  (dvoid **)mesgp,
                                  (dvoid **)nmesgp, 0, 0, 0, 0));
    k+=batch_size;
  }
  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

  checkerr(errhp, OCIServerDetach( srvhp, errhp, (ub4) OCI_DEFAULT));
  return 0;
}
```

Registering for Notification

Purpose

Registers a callback for message notification.

Syntax

```
DBMS_AQ.REGISTER (
   reg_list IN SYS.AQ$_REG_INFO_LIST,
   count    IN NUMBER);
```

Usage Notes

This call is invoked for registration to a subscription which identifies the subscription name of interest and the associated callback to be invoked. Interest in several subscriptions can be registered at one time.

This interface is only valid for the **asynchronous** mode of message delivery. In this mode, a **subscriber** applies a registration call which specifies a callback. When messages are received that match the subscription criteria, the callback is invoked. The callback can then apply an explicit `message_receive` (dequeue) to retrieve the message.

The user must specify a subscription handle at registration time with the namespace attribute set to `OCI_SUBSCR_NAMESPACE_AQ`.

The subscription name is the string `schema.queue` if the registration is for a single-consumer queue and `schema.queue:consumer_name` if the registration is for a multiconsumer queues.

Related Functions: `OCIAQListen()`, `OCISubscriptionDisable()`, `OCISubscriptionEnable()`, `OCISubscriptionUnRegister()`

Examples

Example 10–18 C (OCI): Register for Notifications For Single-Consumer and Multiconsumer Queries

```
/* OCIRegister can be used by the client to register to receive notifications
   when messages are enqueued into nonpersistent and usual queues. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>


static OCIEnv     *envhp;
static OCIServer  *srvhp;
static OCIError   *errhp;
static OCISvcCtx  *svchp;


/* The callback that gets invoked on notification */
ub4 notifyCB(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;      /* subscription handle */
dvoid           *pay;           /* payload  */
ub4              payl;          /* payload length */
dvoid           *desc;          /* the AQ notification descriptor */
ub4              mode;
{
 text                 *subname;
 ub4                   size;
 ub4                  *number = (ub4 *)ctx;
 text                 *queue;
 text                 *consumer;
 OCIRaw               *msgid;
 OCIAQMsgProperties   *msgprop;
```

```
  (*number)++;

  /* Get the subscription name */
  OCIAttrGet((dvoid *)subscrhp, OCI_HTYPE_SUBSCRIPTION,
                             (dvoid *)&subname, &size,
                             OCI_ATTR_SUBSCR_NAME, errhp);
 printf("got notification number %d for %.*s  %d  \n",
         *number, size, subname, payl);

 /* Get the queue name from the AQ notify descriptor */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&queue, &size,
             OCI_ATTR_QUEUE_NAME, errhp);

 /* Get the consumer name for which this notification was received */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&consumer, &size,
       OCI_ATTR_CONSUMER_NAME, errhp);

 /* Get the message ID of the message for which we were notified */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgid, &size,
       OCI_ATTR_NFY_MSGID, errhp);

 /* Get the message properties of the message for which we were notified */
 OCIAttrGet(desc, OCI_DTYPE_AQNFY_DESCRIPTOR, (dvoid *)&msgprop, &size,
       OCI_ATTR_MSG_PROP, errhp);

}


int main(argc, argv)
int argc;
char *argv[];
{
  OCISession *authp = (OCISession *) 0;

  /* The subscription handles */
  OCISubscription *subscrhp[5];

  /* Registrations are for AQ namespace */
  ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;

  /* The context fot the callback */
  ub4 ctx[5] = {0,0,0,0,0};

  printf("Initializing OCI Process\n");
```

```
/* The OCI Process Environment must be initialized  with OCI_EVENTS */
/* OCI_OBJECT flag is set to enable us dequeue */
(void) OCIInitialize((ub4) OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                     (dvoid * (*)(dvoid *, size_t)) 0,
                     (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                     (void (*)(dvoid *, dvoid *)) 0 );

printf("Initialization successful\n");

/* The standard OCI setup */
printf("Initializing OCI Env\n");
(void) OCIEnvInit((OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0,
              (dvoid **) 0 );

(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                 (size_t) 0, (dvoid **) 0);

/* Server contexts */
(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                 (size_t) 0, (dvoid **) 0);

(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                 (size_t) 0, (dvoid **) 0);


printf("connecting to server\n");
(void) OCIServerAttach( srvhp, errhp, (text *)"", strlen(""), 0);
printf("connect successful\n");

/* Set attribute server context in the service context */
(void) OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
        (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp);

(void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
           (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0);

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
               (dvoid *) "scott", (ub4) strlen("scott"),
               (ub4) OCI_ATTR_USERNAME, errhp);

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
               (dvoid *) "tiger", (ub4) strlen("tiger"),
               (ub4) OCI_ATTR_PASSWORD, errhp);
```

```
 checkerr(errhp, OCISessionBegin ( svchp,  errhp, authp, OCI_CRED_RDBMS,
          (ub4) OCI_DEFAULT));

 (void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                   (dvoid *) authp, (ub4) 0,
                   (ub4) OCI_ATTR_SESSION, errhp);

/* Setting the subscription handle for notification on
   a NORMAL single-consumer queue */
 printf("allocating subscription handle\n");
 subscrhp[0] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[0],
        (ub4) OCI_HTYPE_SUBSCRIPTION,
        (size_t) 0, (dvoid **) 0);

 printf("setting subscription name\n");
 (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                (dvoid *) "SCOTT.SCQ1", (ub4) strlen("SCOTT.SCQ1"),
                (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

 printf("setting subscription callback\n");
 (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) notifyCB, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

printf("setting subscription context \n");
(void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *)&ctx[0], (ub4)sizeof(ctx[0]),
                 (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

 printf("setting subscription namespace\n");
 (void) OCIAttrSet((dvoid *) subscrhp[0], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) &namespace, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);

/* Setting the subscription handle for notification on a NORMAL multiconsumer
   consumer queue */
 subscrhp[1] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[1],
        (ub4) OCI_HTYPE_SUBSCRIPTION,
        (size_t) 0, (dvoid **) 0);

 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) "SCOTT.MCQ1:APP1",
                 (ub4) strlen("SCOTT.MCQ1:APP1"),
```

```
                 (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) notifyCB, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *)&ctx[1], (ub4)sizeof(ctx[1]),
                 (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[1], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) &namespace, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);


/* Setting the subscription handle for notification on a nonpersistent
  single-consumer queue */
 subscrhp[2] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[2],
         (ub4) OCI_HTYPE_SUBSCRIPTION,
         (size_t) 0, (dvoid **) 0);

 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) "SCOTT.NP_SCQ1",
                 (ub4) strlen("SCOTT.NP_SCQ1"),
                 (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) notifyCB, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *)&ctx[2], (ub4)sizeof(ctx[2]),
                 (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[2], (ub4) OCI_HTYPE_SUBSCRIPTION,
                 (dvoid *) &namespace, (ub4) 0,
                 (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);


/* Setting the subscription handle for notification on
   a nonpersistent multi consumer queue */
/* Waiting on user specified recipient */
 subscrhp[3] = (OCISubscription *)0;
 (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&subscrhp[3],
```

```
        (ub4) OCI_HTYPE_SUBSCRIPTION,
        (size_t) 0, (dvoid **) 0);

 (void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *) "SCOTT.NP_MCQ1",
               (ub4) strlen("SCOTT.NP_MCQ1"),
               (ub4) OCI_ATTR_SUBSCR_NAME, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *) notifyCB, (ub4) 0,
               (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);

(void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *)&ctx[3], (ub4)sizeof(ctx[3]),
               (ub4) OCI_ATTR_SUBSCR_CTX, errhp);

 (void) OCIAttrSet((dvoid *) subscrhp[3], (ub4) OCI_HTYPE_SUBSCRIPTION,
               (dvoid *) &namespace, (ub4) 0,
               (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);


 printf("Registering for all the subscriptiosn \n");
 checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 4, errhp,
               OCI_DEFAULT));

 printf("Waiting for notifcations \n");

 /* wait for minutes for notifications */
 sleep(300);

 printf("Exiting\n");
}
```

Posting for Subscriber Notification

Purpose

Posts to a list of anonymous subscriptions so clients registered for the subscription get notifications.

Syntax

```
DBMS_AQ.POST (
  post_list  IN  SYS.AQ$_POST_INFO_LIST,
  count      IN NUMBER);
```

Usage Notes

Several subscriptions can be posted to at one time. Posting to a subscription involves identifying the subscription name and the payload, if wanted. 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 and is useful in the case of several system events. If an application needs more rigid guarantees, then it can use Oracle Streams AQ functionality by enqueuing to a queue.

When using OCI, you must specify a subscription handle at registration time with the namespace attribute set to `OCI_SUBSCR_NAMESPACE_ANONYMOUS`.

When using PL/SQL, the namespace attribute in `aq$_post_info` must be set to `DBMS_AQ.NAMESPACE_ANONYMOUS`.

Related functions: `OCIAQListen()`, `OCISvcCtxToLda()`, `OCISubscriptionEnable()`, `OCISubscriptionRegister()`, `OCISubscriptionUnRegister()`, `dbms_aq.register`, `dbms_aq.unregister`.

Examples

Example 10–19 PL/SQL: Post of Object-Type Messages

```
-- Register for notification
DECLARE
  reginfo             sys.aq$_reg_info;
  reginfolist         sys.aq$_reg_info_list;

BEGIN
  -- Register for anonymous subscription PUBSUB1.ANONSTR, consumer_name ADMIN
```

```
  -- The PL/SQL callback pubsub1.mycallbk is invoked
  -- when a notification is received
  reginfo := sys.aq$_reg_info('PUBSUB1.ANONSTR:ADMIN',
      DBMS_AQ.NAMESPACE_ANONYMOUS,
      'plsql://PUBSUB1.mycallbk', HEXTORAW('FF'));

  reginfolist := sys.aq$_reg_info_list(reginfo);

  sys.dbms_aq.register(reginfolist, 1);

  commit;
END;
/

-- Post to an anonymous subscription
DECLARE

  postinfo            sys.aq$_post_info;
  postinfolist        sys.aq$_post_info_list;

BEGIN

  -- Post to the anonymous subscription PUBSUB1.ANONSTR, consumer_name ADMIN
  postinfo := sys.aq$_post_info('PUBSUB1.ANONSTR:ADMIN',0,HEXTORAW('FF'));
  postinfolist := sys.aq$_post_info_list(postinfo);

  sys.dbms_aq.post(postinfolist, 1);

  commit;

END;
/
```

Adding an Agent to the LDAP Server

Purpose

Adds an agent to the **Lightweight Directory Access Protocol** (LDAP) server.

Syntax

```
DBMS_AQ.BIND_AGENT(
   agent        IN SYS.AQ$_AGENT,
   certificate  IN VARCHAR2 default NULL);
```

Usage Notes

This call takes an agent and an optional certificate location as the arguments, and adds the agent entry to the LDAP server. The certificate location parameter is the distinguished name of the LDAP entry that contains the digital certificate which the agent uses. If the agent does not have a digital certificate, then this parameter is defaulted to null.

Removing an Agent from the LDAP Server

Purpose

Removes an agent from the LDAP server.

Syntax

```
DBMS_AQ.UNBIND_AGENT(
   agent     IN SYS.AQ$_AGENT);
```

Usage Notes

This call takes an agent as the argument, and removes the corresponding agent entry in the LDAP server.

Part V

Using Oracle JMS and Oracle Streams AQ

Part V describes how to use Oracle JMS and Oracle Streams Advanced Queuing (AQ).

This part contains the following chapters:

- Chapter 11, "Creating Oracle Streams AQ Applications Using JMS"
- Chapter 12, "Oracle Streams AQ JMS Interface: Basic Operations"
- Chapter 13, "Oracle Streams AQ JMS Operational Interface: Point-to-Point"
- Chapter 14, "Oracle Streams AQ JMS Operational Interface: Publish/Subscribe"
- Chapter 15, "Oracle Streams AQ JMS Operational Interface: Shared Interfaces"
- Chapter 16, "Oracle Streams AQ JMS Types Examples"

See Also:

For Oracle APIs for JMS see:

- `http://otn.oracle.com/docs/products/aq/doc_library/ojms/index.html`

For J2EE Guides see:

- *Oracle9iAS Containers for J2EE Enterprise JavaBeans Guide*
- *Oracle9iAS Containers for J2EE Services Guide*

11

Creating Oracle Streams AQ Applications Using JMS

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

General Features of JMS and Oracle JMS

This section contains these topics:

- J2EE Compliance
- 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

J2EE Compliance

In Oracle Database 10*g*, 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 run time. 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 Streams AQ that support J2EE compliance (and are also available in the noncompliant mode) include:

- Nontransactional sessions
- Nondurable subscribers
- Temporary queues and topics
- Nonpersistent delivery mode
- Multiple JMS messages types on a single JMS **queue** or topic (using Oracle Streams AQ queues of the `AQ$_JMS_MESSAGE` type)
- The `noLocal` option for durable subscribers

See Also:

- *Java Message Service Specification*, version 1.1, March 18, 2002, Sun Microsystems, Inc.
- http://otn.oracle.com/docs/products/aq/doc_library/ojms/index.html for more information on Oracle JMS

Features of JMSPriority, JMSExpiration, and nondurable subscribers vary depending on which mode you use.

JMSPriority

Table 11–1 shows how JMSPriority values depend on whether you are running the default, noncompliant mode or the compliant mode, in which you set the compliance flag to TRUE.

Table 11–1 JMSPriority

Priority	Noncompliant Mode	Compliant Mode
Lowest	java.lang.Integer.MAX_VALUE	0
Highest	java.lang.Integer.MIN_VALUE	9
Default	1	4

JMSExpiration

JMSExpiration values depend on whether you are running the default, noncompliant mode or the compliant mode, in which you set the compliance flag to TRUE.

In noncompliant mode, the JMSExpiration header value is the sum of the **enqueue** time and the TimeToLive, as specified in the JMS specification when a **message** is enqueued. When a message is received, the duration of the expiration (not the expiration time) is returned. If a message never expires, then -1 is returned.

In compliant mode, the JMSExpiration header value in a dequeued message is the sum of the JMS time stamp when the message was enqueued (Greenwich Mean Time, in milliseconds) and the TimeToLive (in milliseconds). If a message never expires, then 0 is returned.

Durable Subscribers

Durable **subscriber** action, when subscribers use the same name, depends on whether you are running the default, noncompliant mode or the compliant mode, in which you set the compliance flag to `TRUE`.

In noncompliant mode, two durable TopicSubscribers 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 Streams AQ 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.

JMS Connection and Session

This section contains these topics:

- ConnectionFactory Objects
- Using AQjmsFactory to Obtain ConnectionFactory Objects
- Using JNDI to Look Up ConnectionFactory Objects
- JMS Connection
- JMS Session
- JMS Connection Examples

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`

You can obtain `ConnectionFactory` objects two different ways:

- Using AQjmsFactory to Obtain ConnectionFactory Objects
- Using JNDI to Look Up ConnectionFactory Objects

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.

Example 11–1 JMS: Getting a Queue Connection Factory for a Database

```
public static void get_Factory() throws JMSException
{
  QueueConnectionFactory    qc_fact   = null;
```

```
/* get queue connection factory for database "aqdb", host "sun-123",
port 5521, driver "thin" */
qc_fact = AQjmsFactory.getQueueConnectionFactory("sun-123", "aqdb",
                                                  5521, "thin");
}
```

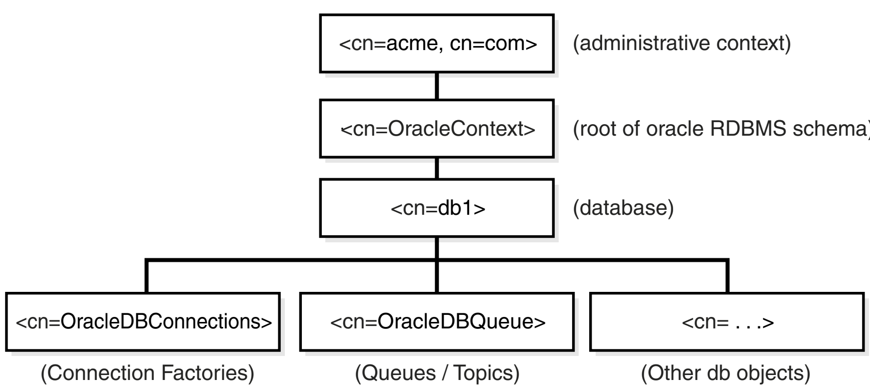
Using JNDI to Look Up ConnectionFactory Objects

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:

- Register Database
- Set Parameter GLOBAL_TOPIC_ENABLED
- Register ConnectionFactory Objects

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 Streams AQ entries in the LDAP server. `ConnectionFactory` information is stored under `<cn=OracleDBConnections>`, while topics and queues are stored under `<cn=OracleDBQueues>`.

Figure 11–1 Structure of Oracle Streams AQ Entries in LDAP Server



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 Streams AQ are automatically registered with the LDAP server. This parameter can be set by using

```
ALTER SYSTEM SET GLOBAL_TOPICS_ENABLED = TRUE
```

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).

After the `ConnectionFactory` objects have been registered in LDAP by a JMS administrator, they can be looked up by using JNDI

Example 11–2 Registering a Order Entry Queue Connection Factory in LDAP

Suppose the JMS administrator wants to register an order entry queue **connection factory**, `oe_queue_factory`. In LDAP, it can be registered as follows:

```
public static void register_Factory_in_LDAP() throws Exception
{
    Hashtable env = new Hashtable(5, 0.75f);
    env.put(Context.INITIAL_CONTEXT_FACTORY, AQjmsConstants.INIT_CTX_FACTORY);

    // aqldapserv is your LDAP host and 389 is your port
    env.put(Context.PROVIDER_URL, "ldap://aqldapserv:389);

    // now authentication information
    // username/password scheme, user is OE, password is OE
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "cn=oe,cn=users,cn=acme,cn=com");
    env.put(Context.SECURITY_CREDENTIALS, "oe");

    /* register queue connection factory for database "aqdb", host "sun-123",
    port 5521, driver "thin" */
    AQjmsFactory.registerConnectionFactory(env, "oe_queue_factory", "sun-123",
        "aqdb", 5521, "thin", "queue");
}
```

After order entry, queue connection factory `oe_queue_factory` has been registered in LDAP; it can be looked up as follows:

```
public static void get_Factory_from_LDAP() throws Exception
{
    Hashtable env = new Hashtable(5, 0.75f);
    env.put(Context.INITIAL_CONTEXT_FACTORY, AQjmsConstants.INIT_CTX_FACTORY);

    // aqldapserv is your LDAP host and 389 is your port
    env.put(Context.PROVIDER_URL, "ldap://aqldapserv:389);

    // now authentication information
    // username/password scheme, user is OE, password is OE
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "cn=oe,cn=users,cn=acme,cn=com");
    env.put(Context.SECURITY_CREDENTIALS, "oe");

    DirContext inictx = new InitialDirContext(env);
    // initialize context with the distinguished name of the database server
    inictx=(DirContext)inictx.lookup("cn=db1,cn=OracleContext,cn=acme,cn=com");
```

```
    //go to the connection factory holder cn=OraclDBConnections
    DirContext connctx = (DirContext)inictx.lookup("cn=OracleDBConnections");

    // get connection factory "oe_queue_factory"
    QueueConnectionFactory qc_fact =
      (QueueConnectionFactory)connctx.lookup("cn=oe_queue_factory");
}
```

JMS Connection

A JMS `Connection` is a client's active connection to 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.

Connection Setup A JMS client typically creates a `Connection`, `Session` and a number of `MessageProducers` and `MessageConsumers`. 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.

Some of the methods that are supported on the `Connection` object are

- `start()`, which starts or restart delivery of incoming messages
- `stop()`, which 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()`, which closes the **JMS session** and releases all associated resources
- `createSession(true, 0)`, which creates a JMS `Session` using a JMS `Connection` instance
- `createQueueSession(true, 0)`, which creates a `QueueSession`
- `createTopicSession(true, 0)`, which creates a `TopicSession`
- `setExceptionListener(ExceptionListener)`, which sets an exception listener for the connection

  This allows a client to be asynchronously notified of a problem. Some connections only consume messages, so they have no other way to learn the connection has failed.
- `getExceptionListener()`, which gets the `ExceptionListener` for this connection

JMS Session

A `Connection` is a factory for `Sessions` that use its underlying connection to a JMS provider for producing and consuming messages. 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 light-weight JMS object.

A `Session` serves several purposes:

- Constitutes a factory for its `MessageProducers` and `MessageConsumers`
- 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 this session's `MessageProducers` and `MessageConsumers`, organizing these into units
- Defines a serial order for the messages it consumes and the messages it produces
- Serializes execution of `MessageListeners` registered with it

In Oracle Database 10*g*, 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 the `MessageProducers` and `MessageConsumers` created by a `Session`.

Methods on the `Session` object include:

- `commit()`, which commits all messages performed in this transaction and releases locks currently held
- `rollback()`, which rolls back any messages accomplished in the transaction and release locks currently held
- `close()`, which closes the session
- `getDBConnection()`, which 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()`, which acknowledges message receipt in a nontransactional session
- `recover()`, which 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()` creates a **queue table**
- `getQueueTable()` gets a handle to an existing queue table
- `createQueue()` creates a queue
- `getQueue()` gets a handle to an existing queue
- `createTopic()` creates a topic
- `getTopic()` 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 Connection Examples

The following code illustrates how some of the preceding calls are used.

Example 11–3 JMS: Creating and Starting Queues and Queue Connections

```
public static void bol_example(String ora_sid, String host, int port,
                               String driver)
{

 QueueConnectionFactory    qc_fact   = null;
 QueueConnection           q_conn    = null;
 QueueSession              q_sess    = null;
 AQQueueTableProperty      qt_prop   = null;
 AQQueueTable              q_table   = null;
 AQjmsDestinationProperty  dest_prop = null;
 Queue                     queue     = null;
 BytesMessage              bytes_msg = null;

 try
```

```
{
  /* get queue connection factory */
  qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid,
                   port, driver);

  /* create queue connection */
  q_conn = qc_fact.createQueueConnection("boluser", "boluser");

  /* create QueueSession */
  q_sess = q_conn.createQueueSession(true, Session.CLIENT_ACKNOWLEDGE);

  /* start the queue connection */
  q_conn.start();

  qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");

  /* create a queue table */
  q_table = ((AQjmsSession)q_sess).createQueueTable("boluser",
                                                  "bol_ship_queue_table",
                                                  qt_prop);

  dest_prop = new AQjmsDestinationProperty();

  /* create a queue */
  queue = ((AQjmsSession)q_sess).createQueue(q_table, "bol_ship_queue",
                                          dest_prop);

  /* start the queue */
  ((AQjmsDestination)queue).start(q_sess, true, true);

  /* create a bytes message */
  bytes_msg = q_sess.createBytesMessage();

  /* close session */
  q_sess.close();

  /* close connection */
  q_conn.close();
}
catch (Exception ex)
{
  System.out.println("Exception: " + ex);
}
}
```

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 Streams AQ, these map to a *`schema.queue`* at a specific database. `Queue` maps to a single-consumer queue, and `Topic` maps to a multiconsumer queue.

`Destination` objects can be obtained in one of the following ways:

- Using a JMS Session to Obtain Destination Objects
- Using JNDI to Look Up Destination Objects

Using a JMS Session to Obtain Destination Objects

`Destination` objects are created from a `Session` object using domain-specific `Session` methods:

- `AQjmsSession.getQueue(queue_owner, queue_name)` gets a handle to a JMS queue
- `AQjmsSession.getTopic(topic_owner, topic_name)` 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 using the `DBMS_AQAQDM.add_alias_to_ldap` PL/SQL procedure.

Queues and topics that are registered in LDAP can be looked up through JNDI using the name or alias of the queue or topic.

JMS Destination Methods

Methods on the `Destination` object include:

- `alter()`, which alters a `Queue` or a `Topic`
- `schedulePropagation()`, which schedules **propagation** from a source to a destination

- `unschedulePropagation()`, which unschedules a previously scheduled propagation
- `enablePropagationSchedule()`, which enables a propagation schedule
- `disablePropagationSchedule()`, which disables a propagation schedule
- `start()`, which 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()`, which stops a `Queue` or a `Topic`

  The queue is stopped for enqueue or dequeue. The topic is stopped for publish or subscribe.

  `drop()`, which drops a `Queue` or a `Topic`

JMS Destination Examples

The following code illustrates how some of the preceding calls are used.

Example 11–4 JMS: Using JNDI to Lookup Destination Objects

Suppose we have a new orders queue `OE.OE_neworders_que` stored in LDA. It can be looked up as follows:

```
public static void get_Factory_from_LDAP() throws Exception
{
    Hashtable env = new Hashtable(5, 0.75f);
    env.put(Context.INITIAL_CONTEXT_FACTORY, AQjmsConstants.INIT_CTX_FACTORY);

    // aqldapserv is your LDAP host and 389 is your port
    env.put(Context.PROVIDER_URL, "ldap://aqldapserv:389);

    // now authentication information
    // username/password scheme, user is OE, password is OE
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "cn=oe,cn=users,cn=acme,cn=com");
    env.put(Context.SECURITY_CREDENTIALS, "oe");

    DirContext inictx = new InitialDirContext(env);
    // initialize context with the distinguished name of the database server
    inictx=(DirContext)inictx.lookup("cn=db1,cn=OracleContext,cn=acme,cn=com");

   // go to the destination holder
    DirContext destctx = (DirContext)inictx.lookup("cn=OracleDBQueues");
```

```
    // get the destination OE.OE_new_orders queue
    Queue myqueue = (Queue)destctx.lookup("cn=OE.OE_new_orders_que");

}
```

Example 11–5 JMS: Using JNDI to Perform Various Operations on a Destination Object

```
public static void setup_example(TopicSession t_sess)
{
  AQQueueTableProperty     qt_prop   = null;
  AQQueueTable             q_table   = null;
  AQjmsDestinationProperty dest_prop = null;
  Topic                    topic     = null;
  TopicConnection          t_conn    = null;

  try
  {
    qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_BYTES_MESSAGE");
    /* create a queue table */
    q_table = ((AQjmsSession)t_sess).createQueueTable("boluser",
                           "bol_ship_queue_table",
                           qt_prop);
    dest_prop = new AQjmsDestinationProperty();
    /* create a topic */
    topic = ((AQjmsSession)t_sess).createTopic(q_table, "bol_ship_queue",
                        dest_prop);

    /* start the topic */
    ((AQjmsDestination)topic).start(t_sess, true, true);

    /* schedule propagation from topic "boluser" to the destination
      dblink "dba" */
    ((AQjmsDestination)topic).schedulePropagation(t_sess, "dba", null,
                  null, null, null);
    /*
       some processing accomplished here
    */
    /* Unschedule propagation */
    ((AQjmsDestination)topic).unschedulePropagation(t_sess, "dba");
    /* stop the topic */
    ((AQjmsDestination)topic).stop(t_sess, true, true, true);
    /* drop topic */
```

```
    ((AQjmsDestination)topic).drop(t_sess);
    /* drop queue table */
    q_table.drop(true);
    /* close session */
    t_sess.close();
    /* close connection */
    t_conn.close();
  }
  catch(Exception ex)
  {
    System.out.println("Exception: " + ex);
  }
}
```

System-Level Access Control in JMS

Oracle8*i* 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 5-10

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 to 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

Oracle8*i* 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 must 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.

See Also: "Oracle Enterprise Manager Support" on page 5-10

Retention and Message History in JMS

Oracle Streams AQ allows users to retain messages in the queue table. This means that SQL can then be used to query these messages for analysis. 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. Along with retention and message identifiers, Oracle Streams AQ 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.

Example 11–6 JMS: Analyzing Retention and Message History in Oracle Streams AQ

Suppose that the shipping application must determine the average processing times of orders. This includes the time the order must wait in the backed_order topic. Specifying the retention as TRUE for the shipping queues and specifying the order number in the correlation field of the message, SQL queries can be written to determine the wait time for orders in the shipping application.

For simplicity, we analyze only orders that have already been processed. The processing time for an order in the shipping application is the difference between the enqueue time in the WS_bookedorders_topic and the enqueue time in the WS_shipped_orders_topic.

```
SELECT  SUM(SO.enq_time - BO.enq_time) / count (*) AVG_PRCS_TIME
   FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
   WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
   AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_topic';

/* Average waiting time in the backed order queue: */
SELECT SUM(BACK.deq_time - BACK.enq_time)/count (*) AVG_BACK_TIME
   FROM WS.AQ$WS_orders_mqtab BACK
   WHERE BACK.msg_state = 'PROCESSED' AND BACK.queue = 'WS_backorders_topic';
```

Supporting Oracle Real Application Clusters in JMS

Oracle Real Application Clusters can be used to improve Oracle Streams AQ 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 Streams AQ 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 Streams AQ 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.

See Also:

- "Scheduling a Queue Propagation" on page 8-32
- *Oracle Real Application Clusters Installation and Configuration Guide*

Supporting Statistics Views in JMS

Each instance keeps its own Oracle Streams AQ 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.

Example 11–7 JMS: Querying Oracle Streams AQ Statistics Views

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. The order processing application can use this to dynamically tune the number of order-processing processes.

```
CONNECT oe/oe

/* Count the number as messages and the average time for which the messages
   have been waiting: */
SELECT READY, AVERAGE_WAIT
FROM gv$aq Stats, user_queues Qs
WHERE Stats.qid = Qs.qid and Qs.Name = 'OE_neworders_que';
```

See Also: "Number of Messages in Different States for the Whole Database View" on page 9-17

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.

JMS Message Headers

A **JMS connection** can contain only a header; a message body is not required. The message header contains the following fields:

- `JMSDestination` contains the destination to which the message is sent. In Oracle Streams AQ this corresponds to the destination queue/topic.
- `JMSDeliveryMode` 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). Oracle Streams AQ supports persistent message delivery. JMS permits an administrator to configure JMS to override the client-specified value for `JMSDeliveryMode`.
- `JMSMessageID` uniquely identifies a message in a provider. All message IDs must begin with the string `ID:`.
- `JMSTimeStamp` contains the time the message was handed over to the provider to be sent. This maps to Oracle Streams AQ message enqueue time.

- `JMSCorrelationID` can be used by a client to link one message with another.
- `JMSReplyTo` contains a destination supplied by a client when a message is sent. Clients can use the following types to specify the `ReplyTo` destination: `oracle.jms.AQjmsAgent`; `javax.jms.Queue`; and `javax.jms.Topic`.
- `JMSType` contains a message type identifier supplied by a client at send time. For portability Oracle recommends that the `JMSType` be symbolic values.
- `JMSExpiration` 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 the JMS time stamp when the message was enqueued (Greenwich Mean Time, in milliseconds) and the TimeToLive (in milliseconds). JMS permits an administrator to configure JMS to override the client-specified value for `JMSExpiration`.
- `JMSPriority` contains the priority of the message. 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 Microsystem 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` is a Boolean set by the JMS provider.

See Also: "J2EE Compliance" on page 11-2

Table 11–2 lists the type and use of each JMS message header field and shows by whom it is set.

Table 11–2 JMS Message Header Fields

Message Header Field	Type	Set by	Use
`JMSDestination`	`Destination`	JMS after `Send` method has completed	Destination to which message is sent
`JMSDeliveryMode`	`int`	JMS after `Send` method has completed	Delivery mode (`PERSISTENT` or `NONPERSISTENT`)
`JMSExpiration`	`long`	JMS after Send method has completed	Expiration time can be specified for a message **producer** or can be explicitly specified during each send or publish
`JMSPriority`	`int`	JMS after `Send` method has completed	Message priority can be specified for a `MessageProducer` or can be explicitly specified during each send or publish

Table 11–2 (Cont.) JMS Message Header Fields

Message Header Field	Type	Set by	Use
`JMSMessageID`	`String`	JMS after `Send` method has completed	Uniquely identifies each message sent by the provider
`JMSTimeStamp`	`long`	JMS after `Send` method has completed	Time message is handed to provider to be sent
`JMSCorrelationID`	`String`	JMS client	Links one message with another
`JMSReplyTo`	`Destination`	JMS client	Destination where a reply to the message should be sent. It can be specified as `AQjmsAgent`, `javax.jms.Queue`, or `javax.jms.Topic` types
`JMSType`	`String`	JMS client	Message type identifier
`JMSRedelivered`	`Boolean`	JMS provider	Message probably was delivered earlier, but the client did not acknowledge it at that time

JMS Message Properties

Properties add optional header fields to a message. Properties allow a client, using message selectors, 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` is the identity of the user sending the message.
- `JMSXAppID` is the identity of the application sending the message.
- `JMSXDeliveryCount` is the number of message delivery attempts.
- `JMSXGroupid` is set by the client and refers to the identity of the message group that this message belongs to.
- `JMSXGroupSeq` is the sequence number of a message within a group.
- `JMSXRcvTimeStamp` is the time the message was delivered to the consumer (dequeue time).
- `JMSXState` is the message state, set by the provider. The message state can be `WAITING`, `READY`, `EXPIRED`, or `RETAINED`.

Table 11–3 lists the type and use of each JMS standard message property and shows by whom it is set.

Table 11–3 JMS Defined Message Properties

JMS Defined Message Property	Type	Set by	Use
`JMSXUserID`	`String`	JMS after `Send` method has completed	Identity of user sending message
`JMSAppID`	`String`	JMS after `Send` method has completed	Identity of application sending message
`JMSDeliveryCount`	`int`	JMS after `Receive` method has completed	Number of message delivery attempts
`JMSXGroupID`	`String`	JMS client	Identity of message group to which message belongs
`JMSXGroupSeq`	`int`	JMS client	Sequence number of message within group
`JMSXRcvTimeStamp`	`String`	JMS after `Receive` method has completed	Time that JMS delivered message to consumer
`JMSXState`	`int`	JMS provider	Message state set by provider

Oracle-specific JMS properties, which all begin with `JMS_Oracle`, include the following:

- `JMS_OracleExcpQ` is the queue name to send the message to if it cannot be delivered to the original destination. Only destinations of type `EXCEPTION` can be specified in the `JMS_OracleExcpQ` property.
- `JMS_OracleDelay` is the time in seconds to delay the delivery of the message. This can affect the order of message delivery.
- `JMS_OracleOriginalMessageId` is set to the message ID of the message in the source if the message is propagated from one destination to another. If the message is not propagated, then this property has the same value as the `JMSMessageId`.

A client can add additional header fields to a message by defining properties. These properties can then be used in message selectors to select specific messages.

JMS properties or header fields 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.

Table 11–4 Oracle Defined Message Properties

Header Field/Property	Type	Set by	Use
JMS_OracleExcpQ	String	JMS client	Specifies the name of the **exception queue**
JMS_OracleDelay	int	JMS client	Specifies the time (seconds) after which the message should become available to the consumers
JMS_OracleOriginalMessageID	String	JMS provider	Specifies the message ID of the message in source when the messages are propagated from one destination to another

JMS Message Body

JMS provides five forms of message body:

- **StreamMessage** - a message whose body contains a stream of Java primitive values. It is filled and read sequentially.
- **BytesMessage** - a message whose body contains a stream of uninterpreted bytes. This message type is for directly encoding a body to match an existing message format.
- **MapMessage** - a message whose body contains a set of name-value pairs. Names are strings and values are Java primitive types. The entries can be accessed sequentially by enumerator or randomly by name.
- **TextMessage** - a message whose body contains a `java.lang.String.`
- **ObjectMessage** - a message that contains a serializable Java object.
- **ADTmessage** - a message whose body contains an Oracle **object type** (AdtMessage type has been added in Oracle JMS).

See Also:

- Chapter 16, "Oracle Streams AQ JMS Types Examples"
- *PL/SQL Packages and Types Reference* JMS Types chapter.

The AQ$_JMS_MESSAGE Type

This type can store JMS messages of all the JMS-specified message types: `JMSStream`, `JMSBytes`, `JMSMap`, `JMSText`, and `JMSObject`. You can create a queue table of `AQ$_JMS_MESSAGE` type, but use any message type.

JMS Message Body: Stream Message

A StreamMessage is used to send a stream of Java primitives. It is filled and read sequentially. It inherits from Message and adds a stream message 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 Stream Messages, create the queue table with the `SYS.AQ$_JMS_STREAM_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Stream messages support the following conversion table. A value written as the row type can be read as the column type.

Table 11–5 Stream Message Conversion

Input	Boolean	byte	short	char	int	long	float	double	String	byte[]
`Boolean`	X	-	-	-	-	-	-	-	X	-
`byte`	-	X	X	-	X	X	-	-	X	-
`short`	-	-	X	-	X	X	-	-	X	-
`char`	-	-	-	X	-	-	-	-	X	-
`int`	-	-	-	-	X	X	-	-	X	-
`long`	-	-	-	-	-	X	-	-	X	-
`float`	-	-	-	-	-	-	X	X	X	-
`double`	-	-	-	-	-	-	-	X	X	-
`string`	X	X	X	X	X	X	X	X	X	-
`byte[]`	-	-	-	-	-	-	-	-	-	X

JMS Message Body: Bytes Message

A BytesMessage is used to send a message containing a stream of uninterpreted bytes. It inherits Message and adds a bytes message body. The receiver of the message supplies the interpretation of 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 Bytes Messages, create the queue table with `SYS.AQ$_JMS_BYTES_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

JMS Message Body: Map Message

A MapMessage is used to send a set of name-value pairs where names are Strings and 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 map message 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 Map Messages, create the queue table with the `SYS.AQ$_JMS_MAP_MESSAGE` or `AQ$_JMS_MESSAGE` payload types. Map messages support the following conversion table. A value written as the row type can be read as the column type.

Table 11–6 Map Message Conversion

Input	Boolean	byte	short	char	int	long	float	double	String	byte[]
`Boolean`	X	-	-	-	-	-	-	-	X	-
`byte`	-	X	X	-	X	X	-	-	X	-
`short`	-	-	X	-	X	X	-	-	X	-
`char`	-	-	-	X	-	-	-	-	X	-
`int`	-	-	-	-	X	X	-	-	X	-
`long`	-	-	-	-	-	X	-	-	X	-
`float`	-	-	-	-	-	-	X	X	X	-
`double`	-	-	-	-	-	-	-	X	X	-
`string`	X	X	X	X	X	X	X	X	X	-
`byte[]`	-	-	-	-	-	-	-	-	-	X

JMS Message Body: Text Message

A TextMessage is used to send a message containing a `java.lang.StringBuffer`. It inherits from `Message` and adds a text message body. The text information can be read or written using methods `getText()` and `setText(...)`. To use Text Messages, create the queue table with the `SYS.AQ$_JMS_TEXT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

JMS Message Body: Object Message

An ObjectMessage 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 Object Messages, create the queue table with the `SYS.AQ$_JMS_OBJECT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Example 11–8 JMS: Processing an ObjectMessage Body

```
public void enqueue_new_orders(QueueSession jms_session, BolOrder new_order)
{
   QueueSender   sender;
   Queue         queue;
   ObjectMessage obj_message;

   try
   {
       /* get a handle to the new_orders queue */
       queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
       sender = jms_session.createSender(queue);
       obj_message = jms_session.createObjectMessage();
       obj_message.setJMSCorrelationID("RUSH");
       obj_message.setObject(new_order);
       jms_session.commit();
    }
    catch (JMSException ex)
    {
      System.out.println("Exception: " + ex);
    }

}
```

JMS Message Body: AdtMessage

An AdtMessage 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, 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 to send messages to queues of type `SYS.XMLType`. You must use the `oracle.xdb.XMLType` class to create the message.

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`, the client can set:

- `JMSXAppID`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

For `AdtMessage`, 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`, the client can get:

- `JMSXuserID`
- `JMSXAppID`
- `JMSXDeliveryCount`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`
- `JMS_OracleOriginalMessageID`

For AdtMessage, the client can get:

- `JMSXDeliveryCount`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

The following JMS properties and header fields that can be included in a Message Selector. For `QueueReceiver`, `TopicSubscriber` and `TopicReceiver` on queues containing JMS type payloads, any SQL92 where clause of a string that contains:

- `JMSPriority (int)`
- `JMSCorrelationID (String)`
- `JMSMessageID (String)` - only for `QueueReceiver` and `TopicReceiver`
- `JMSTimestamp (Date)`
- `JMSType (String)`
- `JMSXUserID (String)`
- `JMSXAppID (String)`
- `JMSXGroupID (String)`
- `JMSXGroupSeq (int)`
- Any user-defined property in JMS message

For `QueueReceiver`, `TopicSubscriber` and `TopicReceiver` on queues containing Oracle object type payloads, use Oracle Streams AQ rule syntax for any SQL92 where clause of string that contains:

- `corrid`
- `priority`
- `tab.user_data.`*`adt_field_name`*

JMS Point-to-Point Model Features

- Queues
- QueueSender
- QueueReceiver
- QueueBrowser

Queues

In the point-to-point model, clients exchange messages using queues - from one point to another. These queues are used by message producers and consumers to send and receive messages.

An administrator creates single-consumer queues by means of the `createQueue` method in `AQjmsSession`. A client can obtain a handle to a previously created queue using the `getQueue` method on `AQjmsSession`.

These queues are described as **single-consumer queues** because a message can be consumed by only a single consumer. Put another way: a message can be consumed exactly once. This raises the question: What happens when there are multiple processes or operating system threads concurrently dequeuing from the same queue? Because a locked message cannot be dequeued by a process other than the one that has created the lock, each process dequeues the first unlocked message at the head of the queue.

Before using a queue, the queue must be enabled for enqueue/dequeue using the `start` call in `AQjmsDestination`.

After processing, the message is removed if the retention time of the queue is 0, or 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 using a `QueueBrowser` and specifying the message ID of the processed message.

QueueSender

A client uses a `QueueSender` to send messages to a queue. A `QueueSender` is created by passing a queue to a session's `createSender` method. 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. A `QueueReceiver` is created using the session's `createQueueReceiver` method. A `QueueReceiver` can be created with a message selector. This allows the client to restrict messages delivered to the consumer to those that match the selector.

The selector for queues containing payloads of type `TextMessage`, `StreamMessage`, `BytesMessage`, `ObjectMessage`, `MapMessage` can contain any expression that has a combination of one or more of the following:

- `JMSMessageID ='ID:23452345'` to retrieve messages that have a specified message ID (all message IDs being prefixed with ID:)
- 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
  ```

For queues containing `AdtMessages` the selector must be a SQL expression on the message payload contents or message ID or priority or `correlation ID`.

- Selector on message ID - to retrieve messages that have a specific message ID

  ```
  msgid = '23434556566767676'
  ```

  Note: in this case message IDs must NOT be prefixed with `ID:`
- Selector on priority or correlation is specified as follows

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

Example 11–9 Creating a JMS Connection and Session. Creating a Receiver to Receive Messages

In the BOL application, new orders are retrieved from the `new_orders_queue`. These orders are then published to the `OE.OE_bookedorders_topic`. After creating a JMS connection and session, you create a receiver to receive messages:

```
public void get_new_orders(QueueSession jms_session)
{
 QueueReceiver   receiver;
 Queue           queue;
 ObjectMessage   obj_message;
 BolOrder        new_order;
 BolCustomer     customer;
 String          state;
 String          cust_name;

 try
 {

   /* get a handle to the new_orders queue */
   queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

   receiver = jms_session.createReceiver(queue);

   for(;;)
   {
     /* wait for a message to show up in the queue */
     obj_message = (ObjectMessage)receiver.receive(10);

     new_order = (BolOrder)obj_message.getObject();

     customer = new_order.getCustomer();
     state    = customer.getState();

     obj_message.clearBody();

     /* determine customer region and assign a shipping region*/
  if((state.equals("CA")) || (state.equals("TX")) ||
   (state.equals("WA")) || (state.equals("NV")))
  obj_message.setStringProperty("Region", "WESTERN");
     else
  obj_message.setStringProperty("Region", "EASTERN");

     cust_name = new_order.getCustomer().getName();
```

```
            obj_message.setStringProperty("Customer", cust_name);

         if(obj_message.getJMSCorrelationID().equals("RUSH"))
         book_rush_order(obj_message);
         else
         book_new_order(obj_message);

            jms_session.commit();
          }
        }
        catch (JMSException ex)
        {
          System.out.println("Exception: " + ex);
        }
     }
```

QueueBrowser

A client uses a `QueueBrowser` to view messages on a queue without removing them. The browser methods return a `java.util.Enumeration` that is used to scan the queue's messages. The first call to `nextElement` gets a snapshot of the queue. 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.

A `QueueBrowser` can also be created with a message selector. This allows the client to restrict messages delivered to the browser to those that match the selector.

The selector for queues containing payloads of type `TextMessage`, `StreamMessage`, `BytesMessage`, `ObjectMessage`, `MapMessage` can contain any expression that has a combination of one or more of the following:

- `JMSMessageID ='ID:23452345'` to retrieve messages that have a specified message ID (all message IDs being prefixed with ID:)
- 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
  ```

For queues containing `AdtMessages` the selector must be a SQL expression on the message payload contents or messageID or priority or `correlationID`.

- Selector on message ID - to retrieve messages that have a specific messageID

  ```
  msgid = '23434556566767676'
  ```

  Note: in this case message IDs must NOT be prefixed with `ID:`.

- Selector on priority or correlation is specified as follows

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

JMS Publish/Subscribe Model Features

The following topics are discussed in this section:

- Topic
- Durable Subscriber
- TopicPublisher
- Recipient Lists
- TopicReceiver
- TopicBrowser

Topic

JMS has various features that allow you to develop an application based on a publish/subscribe model. The aim of this application model is to enable flexible and dynamic communication between applications functioning as publishers and applications playing the role of subscribers. The specific design point is that the applications playing these different roles should be decoupled in their communication.They should interact based on messages and message content.

In distributing messages, publisher applications are not required to explicitly handle or manage message recipients. This allows for the dynamic addition of new subscriber applications to receive messages without changing any publisher application logic. Subscriber applications receive messages based on message content without regard to which publisher applications are sending messages. This allows the dynamic addition of subscriber applications 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`.

You use the publish/subscribe model of communication in JMS by taking the following steps:

1. Enable enqueue/dequeue on the Topic using the start call in AQjmsDestination.
2. Set up one or more topics to hold messages. These topics should represent an area or subject of interest. For example, a topic can be used to represent billed orders.
3. Create a set of durable subscribers. Each subscriber can specify a selector that represents a specification (selects) for the messages that the subscriber wishes to receive. A null selector indicates that the subscriber wishes to receive all messages published on the topic.
4. 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 two local and remote topic.

   See Also: Chapter 8, "Oracle Streams AQ Administrative Interface" for details on propagation

5. Create `TopicPublishers` using the session's `createPublisher` method 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.
6. Subscribers can receive messages on the topic by using the `receive` method.
7. Subscribers can also receive messages asynchronously by using message listeners. The concepts of remote subscribers and propagation are Oracle extensions to JMS.

Durable Subscriber

Durable subscribers are instituted in either of the following ways:

- A client uses the session's `createDurableSubscriber` method to create durable subscribers.
- A `DurableSubscriber` is be created with a message selector. This allows the client to restrict messages delivered to the subscriber to those that match the selector.

The selector for topics containing payloads of type `TextMessage`, `StreamMessage`, `BytesMessage`, `ObjectMessage`, `MapMessage` can contain any expression that has a combination of one or more of the following:

- JMS message header fields or properties:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- User-defined message properties:

  ```
  color IN ('RED', BLUE', 'GREEN') AND price < 30000
  ```

For topics containing `AdtMessages` the selector must be a SQL expression on the message payload contents or priority or `correlationID`.

- Selector on priority or correlation is specified as follows

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

The syntax for the selector is described in detail in `createDurableSubscriber` in *Oracle Streams Advanced Queuing Java API Reference*.

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`.

Example 11–10 Creating Local and Remote Subscriber and Scheduling Propagation

```
public void create_bookedorders_subscribers(TopicSession jms_session)
{
   Topic             topic;
   TopicSubscriber   tsubs;
   AQjmsAgent        agt_east;
   AQjmsAgent        agt_west;

   try
   {

     /* get a handle to the OE_bookedorders_topic */
     topic = ((AQjmsSession)jms_session).getTopic("OE",
                   "OR_bookedorders_topic");


     /* Create local subscriber - to track messages for some customers */
     tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
             "JMSPriority < 3 AND Customer = 'MARTIN'",
                   false);

     /* Create remote subscribers in the western and eastern region */
     agt_west = new AQjmsAgent("West_Shipping", "WS.WS_bookedorders_topic");

     ((AQjmsSession)jms_session).createRemoteSubscriber(topic, agt_west,
                    "Region = 'WESTERN'");

     agt_east = new AQjmsAgent("East_Shipping", "ES.ES_bookedorders_topic");

     ((AQjmsSession)jms_session).createRemoteSubscriber(topic, agt_east,
                    "Region = 'EASTERN'");

     /* schedule propagation between bookedorders_topic and
   WS_bookedorders_topic, ES.ES_bookedorders_topic */
     ((AQjmsDestination)topic).schedulePropagation(jms_session,
```

```
                          "WS.WS_bookedorders_topic",
                             null, null, null, null);

     ((AQjmsDestination)topic).schedulePropagation(jms_session,
                          "ES.ES_bookedorders_topic",
                             null, null, null, null);
   }
   catch (Exception ex)
   {
     System.out.println("Exception " + ex);
   }

}
```

TopicPublisher

Messages are published using `TopicPublisher`:

A `TopicPublisher` is created by passing a Topic to a session's `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. The concept of recipient lists is an Oracle extension to JMS.

Example 11–11 JMS: Creating Recipient Lists for Specific Customers

Suppose we want to send high priority messages only to `SUBS1` and `Fedex_Shipping` in the Eastern region instead of publishing them to all the subscribers of the `OE_bookedorders_topic`:

```
public void book_rush_order(TopicSession jms_session,
             ObjectMessage obj_message)
{

   TopicPublisher  publisher;
```

```
   Topic           topic;
   AQjmsAgent[]    recp_list = new AQjmsAgent[2];

   try
   {
     /* get a handle to the bookedorders topic */
     topic = ((AQjmsSession) jms_session).getTopic("OE",
                     "OE_bookedorders_topic");

     publisher = jms_session.createPublisher(null);

     recp_list[0] = new AQjmsAgent("SUBS1", null);
     recp_list[1] = new AQjmsAgent("Fedex_Shipping",
               "ES.ES_bookedorders_topic");

     publisher.setPriority (1);
     ((AQjmsTopicPublisher)publisher).publish(topic, obj_message, recp_list);

     jms_session.commit();

   }
   catch (Exception ex)
   {
     System.out.println("Exception: " + ex);
   }
}
```

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.TopicReceiver` is an Oracle extension to JMS.

A `TopicReceiver` can also be created with a message selector. This allows the client to restrict messages delivered to the recipient to those that match the selector.

The syntax for the selector for `TopicReceiver` is the same as that for a `QueueReceiver`.

Example 11–12 JMS: Creating a Topic and Local Subscriber and Waiting for a Message to Show Up in the Topic

```
public void ship_rush_orders(TopicSession jms_session)
{
   Topic            topic;
   TopicReceiver    trec;
   ObjectMessage    obj_message;
   BolCustomer      customer;
   BolOrder         new_order;
   String           state;
   int              i = 0;

   try
   {
     /* get a handle to the OE_bookedorders_topic */
     topic = ((AQjmsSession)jms_session).getTopic("ES",
                    "ES_bookedorders_topic");

     /* Create local subscriber - to track messages for some customers */
     trec = ((AQjmsSession)jms_session).createTopicReceiver(topic,
                         "Fedex_Shipping",
                         null);

     /* process 10 messages */
     for(i = 0; i < 10; i++)
     {
       /* wait for a message to show up in the topic */
       obj_message = (ObjectMessage)trec.receive(10);

       new_order = (BolOrder)obj_message.getObject();

       customer = new_order.getCustomer();
       state    = customer.getState();

       System.out.println("Rush Order for customer " +
           customer.getName());
       jms_session.commit();
     }
   }
   catch (Exception ex)
   {
     System.out.println("Exception ex: " + ex);
   }
}
```

For remote subscribers - if the subscriber name at the remote topic has explicitly been specified in the `createRemoteSubscriber` call, then to receive a message, we can use `TopicReceivers`

```
public void get_westernregion_bookedorders(TopicSession jms_session)
{
   Topic            topic;
   TopicReceiver    trec;
   ObjectMessage    obj_message;
   BolCustomer      customer;
   BolOrder         new_order;
   String           state;
   int              i = 0;

   try
   {
     /* get a handle to the WS_bookedorders_topic */
     topic = ((AQjmsSession)jms_session).getTopic("WS",
                  "WS_bookedorders_topic");
     /* Create local subscriber - to track messages for some customers */
     trec = ((AQjmsSession)jms_session).createTopicReceiver(topic,
                       "West_Shipping",
                       null);
     /* process 10 messages */
     for(i = 0; i < 10; i++)
     {
       /* wait for a message to show up in the topic */
       obj_message = (ObjectMessage)trec.receive(10);

       new_order = (BolOrder)obj_message.getObject();

       customer = new_order.getCustomer();
       state    = customer.getState();

       System.out.println("Received Order for customer " +
           customer.getName());
       jms_session.commit();
     }
   }
   catch (Exception ex)
   {
     System.out.println("Exception ex: " + ex);
   }

}
```

If the subscriber name is not specified in the `createRemoteSubscriber` call, then clients must use durable subscribers at the remote site to receive messages.

TopicBrowser

A client uses a `TopicBrowser` to view messages on a topic without removing them. The browser methods return a `java.util.Enumeration` that is used to scan the topic's messages. The first call to `nextElement` gets a snapshot of the topic. A `TopicBrowser` 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.

A `TopicBrowser` can also be created with a message selector. This allows the client to restrict messages delivered to the browser to those that match the selector.

The selector for the `TopicBrowser` can take any of the following forms:

- `JMSMessageID ='ID:23452345'` to retrieve messages that have a specified message ID (all message IDs are prefixed with `ID:`)
- 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
  ```

For topics containing `AdtMessages,` the selector must be a SQL expression on the message payload contents or messageID or priority or `correlationID`.

- Selector on message ID - to retrieve messages that have a specific messageID

  ```
  msgid = '23434556566767676'
  ```

  Note: in this case message IDs must NOT be prefixed with `ID:`

  Selector on priority or correlation is specified as follows:

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows:

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

As with any consumer for topics, only durable subscribers are allowed to create a `TopicBrowser`.

`TopicBrowser` also 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`.

As with all other JMS messaging operations, 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.

JMS MessageProducer Features

- Priority and Ordering of Messages
- Time Specification - Delay
- Time Specification - Expiration
- Message Grouping

Priority and Ordering of Messages

The 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 (see "Creating a Queue Table" on page 8-2). Currently, Oracle Streams AQ supports ordering on two of the message attributes:

- Priority
- Enqueue time

When combined, they lead to four possible ways of ordering:

First-In, First-Out (FIFO) Ordering of Messages 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 Streams AQ at message publish/send time. This is also the default ordering.

Priority Ordering of Messages If priority ordering is 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 Ordering A FIFO-priority topic/queue can also be created by specifying both the priority and the enqueue time as the sort order of the messages. A FIFO-priority topic/queue acts like a priority queue, except if two messages are assigned the same priority, 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 Streams AQ does ensure that messages send/published in the same session with the same ordering criteria are received in the order they were sent.

Time Specification - 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 and the message becomes available. Delay for a message is specified as message property (JMS_OracleDelay). This property is not specified in the JMS standard. It is an Oracle Streams AQ extension to JMS message properties.

Delay processing requires the Oracle Streams AQ background process queue monitor to be started. Note also that receiving by `msgid` overrides the delay specification.

Time Specification - 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 Streams AQ 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 only be consumed by one consumer at a time. This requires the queue/topic be created in a queue table that is enabled for **transactional** message grouping (see "Creating a Queue Table" on page 8-2). 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. Using this feature, you can segment a complex message into simple messages. This is an Oracle Streams AQ extension and not part of the JMS specification.

For example, messages directed to a queue containing invoices could be constructed as a group of messages starting with the header message, followed by 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 general message properties (priority, delay, expiration) for the messages in a group are determined solely by the message properties specified for the first message (head) of the group irrespective of which properties are specified for subsequent messages in the group.

The message grouping property is preserved across propagation. However, the destination topic to which messages must be 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.

See Also: "Dequeue Features" on page 1-24

JMS Message Consumer Features

- Receiving Messages
- Message Navigation in Receive
- Browsing Messages
- Retry with Delay Interval
- Asynchronously Receiving Messages Using Message Listener
- Oracle Streams AQ Exception Handling

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

Example 11–13 JMS: Blocking Until a Message Arrives

```
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);
```

```
      }
      return new_order;

   }
```

Example 11–14 JMS: Blocking Messages for a Maximum of 60 Seconds

```
public BolOrder get_new_order2(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 60 seconds for a message to show up in the queue */
       obj_message = (ObjectMessage)qrec.receive(60000);

       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;

   }
```

Example 11–15 JMS: Nonblocking Messages

```
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;

   }
```

Message Navigation in Receive

When a consumer does the first receive in its session, its gets the first message in the queue or topic. Subsequent receives get the next message, and so on. The default action works well for FIFO queues and topics, but not for priority ordered queues. If a high priority message arrives for the consumer, then this client program 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 Streams AQ navigation modes are made available to it as JMS extensions. These modes can be set at the `TopicSubscriber`, `QueueReceiver` or the `TopicReceiver`.

- `FIRST_MESSAGE` resets the consumer's position to the beginning of the queue. This is a useful mode for priority ordered queues, because it allows the consumer to remove the message on the top of the queue.
- `NEXT_MESSAGE` gets the message after the established position of the consumer. For example, a `NEXT_MESSAGE` applied after the position is at the fourth message, will get the second message in the queue. This is the default action.

For transaction grouping

- `FIRST_MESSAGE` resets the consumer's position to the beginning of the queue.
- `NEXT_MESSAGE` sets the position to the next message in the same transaction.
- `NEXT_TRANSACTION` 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 in navigating through the queue, the program 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 the subsequent receive calls.

Example Scenario

The `get_new_orders()` procedure retrieves orders from the `OE_neworders_que`. Each transaction refers to an order, and each message corresponds to an individual book in that order. The `get_orders()` procedure loops through the messages to retrieve the book orders. It resets the position to the beginning of the queue using the `FIRST_MESSAGE` option before the first receive. It then uses the `NEXT_MESSAGE` navigation option to retrieve the next book (message) of an order (transaction). If it gets an exception indicating all messages in the current group/transaction have been fetched, then it changes the navigation option to `NEXT_TRANSACTION` and gets the first book of the next order. It then changes the navigation option back to `NEXT_MESSAGE` for fetching subsequent messages in the same transaction. This is repeated until all orders (transactions) have been fetched.

Example 11–16 JMS: Navigating the Retrieval of Messages

```
public void get_new_orders(QueueSession jms_session)
    {
      Queue              queue;
      QueueReceiver      qrec;
      ObjectMessage      obj_message;
      BolCustomer        customer;
      BolOrder           new_order;
      String             state;
      int                new_orders = 1;

      try
      {

         /* get a handle to the new_orders queue */
         queue = ((AQjmsSession) jms_session).getQueue("OE","OE_neworders_que");
         qrec = jms_session.createReceiver(queue);

    /* set navigation to first message */

((AQjmsTopicSubscriber)qrec).setNavigationMode(AQjmsConstants.NAVIGATION_FIRST_
MESSAGE);

        while(new_orders != 0)
        {
          try{

             /* wait for a message to show up in the topic */
             obj_message = (ObjectMessage)qrec.receiveNoWait();

             if (obj_message != null)   /* no more orders in the queue */
             {
               System.out.println(" No more orders ");
               new_orders = 0;
```

```
            }
            new_order = (BolOrder)obj_message.getObject();
            customer = new_order.getCustomer();
            state    = customer.getState();

            System.out.println("Order: for customer " +
                               customer.getName());

           /* Now get the next message */

((AQjmsTopicSubscriber)qrec).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_
MESSAGE);

          }catch(AQjmsException ex)
          {  if (ex.getErrorNumber() == 25235)
             {
               System.out.println("End of transaction group");

((AQjmsTopicSubscriber)qrec).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_
TRANSACTION);
             }
             else
               throw ex;
          }
        }
     }catch (JMSException ex)
     {
        System.out.println("Exception: " + ex);
      }
   }
```

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. After a message has been browsed, there is no guarantee that the message will be available to the JMS session again, because a receive call from a concurrent session might remove the message.

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 Streams AQ extension to the JMS interface.The lock on the message with a browser with locked mode is released when the session performs a commit or a rollback.

To remove the message viewed by a QueueBrowser, the session must create a QueueReceiver and use the JMSmesssageID as the selector.

Example Code

See "QueueBrowser" on page 11-33.

Remove-No-Data

The MessageConsumer can remove the message from the queue or topic without retrieving the message using the receiveNoData call. This is useful when the application has already examined the message, perhaps using the QueueBrowser. This mode allows the JMS client to avoid the overhead of retrieving the payload from the database, which can be substantial for a large message.

Retry with Delay Interval

If the transaction receiving the message from a queue/topic fails, then it is regarded as an unsuccessful attempt to remove the message. Oracle Streams AQ records the number of failed attempts to remove the message in the message history.

In addition, it also allows the application to specify the maximum number of retries supported on messages at the queue/topic level. If the number of failed attempts to remove a message exceed this maximum, then the message is moved to the exception queue and is no longer available to applications.

The transaction receiving a message could have terminated due to a bad condition. For example, an order could not be fulfilled because there were insufficient books in stock. Because inventory updates are made every twelve hours, it makes sense to retry after that time. If an order is still not filled after four attempts, then there could be a problem serious enough for the order to move to the exception queue.

Oracle Streams AQ 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 Streams AQ background process time manager enforces the retry delay property.

The maximum retries and retry delay are properties of the queue/topic which can be set when the queue/topic is created or using the alter method on the queue/topic. The default value for MAX_RETRIES is 5.

Example 11–17 JMS: Specifying Max Retries and Max Delays in Messages

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.

```
public BolOrder process_booked_order(TopicSession jms_session)
  {
    Topic            topic;
    TopicSubscriber  tsubs;
    ObjectMessage    obj_message;
    BolCustomer      customer;
    BolOrder         booked_order = null;
    String           country;
    int              i = 0;

    try
    {
      /* 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();
          booked_order = null;
       }
    }catch (JMSException ex)
```

```
  { System.out.println("Exception " + ex) ;}

   return booked_order;
 }
```

Asynchronously Receiving Messages Using Message Listener

The JMS client can receive messages asynchronously by setting the `MessageListener` using the `setMessageListener` method available with the Consumer.

When a message arrives for the message 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 the consumers of 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.

Example 11–18 Asynchronous receipt of queue messages

The application processing the new orders queue can be set up for asynchronously receiving messages from the queue.

```
public class OrderListener implements MessageListener
  {
     QueueSession   the_sess;

     /* constructor */
     OrderListener(QueueSession  my_sess)
     {
       the_sess = my_sess;
     }

     /* message listener interface */
     public void onMessage(Message m)
     {
       ObjectMessage    obj_msg;
       BolCustomer      customer;
       BolOrder         new_order = null;

       try {
```

```
          /* cast to JMS Object Message */
          obj_msg = (ObjectMessage)m;

          /*  Print some useful information */
          new_order = (BolOrder)obj_msg.getObject();
          customer = new_order.getCustomer();
          System.out.println("Order:  for customer " +  customer.getName());

          /* call the process order method
          * NOTE: we are assuming it is defined elsewhere
          * /
          process_order(new_order);
          /* commit the asynchronous receipt of the message */
           the_sess.commit();
        }catch (JMSException ex)
        { System.out.println("Exception " + ex) ;}

    }
}

  public void setListener1(QueueSession jms_session)
  {
    Queue             queue;
    QueueReceiver     qrec;
    MessageListener   ourListener;

    try
    {
     /* get a handle to the new_orders queue */
     queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

     /* create a QueueReceiver */
     qrec = jms_session.createReceiver(queue);

     /* create the message listener */
     ourListener = new OrderListener(jms_session);

     /* set the message listener for the receiver */
     qrec.setMessageListener(ourListener);
    }
    catch (JMSException ex)
    {
      System.out.println("Exception: " + ex);
    }
}
```

Oracle Streams AQ Exception Handling

Oracle Streams AQ provides four integrated mechanisms to support exception handling in applications: `EXCEPTION_QUEUES`, `EXPIRATION`, `MAX_RETRIES` and `RETRY_DELAY`.

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 a provider (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. If the queue/topic is created in a queue table, say QTAB, then the default exception queue is called `AQ$_QTAB_E`. The default exception queue is automatically created when the queue table is created.

Messages are moved to the exception queues by Oracle Streams AQ under the following conditions:

- The message is not being 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`. If the `timeToLive` was not specified for the message, (either in the `publish` or `send` call, or as the publisher or sender), then it never expires.
- The message was received successfully, but the application terminates 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. 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. When 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.

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.

> **Note:** A message is moved to an exception queue if `RETRY_COUNT` is greater than `MAX_RETRIES`. 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 client program successfully received a message but terminated before committing the transaction.

JMS Propagation

- Remote Subscribers
- Scheduling Propagation
- Enhanced Propagation Scheduling Capabilities
- Exception Handling During Propagation

Remote Subscribers

This feature enables applications to communicate with each other without having to be connected to the same database.

Oracle Streams AQ allows a remote subscriber, that is a subscriber at another database, to subscribe to a topic. When a message published to the topic meets the criterion of the remote subscriber, Oracle Streams AQ automatically propagates the message to the queue/topic at the remote database specified for the remote subscriber.

The snapshot (`job_queue`) background process performs propagation. Propagation is performed using database links and Oracle Net Services.

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 (`dblink`) to the database of the subscriber.

There are two kinds of remote subscribers:

Case 1

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.

Case 2

Specify a specific remote recipient 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

Example 11–19 JMS: Creating Remote Subscribers

- Scenario for Case 1. Assume the order entry application and Western region shipping application are on different databases, `db1` and `db2`. Further assume that there is a database link `dblink_oe_ws` from database `db1`, the order entry database, to the western shipping database db2. The `WS_bookedorders_topic` at db2 is a remote subscriber to the `OE_bookedorders_topic` in db1.
- Scenario for Case 2. Assume the order entry application and Western region shipping application are on different databases, db1 and db2. Further assume that there is a database link `dblink_oe_ws` from the local order entry database `db1` to the western shipping database `db2`. The agent "Priority" at `WS_bookedorders_topic` in `db2` is a remote subscriber to the `OE_bookedorders_topic` in db1. Messages propagated to the `WS_bookedorders_topic` are for "Priority" only.

```
public void remote_subscriber(TopicSession jms_session)
   {
     Topic            topic;
     ObjectMessage    obj_message;
     AQjmsAgent       remote_sub;

    try
```

```
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("OE",
                                                   "OE_bookedorders_topic");
      /* create the remote subscriber, name unspecified  and address
       * the topic WS_bookedorders_topic at db2
       */
      remote_sub = new AQjmsAgent(null, "WS.WS_bookedorders_topic@dblink_oe_
ws");

      /* subscribe for western region orders */
      ((AQjmsSession)jms_session).createRemoteSubscriber(topic, remote_sub,
"Region = 'Western' ");
    }
    catch (JMSException ex)
    { System.out.println("Exception :" + ex); }
    catch (java.sql.SQLException  ex1)
    {System.out.println("SQL Exception :" + ex1); }
  }
```

Database db2 - shipping database: The `WS_bookedorders_topic` has two subscribers, one for priority shipping and the other normal. The messages from the Order Entry database are propagated to the Shipping database and delivered to the correct subscriber. Priority orders have a message priority of 1.

```
public void  get_priority_messages(TopicSession jms_session)
   {
     Topic            topic;
     TopicSubscriber  tsubs;
     ObjectMessage    obj_message;
     BolCustomer      customer;
     BolOrder         booked_order;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("WS",
                                                   "WS_bookedorders_topic");

       /* Create local subscriber - for priority messages */
      tsubs = jms_session.createDurableSubscriber(topic, "PRIORITY",
                                       " JMSPriority = 1 ", false);

      obj_message = (ObjectMessage) tsubs.receive();
```

```
      booked_order = (BolOrder)obj_message.getObject();
      customer = booked_order.getCustomer();
      System.out.println("Priority Order:  for customer " +
customer.getName());

      jms_session.commit();
    }
    catch (JMSException ex)
    { System.out.println("Exception :" + ex); }
  }

  public void  get_normal_messages(TopicSession jms_session)
   {
     Topic            topic;
     TopicSubscriber  tsubs;
     ObjectMessage    obj_message;
     BolCustomer      customer;
     BolOrder         booked_order;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("WS",
                                             "WS_bookedorders_topic");

       /* Create local subscriber - for priority messages */
      tsubs = jms_session.createDurableSubscriber(topic, "PRIORITY",
                                 " JMSPriority > 1 ", false);

      obj_message = (ObjectMessage) tsubs.receive();

      booked_order = (BolOrder)obj_message.getObject();
      customer = booked_order.getCustomer();
      System.out.println("Normal Order:  for customer " +  customer.getName());

      jms_session.commit();
    }
    catch (JMSException ex)
    { System.out.println("Exception :" + ex); }
  }


public void remote_subscriber1(TopicSession jms_session)
   {
     Topic            topic;
```

```
     ObjectMessage    obj_message;
     AQjmsAgent       remote_sub;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("OE",
                                             "OE_bookedorders_topic");
      /* create the remote subscriber, name "Priority"  and address
       * the topic WS_bookedorders_topic at db2
       */
      remote_sub = new AQjmsAgent("Priority", "WS.WS_bookedorders_topic@dblink_
oe_ws");

      /* subscribe for western region orders */
      ((AQjmsSession)jms_session).createRemoteSubscriber(topic, remote_sub,
"Region = 'Western' ");
    }
    catch (JMSException ex)
    { System.out.println("Exception :" + ex); }
    catch (java.sql.SQLException  ex1)
    {System.out.println("SQL Exception :" + ex1); }
  }


   Remote database:
   database db2 - Western Shipping database.
/* get messages for subscriber priority */
   public void  get_priority_messages1(TopicSession jms_session)
   {
     Topic            topic;
     TopicReceiver    trecs;
     ObjectMessage    obj_message;
     BolCustomer      customer;
     BolOrder         booked_order;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("WS",
                                             "WS_bookedorders_topic");

      /* create a local receiver "Priority" for the remote subscription
       * to WS_bookedorders_topic
       */
```

```
      trecs = ((AQjmsSession)jms_session).createTopicReceiver(topic, "Priority",
null);

      obj_message = (ObjectMessage) trecs.receive();

      booked_order = (BolOrder)obj_message.getObject();
      customer = booked_order.getCustomer();
      System.out.println("Priority Order:  for customer " +
customer.getName());

      jms_session.commit();
    }
    catch (JMSException ex)
    { System.out.println("Exception :" + ex); }
  }
```

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, load at source database, load at 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 Queue Propagation" on page 8-32

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.

Example 11–20 JMS: Scheduling Propagation

```
 public void  schedule_propagation(TopicSession jms_session)
  {
     Topic            topic;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("WS",
                                             "WS_bookedorders_topic");

      /* Schedule propagation immediately with duration of 5 minutes and latency
20 sec */
      ((AQjmsDestination)topic).schedulePropagation(jms_session, "dba", null,
                                       new Double(5*60), null, new Double(20));
    }catch (JMSException ex)
    {System.out.println("Exception: " + ex);}
  }

  Propagation schedule parameters can also be altered.


  /* alter duration to 10 minutes and latency to zero */
  public void  alter_propagation(TopicSession jms_session)
  {
     Topic            topic;
    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("WS",
                                             "WS_bookedorders_topic");

      /* Schedule propagation immediately with duration of 5 minutes and latency
20 sec */
    ((AQjmsDestination)topic).alterPropagationSchedule(jms_session, "dba",
```

```
                        new Double(10*60), null, new Double(0));
    }catch (JMSException ex)
    {System.out.println("Exception: " + ex);}
  }
```

Enhanced Propagation Scheduling Capabilities

Detailed information about the schedules can be obtained from the catalog views defined for propagation. Information about active schedules—such as the name of the background process handling that schedule, the SID (session, serial number) for the session handling the propagation and the Oracle Database instance handling a schedule (relevant if Real Application Clusters are being used)—can be obtained from the catalog views. The same catalog views also provide information about the previous successful execution of a schedule (last successful propagation of message) and the next execution of the schedule.

For each schedule, detailed propagation statistics are maintained:

- 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

These statistics have been designed to provide useful information to the queue administrators for tuning the schedules such that maximum efficiency can be achieved.

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. 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 failure 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 errors such as a network failure occurs, Oracle Streams AQ continues to attempt to propagate messages using an exponential back-off algorithm. In some situations that indicate application errors Oracle Streams AQ marks messages as `UNDELIVERABLE` if there is an error in propagating the message.

Examples of such errors are when the remote queue/topic does not exist or when there is a type mismatch between the source queue/topic and the remote queue/topic.In such situations users must query the `DBA_SCHEDULES` view to determine the last error that occurred during propagation to a particular destination.The trace files in the `$ORACLE_HOME`/log directory can provide additional information about the error.

Message Transformation with JMS AQ

The following topics are discussed in this section:

- Defining Message Transformations
- Sending Messages to a Destination Using a Transformation
- Receiving Messages from a Destination Using a Transformation
- Specifying Transformations for Topic Subscribers
- Specifying Transformations for Remote Subscribers

Defining Message Transformations

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 PLSQL functions.

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 Remote Subscriber. This enables propagation of messages between Topics of different formats.

The Message Transformation feature is an Oracle Streams AQ extension to the standard JMS interface.

Sending Messages to a Destination Using a Transformation

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.

See Also: *PL/SQL Packages and Types Reference*

Example 11–21 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.

```
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);
       }
}
```

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 `setTransformation()` interface of the `AQjmsQueueReceiver`, `AQjmsTopicSubscriber` and `AQjmsTopicReceiver`.

Example 11–22 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`:

```
public AQjmsAdtMessage retrieve_bookedorders(TopicSession jms_session)
       AQjmsTopicReceiver  receiver;
       Topic               topic;
       Message             msg = null;

       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;
}
```

Specifying Transformations for Topic Subscribers

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` already exists, then its transformation is set to the specified transformation.

Example 11–23 JMS: 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`:

```
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;
}
```

Specifying Transformations for Remote Subscribers

Oracle Streams AQ 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`. 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 Streams AQ 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 Streams AQ applies the transformation to the message before propagating it to the queue/topic at the remote database.

Example 11–24 JMS: 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 WSOrder java class has been generated by Jpublisher to map to the Oracle object `WS.WS_order`

```
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, WSOrder.getFactory(),"OE2WS");
     }
     catch (JMSException ex)
     {
       System.out.println("Exception :" ex);
     }
}
```

12

Oracle Streams AQ JMS Interface: Basic Operations

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 Queue/Topic Connection Factory
- Unregistering a Queue/Topic Connection Factory
- Getting a Queue/Topic Connection Factory
- Getting a Queue/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 Queue/Topic Connection Factory

You can register a **queue**/topic **connection factory** 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

Purpose

Registers a queue/topic connection factory through the database with JDBC connection parameters to a **Lightweight Directory Access Protocol** (LDAP) server.

Syntax

```
public static int registerConnectionFactory(java.sql.Connection connection,
                                            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
```

Parameters

connection

JDBC connection used in registration.

conn_name
Name of the connection to be registered.

hostname
Name of the host running Oracle Streams AQ.

oracle_sid
Oracle system identifier.

portno
Port number.

driver
Type of **JDBC driver**.

type
QUEUE or TOPIC.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the database connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, look up the connection factory using **Java Naming and Directory Interface** (JNDI).

Example

```
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

Purpose

Registers a queue/topic connection factory through the database with a JDBC URL to LDAP.

Syntax

```
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
```

Parameters

connection

JDBC connection used in registration.

conn_name

Name of the connection to be registered.

jdbc_url

URL to connect to.

info

Properties information.

type

QUEUE or TOPIC.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the database connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, look up the connection factory using JNDI.

Example

```
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

Purpose

Registers a queue/topic connection factory through LDAP with JDBC connection parameters to LDAP.

Syntax

```
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
```

Parameters

env
Environment of LDAP connection.

conn_name
Name of the connection to be registered.

hostname
Name of the host running Oracle Streams AQ.

oracle_sid
Oracle system identifier.

portno
Port number.

driver

Type of JDBC driver.

type

QUEUE or TOPIC.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, 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

```
Hashtable            env = new Hashtable(5, 0.75f);
/* 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

Purpose

Registers a queue/topic connection factory through LDAP with JDBC connection parameters to LDAP.

Syntax

```
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
```

Parameters

env
Environment of LDAP connection.

conn_name
Name of the connection to be registered.

jdbc_url
URL to connect to.

info
Properties information.

type
QUEUE or TOPIC.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, 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

```
String              url;
Hashtable           env = new Hashtable(5, 0.75f);

/* 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");
url = "jdbc:oracle:thin:@sun-123:1521:db1";
AQjmsFactory.registerConnectionFactory(env, "topic_conn1", url, null, "topic");
```

Unregistering a Queue/Topic Connection Factory

You can unregister a queue/topic connection factory in LDAP two ways:

- Unregistering Through the Database
- Unregistering Through LDAP

Unregistering Through the Database

Purpose

Unregisters a queue/topic connection factory in LDAP.

Syntax

```
public static int unregisterConnectionFactory(java.sql.Connection connection,
                                              java.lang.String conn_name)
                                       throws JMSException
```

Parameters

connection
JDBC connection used in registration.

conn_name
Name of the connection to be unregistered.

Usage Notes

`unregisterConnectionFactory` is a static method. To successfully unregister the connection factory, the database connection passed to `unregisterConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`.

Example

```
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

Purpose

Unregisters a queue/topic connection factory in LDAP.

Syntax

```
public static int unregisterConnectionFactory(java.util.Hashtable env,
                                              java.lang.String conn_name)
                                       throws JMSException
```

Parameters

env
Environment of LDAP connection.

conn_name
Name of the connection to be unregistered.

Usage Notes

`unregisterConnectionFactory` is a static method. To successfully unregister the connection factory, 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

```
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 Queue/Topic Connection Factory

This section contains these topics:

- Getting a Queue Connection Factory with JDBC URL
- Getting a Queue Connection Factory with JDBC Connection Parameters
- Getting a Topic Connection Factory with JDBC URL
- Getting a Topic Connection Factory with JDBC Connection Parameters
- Getting a Queue/Topic Connection Factory in LDAP

Getting a Queue Connection Factory with JDBC URL

Purpose

Gets a queue connection factory with JDBC URL.

Syntax

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String jdbc_url,
              java.util.Properties info)
       throws JMSException
```

Parameters

jdbc_url

URL to connect to.

info

Properties information.

Usage Notes

getQueueConnectionFactory is a static method.

Example

```
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 Queue Connection Factory with JDBC Connection Parameters

Purpose

Gets a queue connection factory with JDBC connection parameters.

Syntax

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String hostname,
              java.lang.String oracle_sid,
```

```
                int portno,
                java.lang.String driver)
        throws JMSException
```

Parameters

hostname
Name of the host running Oracle Streams AQ.

oracle_sid
Oracle system identifier.

portno
Port number.

driver
Type of JDBC driver.

Usage Notes

getQueueConnectionFactory is a static method.

Example

```
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 Topic Connection Factory with JDBC URL

Purpose

Gets a topic connection factory with a JDBC URL.

Syntax

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String jdbc_url,
              java.util.Properties info)
       throws JMSException
```

Parameters

jdbc_url
URL to connect to.

info
Properties information.

Usage Notes

getTopicConnectionFactory is a static method.

Example

```
String      url          = "jdbc:oracle:oci10:internal/oracle"
Properties  info         = new Properties();
TopicConnectionFactory   tc_fact;

info.put("internal_logon", "sysdba");
tc_fact = AQjmsFactory.getTopicConnectionFactory(url, info);
```

Getting a Topic Connection Factory with JDBC Connection Parameters

Purpose

Gets a topic connection factory with JDBC connection parameters.

Syntax

```
public static javax.jms.TopicConnectionFactory getTopicConnectionFactory(
              java.lang.String hostname,
              java.lang.String oracle_sid,
              int portno,
              java.lang.String driver)
       throws JMSException
```

Parameters

hostname
Name of the host running Oracle Streams AQ.

oracle_sid
Oracle system identifier.

portno
Port number.

driver
Type of JDBC driver.

Usage Note

getTopicConnectionFactory is a Static Method.

Example

```
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 Queue/Topic Connection Factory in LDAP

Purpose

Gets a queue/topic connection factory from LDAP.

Example

```
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");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aquser1,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
```

```
ctx = new InitialDirContext(env);
ctx =
(DirContext)ctx.lookup("cn=OracleDBConnections,cn=db1,cn=Oraclecontext,cn=acme,c
n=com");
qc_fact = (queueConnectionFactory)ctx.lookup("cn=queue_conn1");
```

Getting a Queue/Topic in LDAP

Purpose

Gets a queue/topic from LDAP.

Example

```
Hashtable              env = new Hashtable(5, 0.75f);
DirContext             ctx;
topic                  topic_1;

/* 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(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aquser1,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");

ctx = new InitialDirContext(env);
ctx =
(DirContext)ctx.lookup("cn=OracleDBQueues,cn=db1,cn=Oraclecontext,cn=acme,cn=com
");
topic_1 = (topic)ctx.lookup("cn=topic_1");
```

Creating a Queue Table

Purpose

Creates a **queue table**.

Syntax

```
public oracle.AQ.AQQueueTable createQueueTable(
                 java.lang.String owner,
                 java.lang.String name,
                 oracle.AQ.AQQueueTableProperty property)
          throws JMSException
```

Parameters

owner

Queue table owner (**schema**)

name

Queue table name

property

Queue table properties. 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.

Usage Notes

CLOB, **BLOB**, and **BFILE** objects are valid attributes for an Oracle Streams AQ **object type** load. However, only CLOB and BLOB can be propagated using Oracle Streams AQ **propagation** in Oracle8*i* and after.

Example

```
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

Purpose

Gets a queue table.

Syntax

```
public oracle.AQ.AQQueueTable getQueueTable(java.lang.String owner,
                                            java.lang.String name)
                                     throws JMSException
```

Parameters

owner
Queue table owner (schema)

name
Queue table name

Usage Notes

If the caller that opened the connection is not the owner of the queue table, then the caller must have Oracle Streams AQ **enqueue**/**dequeue** privileges on queues/topics in the queue table. Otherwise the queue-table is not returned.

Example

```
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

Purpose

Creates a queue in a specified queue table.

Syntax

```
public javax.jms.Queue createQueue(
             oracle.AQ.AQQueueTable q_table,
             java.lang.String queue_name,
             oracle.jms.AQjmsDestinationProperty dest_property)
      throws JMSException
```

Parameters

q_table
Queue table in which the queue is to be created. The queue table must not be multiconsumer enabled.

queue_name
Name of the queue to be created.

dest_property
Queue properties.

Usage Notes

The queue table in which a queue is created must be a single-consumer queue table.

Example

```
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

Purpose

Creates a topic in the **publish/subscribe** model.

Syntax

```
public javax.jms.Topic createTopic(
             oracle.AQ.AQQueueTable q_table,
             java.lang.String topic_name,
             oracle.jms.AQjmsDestinationProperty dest_property)
      throws JMSException
```

Parameters

q_table
Queue table in which the queue is to be created. The queue table must be multiconsumer enabled.

queue_name
Name of the queue to be created.

dest_property
Queue properties.

Example

```
TopicSession              t_sess;
AQQueueTable              q_table;
AqjmsDestinationProperty dest_prop;
Topic                     topic;

topic = ((AQjmsSessa)t_sess).createTopic(q_table, "jms_t1", dest_prop);
```

Granting and Revoking Privileges

This section contains these topics:

- Granting Oracle Streams AQ System Privileges
- Revoking Oracle Streams AQ System Privileges
- Granting Publish/Subscribe Topic Privileges
- Revoking Publish/Subscribe Topic Privileges
- Granting Point-to-Point Queue Privileges
- Revoking Point-to-Point Queue Privileges

Granting Oracle Streams AQ System Privileges

Purpose

Grants Oracle Streams AQ system privileges to a user/roles.

Syntax

```
public void grantSystemPrivilege(java.lang.String privilege,
                                 java.lang.String grantee,
                                 boolean admin_option)
                          throws JMSException
```

Parameters

privilege

`ENQUEUE_ANY`, `DEQUEUE_ANY` or `MANAGE_ANY`.

grantee

Specifies the grantee. The grantee can be a user, role or the PUBLIC role.

admin_option

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.

Usage Notes

The privileges are `ENQUEUE_ANY`, `DEQUEUE_ANY` and `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

```
TopicSession             t_sess;

((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "scott", false);
```

Revoking Oracle Streams AQ System Privileges

Purpose
Revokes Oracle Streams AQ system privileges from user/roles.

Syntax
```
public void revokeSystemPrivilege(java.lang.String privilege,
                                  java.lang.String grantee)
                           throws JMSException
```

Parameters

privilege
ENQUEUE_ANY, DEQUEUE_ANY or MANAGE_ANY.

grantee
Specifies the grantee. The grantee can be a user, role or the PUBLIC role.

Usage Notes
The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY. 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
```
TopicSession             t_sess;

((AQjmsSession)t_sess).revokeSystemPrivilege("ENQUEUE_ANY", "scott");
```

Granting Publish/Subscribe Topic Privileges

Purpose
Grants a topic privilege in the publish/subscribe model.

Syntax

```
public void grantTopicPrivilege(javax.jms.Session session,
                                java.lang.String privilege,
                                java.lang.String grantee,
                                boolean grant_option)
                         throws JMSException
```

Parameters

session
JMS session.

privilege
Privilege being granted. The options are ENQUEUE, DEQUEUE, or ALL. ALL means both.

grantee
Database user being granted the privilege.

grant_option
If set to true, then the grantee can grant the privilege to other users.

Usage Notes
Initially only the queue table owner can use this procedure to grant privileges on the topic.

Example

```
TopicSession             t_sess;
Topic                    topic;

((AQjmsDestination)topic).grantTopicPrivilege(t_sess,
                                              "ENQUEUE",
                                              "scott",
                                               false);
```

Revoking Publish/Subscribe Topic Privileges

Purpose
Revokes a topic privilege in the publish/subscribe model.

Syntax

```
public void revokeTopicPrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee)
                          throws JMSException
```

Parameters

session
JMS session.

privilege
The privilege being revoked. The options are ENQUEUE, DEQUEUE, or ALL. ALL means both.

grantee
Database user from whom the privilege is being revoked.

Example

```
TopicSession              t_sess;
Topic                     topic;

((AQjmsDestination)topic).revokeTopicPrivilege(t_sess, "ENQUEUE", "scott");
```

Granting Point-to-Point Queue Privileges

Purpose
Grants a queue privilege in the point-to-point model.

Syntax

```
public void grantQueuePrivilege(javax.jms.Session session,
                                java.lang.String privilege,
                                java.lang.String grantee,
                                boolean grant_option)
                         throws JMSException
```

Parameters

session
JMS session.

privilege
The privilege being granted. The options are `ENQUEUE`, `DEQUEUE`, or `ALL`. `ALL` means both.

grantee
Database user being granted the privilege.

grant_option
If set to true, then the grantee can grant the privilege to other users.

Usage Notes

Initially only the queue table owner can use this procedure to grant privileges on the queue.

Example

```
QueueSession              q_sess;
Queue                     queue;

((AQjmsDestination)queue).grantQueuePrivilege(q_sess,
                                              "ENQUEUE",
                                              "scott",
                                               false);
```

Revoking Point-to-Point Queue Privileges

Purpose

Revokes queue privilege in the point-to-point model.

Syntax

```
public void revokeQueuePrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee)
                          throws JMSException
```

Parameters

session
JMS session.

privilege
The privilege being revoked. The options are ENQUEUE, DEQUEUE, or ALL. ALL means both.

grantee
Database user from whom the privilege is being revoked.

Usage Notes

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 grantors privilege is also revoked.

Example

```
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

Purpose

Starts a destination.

Syntax

```
public void start(javax.jms.Session session,
                  boolean enqueue,
                  boolean dequeue)
           throws JMSException
```

Parameters

session
JMS session

enqueue
Determines whether enqueue should be enabled or not.

dequeue
Determines whether dequeue should be enabled or not.

Usage Notes

After creating a destination, the administrator must use the start method to enable the destination. If `enqueue` is set to `TRUE`, then the destination is enabled for enqueue. If `enqueue` is set to `FALSE`, then the destination is disabled for enqueue. Similarly, if `dequeue` is set to `TRUE`, then the destination is enabled for dequeue. If `dequeue` is set to `FALSE`, then the destination is disabled for dequeue.

Example

```
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

Purpose

Stops a destination.

Syntax

```
public void stop(javax.jms.Session session,
                 boolean enqueue,
                 boolean dequeue,
                 boolean wait)
          throws JMSException
```

Parameters

session
JMS session.

enqueue
If set to true, then enqueue is disabled.

dequeue
If set to true, then dequeue is disabled.

wait
If set to true, then pending transactions on the queue/topic are allowed to complete before the destination is stopped

Usage Notes

If `dequeue` is set to `TRUE`, then the destination is disabled for dequeue. If `dequeue` is set to `FALSE`, then the current setting is not altered. Similarly, if `enqueue` is set to `TRUE`, then the destination is disabled for enqueue. If `enqueue` is set to `FALSE`, then the current setting is not altered.

Example

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).stop(t_sess, true, false);
```

Altering a Destination

Purpose

Alters a destination.

Syntax

```
public void alter(javax.jms.Session session,
                  oracle.jms.AQjmsDestinationProperty dest_property)
           throws JMSException
```

Parameters

session
JMS session.

dest_property
New properties of the queue or topic.

Example

```
QueueSession q_sess;
Queue        queue;
TopicSession t_sess;
Topic        topic;

AQjmsDestionationProperty dest_prop1, dest_prop2;

((AQjmsDestination)queue).alter(dest_prop1);
((AQjmsDestination)topic).alter(dest_prop2);
```

Dropping a Destination

Purpose

Drops a destination.

Syntax

```
public void drop(javax.jms.Session session)
          throws JMSException
```

Parameters

session
JMS session.

Example

```
QueueSession q_sess;
Queue        queue;
TopicSession t_sess;
Topic        topic;

((AQjmsDestionation)queue).drop(q_sess);
```

```
((AQjmsDestionation)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
- Unscheduling a Propagation

Scheduling a Propagation

Purpose

Schedules a propagation.

Syntax

```
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
```

Parameters

session

JMS session

destination

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.

start_time

Time propagation must be started.

duration
Duration of propagation.

next_time
Next time propagation must be accomplished.

latency
Latency in seconds that can be tolerated. Latency is the difference between the time a message was enqueued and the time it was propagated.

Usage Notes

Messages can be propagated to other topics in the same database by specifying a NULL destination. If the **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.

Example

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).schedulePropagation(t_sess,
                                              null,
                                              null,
                                              null,
                                              null,
                                              new Double(0));
```

Enabling a Propagation Schedule

Purpose

Enables a propagation schedule.

Syntax

```
public void enablePropagationSchedule(javax.jms.Session session,
                                      java.lang.String destination)
                               throws JMSException
```

Parameters

session
JMS session

destination
Database link of the destination database.

Usage Notes

NULL destination indicates that the propagation is to the local database.

Example

```
TopicSession             t_sess;
Topic                    topic;

 ((AQjmsDestination)topic).enablePropagationSchedule(t_sess, "dbs1");
```

Altering a Propagation Schedule

Purpose

Alters a propagation schedule.

Syntax

```
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
```

Parameters

session
JMS session

destination
Database link of the destination database.

duration
The new duration.

next_time
The new next time for propagation.

latency
The new latency.

Usage Notes

NULL destination indicates that the propagation is to the local database

Example

```
TopicSession             t_sess;
Topic                    topic;

 ((AQjmsDestination)topic).alterPropagationSchedule(t_sess,
                                                    null,
                                                    30,
                                                    null,
                                                    new Double(30));
```

Disabling a Propagation Schedule

Purpose

Disables a propagation schedule.

Syntax

```
public void disablePropagationSchedule(javax.jms.Session session,
                                       java.lang.String destination)
                                throws JMSException
```

Parameters

session
JMS session

destination
Database link of the destination database.

Usage Notes

NULL destination indicates that the propagation is to the local database

Example

```
TopicSession            t_sess;
Topic                   topic;

 ((AQjmsDestination)topic).disablePropagationSchedule(t_sess, "dbs1");
```

Unscheduling a Propagation

Purpose

Unschedules a previously scheduled propagation.

Syntax

```
public void unschedulePropagation(javax.jms.Session session,
                                  java.lang.String destination)
                           throws JMSException
```

Parameters

session

JMS session

destination

Database link of the destination database.

Example

```
TopicSession   t_sess;
Topic          topic;

((AQjmsDestination)topic).unschedulePropagation(t_sess, "dbs1");
```

13

Oracle Streams AQ JMS Operational Interface: Point-to-Point

This chapter describes the Oracle Streams Advanced Queuing (AQ) **Java Message Service** (JMS) operational interface for basic point-to-point operations.

This chapter contains these topics:

- Creating a Connection
- Creating a Queue Connection
- Creating a Session
- Creating a QueueSession
- Creating a QueueSender
- Sending Messages
- Creating a QueueBrowser
- Creating a QueueReceiver

Creating a Connection

A JMS `Connection` supports both point-to-point and publish/subscribe operations. The methods in this section are new and support JMS version 1.1 specifications.

This section contains these topics:

- Creating a Connection with Username/Password
- Creating a Connection with Default Connection Factory Parameters

Creating a Connection with Username/Password

Purpose

Creates a connection with username and password.

Syntax

```
public javax.jms.Connection createConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

Parameters

username

Name of the user connecting to the database for queuing.

password

Password for creating the connection to the server.

Usage Notes

This connection supports both point-to-point and publish/subscribe operations.

Creating a Connection with Default Connection Factory Parameters

Purpose

Creates a connection with default **connection factory** parameters.

Syntax

```
public javax.jms.Connection createConnection()
      throws JMSException
```

Usage Notes

The ConnectionFactory properties must contain a default username and password; otherwise, this method throws a JMSException. This connection supports both point-to-point and publish/subscribe operations.

Creating a Queue Connection

This section contains these topics:

- Creating a Queue Connection with Username/Password
- Creating a Queue Connection with an Open JDBC Connection
- Creating a Queue Connection with Default Connection Factory Parameters
- Creating a Queue Connection with an Open OracleOCIConnection Pool

Creating a Queue Connection with Username/Password

Purpose

Creates a **queue** connection with username and password.

Syntax

```
public javax.jms.QueueConnection createQueueConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

Parameters

username

Name of the user connecting to the database for queuing.

password

Password for creating the connection to the server.

Example

```
QueueConnectionFactory qc_fact =
AQjmsFactory.getQueueConnectionFactory("sun123", "oratest", 5521, "thin");
/* Create a queue connection using a username/password */
QueueConnection qc_conn = qc_fact.createQueueConnection("jmsuser", "jmsuser");
```

Creating a Queue Connection with an Open JDBC Connection

Purpose

Creates a queue connection with an open JDBC connection.

Syntax

```
public static javax.jms.QueueConnection
createQueueConnection(java.sql.Connection jdbc_connection)
               throws JMSException
```

Parameters

jdbc_connection

Valid open connection to the database.

Usage Notes

This is a static method.

Example 1

This method 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 use the supplied JDBC connection to create the JMS QueueConnection object.

```
Connection db_conn;     /* previously opened JDBC connection */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(
          db_conn);
```

Example 2

This method is the only way to create a JMS QueueConnection when using JMS from java stored procedures inside the database (JDBC Server driver)

```
OracleDriver ora = new OracleDriver();
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(ora.defaultConnection());
```

Creating a Queue Connection with Default Connection Factory Parameters

Purpose

Creates a queue connection with default **connection factory** parameters.

Syntax

```
public javax.jms.QueueConnection createQueueConnection()
      throws JMSException
```

Usage Notes

The QueueConnectionFactory properties must contain a default username and password: otherwise, this method throws a JMSException.

Creating a Queue Connection with an Open OracleOCIConnection Pool

Purpose

Creates a queue connection with an open `OracleOCIConnectionPool`.

Syntax

```
public static javax.jms.QueueConnection createQueueConnection(
          oracle.jdbc.pool.OracleOCIConnectionPool cpool)
   throws JMSException
```

Parameters

cpool

Valid open connection OCI connection pool to the database.

Usage notes

This is a static method.

Example

This method can be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS does not open an new `OracleOCIConnectionPool` instance, but instead uses the supplied `OracleOCIConnectionPool` instance to create the JMS QueueConnection object.

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(cpool);
```

Creating a Session

Purpose

Creates a `Session`, which supports both point-to-point and publish/subscribe operations.

Syntax

```
public javax.jms.Session createSession(boolean transacted,
                                                 int ack_mode)
                                          throws JMSException
```

Parameters

transacted

If set to true, then the session is **transactional**.

ack_mode

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`.

Usage Notes

This method is new and supports JMS version 1.1 specifications. Transactional and **nontransactional** sessions are supported.

Creating a QueueSession

Purpose

Creates a `QueueSession`.

Syntax

```
public javax.jms.QueueSession createQueueSession(boolean transacted,
                                                  int ack_mode)
                                           throws JMSException
```

Parameters

transacted

If set to true, then the session is **transactional**.

ack_mode

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`.

Usage Notes

Transactional and **nontransactional** sessions are supported.

Example

For a transactional session:

```
QueueConnection qc_conn;
QueueSession  q_sess = qc_conn.createQueueSession(true, 0);
```

Creating a QueueSender

Purpose

Creates a `QueueSender`.

Syntax

```
public javax.jms.QueueSender createSender(javax.jms.Queue queue)
                                   throws JMSException
```

Usage Notes

If a sender is created without a default queue, then the destination queue must be specified on every **send** operation.

Sending Messages

This section contains these topics:

- Sending Messages Using a QueueSender with Default Send Options
- Sending Messages Using a QueueSender by Specifying Send Options

Sending Messages Using a QueueSender with Default Send Options

Purpose

Sends a **message** using a `QueueSender` with default send options.

Syntax

```
public void send(javax.jms.Queue queue,
                 javax.jms.Message message)
          throws JMSException
```

Parameters

queue
Queue to send this message to.

message
Message to send.

Usage Notes

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.

This send operation uses default values for message `priority` (`1`) and `timeToLive` (`infinite`).

Example 1

```
/* 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 2

```
/* 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

Purpose

Sends messages using a `QueueSender` by specifying send options.

Syntax

```
public void send(javax.jms.Queue queue,
                 javax.jms.Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive)
          throws JMSException
```

Parameters

queue

Queue to send this message to.

message

Message to send.

deliveryMode

Delivery mode to use.

priority
Priority for this message.

timeToLive
Message lifetime (in milliseconds).

Usage Notes

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 1

```
/* 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 2

```
/* 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

You can create a QueueBrowser for:

- Queues with Text, Stream, Objects, Bytes or Map Messages
- Queues with Text, Stream, Objects, Bytes, Map Messages, Locking Messages
- Queues of Oracle Object Type Messages
- Queues of Oracle Object Type Messages, Locking Messages

Queues with Text, Stream, Objects, Bytes or Map Messages

Purpose

Creates a QueueBrowser for queues with text, stream, objects, bytes or map messages.

Syntax

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector)
                                     throws JMSException
```

Parameters

queue

Queue to access.

messageSelector

Only messages with properties matching the message selector expression are delivered.

Usage Notes

To retrieve messages that match certain criteria, the selector for the QueueBrowser can be any expression that has a combination of one or more of the following:

- `JMSMessageID = 'ID:23452345'` to retrieve messages that have a specified message ID
- **JMS message** header fields or properties:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- User-defined message properties:

```
color IN ('RED', BLUE', 'GREEN') AND price < 30000
```

All message IDs must be prefixed with "ID:"

Use methods in `java.util.Enumeration` to go through list of messages.

Example 1

```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
browser = jms_session.createBrowser(queue);
```

Example 2

```
/* 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'");
```

Queues with Text, Stream, Objects, Bytes, Map Messages, Locking Messages

Purpose

Creates a `QueueBrowser` for queues with text, stream, objects, bytes or map messages, locking messages while browsing.

Syntax

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            boolean locked)
                                     throws JMSException
```

Parameters

queue

Queue to access.

messageSelector
Only messages with properties matching the message selector expression are delivered.

locked
If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE).

Usage Notes

Locked messages cannot be removed by other consumers until the browsing session ends the transaction.

Example 1

```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
browser = jms_session.createBrowser(queue, null, true);
```

Example 2

```
/* 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);
```

Queues of Oracle Object Type Messages

Purpose

Creates a `QueueBrowser` for queues of Oracle **object type** messages.

Syntax

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            java.lang.Object payload_factory)
                                     throws JMSException
```

Parameters

queue
Queue to access.

messageSelector
Only messages with properties matching the message selector expression are delivered.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

For queues containing `AdtMessages` the selector for the `QueueBrowser` can be a SQL expression on the message payload contents or messageID or priority or `correlationID`.

- Selector on message ID - to retrieve messages that have a specific messageID

  ```
  msgid = '23434556566767676'
  ```

  Note: in this case message IDs must NOT be prefixed with `ID:`

- Selector on priority or correlation is specified as follows

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

Example

The CustomDatum factory 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.

```
/* Create a browser for a Queue with Adt messages of type EMPLOYEE*/
QueueSession jms_session
QueueBrowser browser;
Queue        test_queue;
browser = ((AQjmsSession)jms_session).createBrowser(test_queue,
                                                    "corrid='EXPRESS'",
                                                     Employee.getFactory());
```

Queues of Oracle Object Type Messages, Locking Messages

Purpose

Creates a `QueueBrowser` for queues of Oracle object type messages, locking messages while browsing.

Syntax

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            java.lang.Object payload_factory,
                                            boolean locked)
                                     throws JMSException
```

Parameters

queue
Queue to access.

messageSelector
Only messages with properties matching the message selector expression are delivered.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

Note: CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

---

locked
If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE).

Example
```
/* Create a browser for a Queue with Adt messages of type EMPLOYEE* in lock
mode/
QueueSession jms_session
QueueBrowser browser;
Queue        test_queue;
browser = ((AQjmsSession)jms_session).createBrowser(test_queue,
                                                     null,
                                                     Employee.getFactory(),
                                                     true);
```

Creating a QueueReceiver

You can create a QueueReceiver for:

- Queues with Text, Stream, Objects, Bytes or Map Messages
- Queues of Oracle Object Type Messages

Queues of Standard JMS Type Messages

Purpose
Creates a QueueReceiver for queues of standard JMS type messages.

Syntax
```
public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
                                              java.lang.String messageSelector)
                                       throws JMSException
```

Parameters

queue
Queue to access.

messageSelector
Only messages with properties matching the message selector expression are delivered.

Usage Notes

The selector for the QueueReceiver can be any expression that has a combination of one or more of the following:

- `JMSMessageID = 'ID:23452345'` to retrieve messages that have a specified message ID. All message IDs must be prefixed with "ID:"
- JMS message header fields or properties:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- User-defined message properties:

  ```
  color IN ('RED', BLUE', 'GREEN') AND price < 30000
  ```

Example 1

```
/* Create a receiver without a selector */
QueueSession    jms_session
QueueReceiver   receiver;
Queue           queue;
receiver = jms_session.createReceiver(queue);
```

Example 2

```
/* Create a receiver for queues with a specified selector */
QueueSession    jms_session;
QueueReceiver   receiver;
Queue           queue;
/* create Receiver to receive messages with correlationID starting with EXP  */
browser = jms_session.createReceiver(queue, "JMSCorrelationID LIKE 'EXP%'");
```

Queues of Oracle Object Type Messages

Purpose

Creates a QueueReceiver for queues of Oracle object type messages.

Syntax

```
public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
                                              java.lang.String messageSelector,
                                              java.lang.Object payload_factory)
                                       throws JMSException
```

Parameters

queue
Queue to access.

messageSelector
Only messages with properties matching the message selector expression are delivered.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

The CustomDatum factory for a particular java class that maps to the SQL object type payload can be obtained using the `getFactory` static method.

For queues containing `AdtMessages` the selector for the QueueReceiver can be a SQL expression on the message payload contents or messageID or priority or `correlationID`.

- Selector on message ID - to retrieve messages that have a specific messageID. In this case message IDs must NOT be prefixed with `ID:`

  ```
  msgid = '23434556566767676'
  ```

- Selector on priority or correlation is specified as follows

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

Example

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.

```
/* Create a receiver for a Queue with Adt messages of type EMPLOYEE*/
QueueSession jms_session
QueueReceiver receiver;
Queue        test_queue;
browser = ((AQjmsSession)jms_session).createReceiver(
                 test_queue,
                "JMSCorrelationID = 'MANAGER',
                 Employee.getFactory());
```

14

Oracle Streams AQ JMS Operational Interface: Publish/Subscribe

This chapter describes the **Java Message Service** (JMS) **publish/subscribe** operational interface to Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- Creating a Connection
- Creating a TopicConnection
- Creating a Session
- Creating a TopicSession
- Creating a TopicPublisher
- Publishing a Message
- Creating a Durable Subscriber
- Creating a Remote Subscriber
- Unsubscribing a Durable Subscription
- Creating a TopicReceiver
- Creating a TopicBrowser
- Browsing Messages Using a TopicBrowser

Creating a Connection

A JMS `Connection` supports both point-to-point and publish/subscribe operations. The methods in this section are new and support JMS version 1.1 specifications.

This section contains these topics:

- Creating a Connection with Username/Password
- Creating a Connection with Default Connection Factory Parameters

Creating a Connection with Username/Password

Purpose

Creates a connection with username and password.

Syntax

```
public javax.jms.Connection createConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

Parameters

username

Name of the user connecting to the database for queuing.

password

Password for creating the connection to the server.

Usage Notes

This connection supports both point-to-point and publish/subscribe operations.

Creating a Connection with Default Connection Factory Parameters

Purpose

Creates a connection with default **connection factory** parameters.

Syntax

```
public javax.jms.Connection createConnection()
      throws JMSException
```

Usage Notes

The ConnectionFactory properties must contain a default username and password; otherwise, this method throws a JMSException. This connection supports both point-to-point and publish/subscribe operations.

Creating a TopicConnection

This section contains these topics:

- Creating a TopicConnection with Username/Password
- Creating a TopicConnection with Open JDBC Connection
- Creating a TopicConnection with Default Connection Factory Parameters
- Creating a TopicConnection with an Open OracleOCIConnectionPool

Creating a TopicConnection with Username/Password

Purpose

Creates a `TopicConnection` with username/password.

Syntax

```
public javax.jms.TopicConnection createTopicConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

Parameters

username

Name of the user connecting to the database for queuing.

password

Password for the user creating the connection.

Example

```
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

Purpose

Creates a `TopicConnection` with open JDBC connection.

Syntax

```
public static javax.jms.TopicConnection createTopicConnection(
              java.sql.Connection jdbc_connection)
       throws JMSException
```

Parameters

jdbc_connection

Valid open connection to the database.

Example 1

```
Connection db_conn;   /*previously opened JDBC connection */
TopicConnection tc_conn = AQjmsTopicConnectionFactory createTopicConnection(db_
conn);
```

Example 2

```
OracleDriver ora = new OracleDriver();
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(ora.defaultConnection());
```

Creating a TopicConnection with Default Connection Factory Parameters

Purpose

Creates a `TopicConnection` with default **connection factory** parameters.

Syntax

```
public javax.jms.TopicConnection createTopicConnection()
         throws JMSException
```

Creating a TopicConnection with an Open OracleOCIConnectionPool

Purpose

Creates a `TopicConnection` with an open `OracleOCIConnectionPool`.

Syntax

```
public static javax.jms.TopicConnection createTopicConnection(
              oracle.jdbc.pool.OracleOCIConnectionPool cpool)
       throws JMSException
```

Parameters

cpool
Valid open connection to the database.

Usage notes

This is a static method.

Example

This method can be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS does not open an new `OracleOCIConnectionPool` instance, but instead use the supplied `OracleOCIConnectionPool` instance to create the JMS `TopicConnection` object.

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(cpool);
```

Creating a Session

Purpose

Creates a `Session`, which supports both point-to-point and publish/subscribe operations.

Syntax

```
public javax.jms.Session createSession(boolean transacted,
                                       int ack_mode)
                                throws JMSException
```

Parameters

transacted

If set to `true`, then the session is transactional.

ack_mode

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`.

Usage Notes

This method is new and supports JMS version 1.1 specifications.

Creating a TopicSession

Purpose

Creates a `TopicSession`.

Syntax

```
public javax.jms.TopicSession createTopicSession(boolean transacted,
                                                 int ack_mode)
                                          throws JMSException
```

Parameters

transacted

If set to true, then the session is transactional.

ack_mode

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`.

Example

```
TopicConnection tc_conn;
TopicSession t_sess = tc_conn.createTopicSession(true,0);
```

Creating a TopicPublisher

Purpose

Creates a `TopicPublisher`.

Syntax

```
public javax.jms.TopicPublisher createPublisher(javax.jms.Topic topic)
                                         throws JMSException
```

Parameters

topic

Topic to publish to, or null if this is an unidentified **producer**.

Publishing a Message

You can publish a **message** using a:

- TopicPublisher with Minimal Specification
- TopicPublisher and Specifying Correlation and Delay
- TopicPublisher and Specifying Priority and TimeToLive
- TopicPublisher and Specifying a Recipient List Overriding Topic Subscribers

TopicPublisher with Minimal Specification

Purpose

Publishes a message with minimal specification.

Syntax

```
public void publish(javax.jms.Message message)
             throws JMSException
```

Parameters

message
Message to publish.

Usage Notes

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 in the **send** operation, 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. The `TopicPublisher` uses the default values for message `priority` (1) and `timeToLive` (`infinite`).

Example 1

```
/* 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);
```

Example 2

```
/* Publish without 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");
/* create TopicSession */
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* get shipped orders topic */
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE",
          "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* create text message */
TextMessage     jms_sess.createTextMessage();
/* publish without specifying the topic */
publisher1.publish(text_message);
```

TopicPublisher and Specifying Correlation and Delay

Purpose

Publishes a message specifying correlation and delay.

Syntax

```
public void publish(javax.jms.Message message)
             throws JMSException
```

Parameters

message

Message to publish.

Usage Notes

The publisher can set the message properties like delay and correlation before publishing.

Example

```
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 text message */
TextMessage     jms_sess.createTextMessage();
/* Set correlation and delay */
/* Set correlation */
jms_sess.setJMSCorrelationID("FOO");
/* Set delay of 30 seconds */
jms_sess.setLongProperty("JMS_OracleDelay", 30);
/* Publish  */
publisher1.publish(text_message);
```

TopicPublisher and Specifying Priority and TimeToLive

Purpose

Publishes a message specifying priority and `TimeToLive`.

Syntax

```
public void publish(javax.jms.Topic topic,
                    javax.jms.Message message,
```

```
                oracle.jms.AQjmsAgent[] recipient_list,
                int deliveryMode,
                int priority,
                long timeToLive)
         throws JMSException
```

Parameters

topic
Topic to which to publish the message. This overrides the default topic of the `MessageProducer`.

message
Message to be published.

recipient_list
List of recipients to which the message is published. Recipients are of type AQjmsAgent.

deliveryMode
Delivery mode. The options are `PERSISTENT` or `NON_PERSISTENT`, but only `PERSISTENT` is supported in this release.

priority
Priority of the message.

timeToLive
Message time to live in milliseconds; zero is unlimited.

Usage Notes

Message `priority` and `timeToLive` can be specified with the publish call. The only delivery mode supported for this release is `PERSISTENT`.

Example

```
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 text message */
TextMessage     jms_sess.createTextMessage();
/* Publish  message with priority 1 and time to live 200 seconds */
publisher1.publish(text_message, DeliveryMode.PERSISTENT, 1, 200000);
```

TopicPublisher and Specifying a Recipient List Overriding Topic Subscribers

Purpose

Publishes a message specifying a **recipient** list overriding topic subscribers.

Syntax

```
public void publish(javax.jms.Message message,
                    oracle.jms.AQjmsAgent[] recipient_list)
             throws JMSException
```

Parameters

message

The message to be published.

recipient_list

The list of recipients to which the message is published. The recipients are of type AQjmsAgent.

Usage Notes

The subscription list of the topic can be overridden by specifying the recipient list with the publish call.

Example

```
/* 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(
          "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 text message */
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 Durable Subscriber

`CreateDurableSubscriber` requires exclusive access to the topics. If there are pending JMS send, publish, or receive operations on the same topic when this call is applied, then exception ORA - 4020 is raised. There are two solutions to the problem:

- Limit calls to `createDurableSubscriber` at 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`.

This section contains these topics:

- Creating a Durable Subscriber for a JMS Topic Without Selector
- Creating a Durable Subscriber for a JMS Topic With Selector
- Creating a Durable Subscriber for an Oracle Object Type Topic Without Selector
- Creating a Durable Subscriber for an Oracle Object Type Topic With Selector

Creating a Durable Subscriber for a JMS Topic Without Selector

Purpose

Creates a durable **subscriber** for a **JMS topic** without selector.

Syntax

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name)
      throws JMSException
```

Parameters

topic

Non-temporary topic to subscribe to.

subs_name

Name used to identify this subscription.

Usage Notes

The subscriber name and JMS topic must be specified to create a durable subscriber. An unsubscribe call ends the subscription to the topic.

Example

```
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 Durable Subscriber for a JMS Topic With Selector

Purpose

Creates a durable subscriber for a JMS topic with selector.

Syntax

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name,
             java.lang.String messageSelector,
             boolean noLocal)
      throws JMSException
```

Parameters

topic

Non-temporary topic to subscribe to.

subs_name

Name used to identify this subscription.

messageSelector
Only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

noLocal
If set to true, then it inhibits the delivery of messages published by its own connection.

Usage Notes

The client creates a durable subscriber by specifying a subscriber name and JMS topic. Optionally, a message selector can be specified. Only messages with properties matching the message selector expression are delivered to the subscriber. The selector value can be null. The selector can contain any SQL92 expression that has a combination of one or more of the following:

For example:

```
JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
```

- **JMS message** header fields or properties:
  - JMSPriority (int)
  - JMSCorrelationID (string)
  - JMSType (string)
  - JMSXUserID (string)
  - JMSXAppID (string)
  - JMSXGroupID (string)
  - JMSXGroupSeq (int)
- User-defined message properties

  For example:

  ```
  color IN ('RED', BLUE', 'GREEN') AND price < 30000
  ```

Operators allowed are:

- Logical operators in precedence order `NOT`, `AND`, `OR` comparison operators
- `=`, `>`, `>=`, `<`, `<=`, `<>`, `!` (both `<>` and `!` can be used for not equal)
- Arithmetic operators in precedence order `+`, `-` unary, `*`, `/`, `+`, `-`

- Identifier [NOT] IN (string-literal1, string-literal2, ..)
- Arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3
- Identifier [NOT] LIKE pattern-value [ESCAPE escape-character]

  Pattern-value is a string literal where % refers to any sequence of characters and _ refers to any single character. The optional escape-character is used to escape the special meaning of the '_' and '%' in pattern-value
- Identifier IS [NOT] NULL

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a different message selector. An unsubscribe call is needed to end the subscription to the topic.

Example

```
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 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 Durable Subscriber for an Oracle Object Type Topic Without Selector

Purpose

Creates a durable subscriber for an Oracle **object type** topic without selector.

Syntax

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name,
             java.lang.Object payload_factory)
      throws JMSException
```

Parameters

topic

Non-temporary topic to subscribe to.

subs_name

Name used to identify this subscription.

payload_factory

CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

To create a durable subscriber for a topic of Oracle object type, the client must specify the CustomDatumFactory for the Oracle object type in addition to the topic and subscriber name.

Example

```
/* 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, specifying the correct CustomDatumFactory */
subscriber1 = jms_sess.createDurableSubscriber(
          shipped_orders,
         'WesternShipping',
          AQjmsAgent.getFactory());
```

Creating a Durable Subscriber for an Oracle Object Type Topic With Selector

Purpose

Creates a durable subscriber for an Oracle object type topic with selector.

Syntax

```
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
```

Parameters

topic

Non-temporary topic to subscribe to.

subs_name
Name used to identify this subscription.

messageSelector
Only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

noLocal
If set to true, then it inhibits the delivery of messages published by its own connection.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

To create a durable subscriber for a Topic of Oracle object type, the client must specify the CustomDatumFactory for the Oracle object type in addition to the topic and subscriber name.

Optionally, a message selector can be specified. Only messages matching the selector are delivered to the subscriber.

Oracle object type messages do not contain any user-defined properties. However, the selector can be used to select messages based on priority or correlation ID or attribute values of the message payload

The syntax for the selector for queues containing Oracle object type messages is different from the syntax for selectors on queues containing standard JMS payloads (text, stream, object, bytes, map).

The selector is similar to the Oracle Streams AQ **rules** syntax. An example of a selector on priority or correlation is:

```
priority > 3 AND corrid = 'Fiction'
```

An example of a selector on message payload is:

```
tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
```

The attribute name must be prefixed with `tab.user_data`.

Example

```
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());
```

Creating a Remote Subscriber

This section contains these topics:

- Creating a Remote Subscriber for Topics of JMS Messages
- Creating a Remote Subscriber for Topics of Oracle Object Type Messages

Creating a Remote Subscriber for Topics of JMS Messages

Purpose

Creates a remote subscriber for topics of JMS messages without selector.

Syntax

```
public void createRemoteSubscriber(javax.jms.Topic topic,
                                   oracle.jms.AQjmsAgent remote_subscriber,
                                   java.lang.String messageSelector)
                            throws JMSException
```

Parameters

topic

Topic to subscribe to.

remote_subscriber

AQjmsAgent that refers to the remote subscriber.

messageSelector

Only messages with properties matching the message selector expression are delivered. This value can be null. The selector syntax is the same as that for `createDurableSubscriber`.

Usage Notes

Oracle Streams AQ 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 - the syntax is (schema).(topic_name)[@dblink].

a) 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

b) 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

A message selector can also be specified. Only messages that satisfy the selector are delivered to the remote subscriber. The message selector can be null. The syntax for the selector is the same as that for createDurableSubscriber. The selector can be null.

Example

```
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                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("WesternRegion", "WS.shipped_orders_topic", null);
/* create a remote subscriber (selector is null )*/
subscriber1 = ((AQjmsSession)jms_sess).createRemoteSubscriber(
          shipped_orders,
          remoteAgent,
          null);
```

Creating a Remote Subscriber for Topics of Oracle Object Type Messages

Purpose

Creates a remote subscriber for topics of Oracle object type messages.

Syntax

```
public void createRemoteSubscriber(javax.jms.Topic topic,
                                   oracle.jms.AQjmsAgent remote_subscriber,
                                   java.lang.String messageSelector,
                                   java.lang.Object payload_factory)
                            throws JMSException
```

Parameters

topic

Topic to subscribe to.

remote_subscriber

AQjmsAgent that refers to the remote subscriber.

messageSelector

Only messages with properties matching the message selector expression are delivered. This value can be null. The selector syntax is the same as that for `createDurableSubscriber`.

payload_factory

CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

Oracle Streams AQ 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 - the syntax is (schema).(topic_name)[@dblink].

a) 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

b) 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

The CustomDatumFactory of the Oracle object type of the topic must be specified. A message selector can also be specified. Only messages that satisfy the selector are delivered to the remote subscriber. The message selector can be null. The syntax for message selector is the same as that for createDurableSubscriber with Topics of Oracle object type messages. The message selector can be null.

Example

```
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                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 */
shipped_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);
```

Unsubscribing a Durable Subscription

`Unsubscribe` requires exclusive access to the topics. If there are pending JMS send, publish, or receive operations on the same topic when this call is applied, then exception ORA - 4020 is raised. There are two solutions to the problem:

- Limit calls to `unsubscribe` at 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 `unsubscribe`.

This section contains these topics:

- Unsubscribing a Durable Subscription for a Local Subscriber
- Unsubscribing a Durable Subscription for a Remote Subscriber

Unsubscribing a Durable Subscription for a Local Subscriber

Purpose

Unsubscribes a durable subscription for a local subscriber.

Syntax

```
public void unsubscribe(javax.jms.Topic topic,
                        java.lang.String subs_name)
                 throws JMSException
```

Parameters

topic

Non-temporary topic to subscribe to.

subs_name

Name used to identify this subscription.

Usage Notes

Unsubscribe a durable subscription that has been created by a client on the specified topic.

Example

```
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");
/* unsusbcribe "WesternShipping" from shipped_orders */
jms_sess.unsubscribe(shipped_orders, "WesternShipping");
```

Unsubscribing a Durable Subscription for a Remote Subscriber

Purpose

Unsubscribes a durable subscription for a remote subscriber.

Syntax

```
public void unsubscribe(javax.jms.Topic topic,
                        oracle.jms.AQjmsAgent remote_subscriber)
                 throws JMSException
```

Parameters

topic

Non-temporary topic to subscribe to.

remote_subscriber

`AQjmsAgent` that refers to the remote subscriber. The address field of the `AQjmsAgent` cannot be null.

Example

```
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

This section contains these topics:

- Creating a TopicReceiver for a Topic of Standard JMS Type Messages
- Creating a TopicReceiver for a Topic of Oracle Object Type Messages

Creating a TopicReceiver for a Topic of Standard JMS Type Messages

Purpose

Creates a `TopicReceiver` for a topic of standard JMS type messages.

Syntax

```
public oracle.jms.AQjmsTopicReceiver createTopicReceiver(
              javax.jms.Topic topic,
              java.lang.String receiver_name,
              java.lang.String messageSelector)
       throws JMSException
```

Parameters

topic
Topic to access.

receiver_name
Name of message receiver.

messageSelector
Only messages with properties matching the message selector expression are delivered. This value can be null. The selector syntax is the same as that for `createDurableSubscriber`.

Usage Notes

Oracle Streams AQ 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 order to create a `TopicReceiver` object for consumers that are not durable subscribers. A message selector can be specified. The syntax for the message selector is the same as that of a `QueueReceiver` for a **queue** of standard JMS type messages.

Example

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = ull;
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

Purpose

Creates a `TopicReceiver` for a topic of Oracle object type messages with selector.

Syntax

```
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
```

Parameters

topic
Topic to access.

receiver_name
Name of message receiver.

messageSelector
Only messages with properties matching the message selector expression are delivered. This value can be null. The selector syntax is the same as that for `createDurableSubscriber`.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

Note: CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

Oracle Streams AQ 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 order to create a `TopicReceiver` object for consumers that are not durable subscribers. The CustomDatumFactory of the Oracle object type of the queue must be specified. A message selector can also be specified. This can be null. The syntax for the message selector is the same as that of a `QueueReceiver` for queues with Oracle object type messages.

Example

```
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 TopicBrowser

You can create a `TopicBrowser` for:

- Topics with Text, Stream, Objects, Bytes or Map Messages
- Topics with Text, Stream, Objects, Bytes, Map Messages, Locking Messages
- Topics of Oracle Object Type Messages
- Topics of Oracle Object Type Messages, Locking Messages

Topics with Text, Stream, Objects, Bytes or Map Messages

Purpose

Creates a `TopicBrowser` for topics with text, stream, objects, bytes, or map messages.

Syntax

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector)
                                      throws JMSException
```

Parameters

topic
Topic to access.

cons_name
Name of the durable subscriber or consumer.

messageSelector
Only messages with properties matching the message selector expression are delivered.

Usage Notes

To retrieve messages that have a certain `correlationID`, the selector for the `TopicBrowser` can be one of the following:

- `JMSMessageID = 'ID:23452345'` to retrieve messages that have a specified message ID

- JMS message header fields or properties:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- User-defined message properties:

  ```
  color IN ('RED', BLUE', 'GREEN') AND price < 30000
  ```

All message IDs must be prefixed with "ID:". Use methods in `java.util.Enumeration` to go through a list of messages.

Example 1

```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(topic, "SUBS1");
```

Example 2

```
/* 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'");
```

Topics with Text, Stream, Objects, Bytes, Map Messages, Locking Messages

Purpose

Creates a `TopicBrowser` for topics with text, stream, objects, bytes or map messages, locking messages while browsing.

Syntax

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector,
                                             boolean locked)
                                      throws JMSException
```

Parameters

topic
Topic to access.

cons_name
Name of the durable subscriber or consumer.

messageSelector
Only messages with properties matching the message selector expression are delivered.

locked
If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE).

Usage Notes

If a locked parameter is specified as true, then messages are locked as they are browsed. Hence these messages cannot be removed by other consumers until the browsing session ends the transaction.

Example 1

```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(
          topic,
         "SUBS1",
          true);
```

Example 2

```
/* 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(
          topic,
         "SUBS1",
         "JMSCorrelationID = 'RUSH'",
```

```
true);
```

Topics of Oracle Object Type Messages

Purpose

Creates a `TopicBrowser` for topics of Oracle object type messages.

Syntax

```
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
```

Parameters

topic

Topic to access.

cons_name

Name of the durable subscriber or consumer.

messageSelector

Only messages with properties matching the message selector expression are delivered.

payload_factory

CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

Usage Notes

For topics containing `AdtMessages`, the selector for `TopicBrowser` can be a SQL expression on the message payload contents or messageID or priority or `correlationID`.

- Selector on message ID - to retrieve messages that have a specific messageID

```
msgid = '23434556566767676'
```

Note: in this case message IDs must NOT be prefixed with "ID:"

- Selector on priority or correlation is specified as follows:

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Selector on message payload is specified as follows:

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

Example

The CustomDatum factory 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.

```
/* Create a browser for a Topic with Adt messages of type EMPLOYEE*/
TopicSession jms_session
TopicBrowser browser;
Topic        test_topic;
browser = ((AQjmsSession) jms_session).createBrowser(
            test_topic,
           "SUBS1",
            Employee.getFactory());
```

Topics of Oracle Object Type Messages, Locking Messages

Purpose

Creates a `TopicBrowser` for topics of Oracle object type messages, locking messages while browsing.

Syntax

```
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
```

Parameters

topic
Topic to access.

cons_name
Name of the durable subscriber or consumer.

messageSelector
Only messages with properties matching the message selector expression are delivered.

payload_factory
CustomDatumFactory or ORADataFactory for the java class that maps to the Oracle ADT.

> **Note:** CustomDatum support will be deprecated in a future release. Use ORADataFactory payload factories instead.

locked
If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE).

Example

```
/* Create a browser for a Topic with ADT 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

Purpose

Browses messages using a `TopicBrowser`.

Syntax

```
public void purgeSeen()
               throws JMSException
```

Usage Notes

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

```
/* Create a browser for topics with a specified selector */
public void browse_rush_orders(TopicSession jms_session)
TopicBrowser    browser;
Topic           topic;
ObjectMessage   obj_message
BolOrder        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()
```

15

Oracle Streams AQ JMS Operational Interface: Shared Interfaces

This chapter describes the **Java Message Service** (JMS) operational interface (shared interfaces) to Oracle Streams Advanced Queuing (AQ).

This chapter contains these topics:

- Oracle Streams AQ JMS Operational Interface: Shared Interfaces
- Specifying JMS Message Property
- 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
- Using a Message Consumer Without Waiting
- Specifying the Navigation Mode for Receiving Messages
- Receiving a Message Asynchronously
- Specifying a Message Listener at the Session
- Getting Message ID
- Getting the JMS Message Properties
- Closing and Shutting Down
- Troubleshooting

Oracle Streams AQ JMS Operational Interface: Shared Interfaces

This section discusses Oracle Streams AQ shared interfaces for JMS operations.

This section contains these topics:

- AQjmsConnection.start
- AQjmsSession.getJmsConnection
- AQjmsSession.commit
- AQjmsSession.rollback
- AQjmsSession.getDBConnection
- AQjmsConnection.getOCIConnectionPool
- AQjmsSession.createBytesMessage
- AQjmsSession.createMapMessage
- AQjmsSession.createStreamMessage
- AQjmsSession.createObjectMessage
- AQjmsSession.createTextMessage
- AQjmsSession.createMessage
- AQjmsSession.createAdtMessage
- AQjmsMessage.setJMSCorrelationID

AQjmsConnection.start

Purpose

Starts a **JMS connection** for receiving messages.

Syntax

```
public void start()
          throws JMSException
```

Usage Notes

The start method is used to start (or restart) the connection's delivery of incoming messages.

AQjmsSession.getJmsConnection

Purpose

Gets the JMS connection from a session.

Syntax

```
public oracle.jms.AQjmsConnection getJmsConnection()
                                            throws JMSException
```

AQjmsSession.commit

Purpose

Commits all operations in a session.

Syntax

```
public void commit()
           throws JMSException
```

Usage Notes

This method commits all JMS and SQL operations performed in this session.

AQjmsSession.rollback

Purpose

Rolls back all operations in a session.

Syntax

```
public void rollback()
              throws JMSException
```

Usage Notes

This method terminates all JMS and SQL operations performed in this session.

AQjmsSession.getDBConnection

Purpose

Gets the underlying JDBC connection from a **JMS session**.

Syntax

```
public java.sql.Connection getDBConnection()
                                          throws JMSException
```

Usage Notes

This method is used to obtain the underlying JDBC connection from a JMS session. The JDBC connection can be used to perform SQL operations as part of the same transaction that the JMS operations are accomplished.

Example

```
java.sql.Connection db_conn;
QueueSession     jms_sess;
db_conn = ((AQjmsSession)jms_sess).getDBConnection();
```

AQjmsConnection.getOCIConnectionPool

Purpose

Gets the underlying `OracleOCIConnectionPool` from a JMS connection.

Syntax

```
public oracle.jdbc.pool.OracleOCIConnectionPool getOCIConnectionPool()
```

Usage Notes

This method is used to obtain the underlying `OracleOCIConnectionPool` instance 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

```
oracle.jdbc.pool.OracleOCIConnectionPool cpool;
QueueConnection jms_conn;
cpool = ((AQjmsConnection)jms_conn).getOCIConnectionPool();
```

AQjmsSession.createBytesMessage

Purpose

Creates a bytes **message**.

Syntax

```
public javax.jms.BytesMessage createBytesMessage()
                                                    throws JMSException
```

Usage Notes

This method 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.

AQjmsSession.createMapMessage

Purpose

Creates a map message.

Syntax

```
public javax.jms.MapMessage createMapMessage()
                                                throws JMSException
```

Usage Notes

This method 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.

AQjmsSession.createStreamMessage

Purpose

Creates a stream message.

Syntax

```
public javax.jms.StreamMessage createStreamMessage()
                                                        throws JMSException
```

Usage Notes

This method 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.

AQjmsSession.createObjectMessage

Purpose

Creates an object message.

Syntax

```
public javax.jms.ObjectMessage createObjectMessage(java.io.Serializable object)
                                                        throws JMSException
```

Usage Notes

This method 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.

AQjmsSession.createTextMessage

Purpose

Creates a text message.

Syntax

```
public javax.jms.TextMessage createTextMessage()
                                                    throws JMSException
```

Usage Notes

This method 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.

AQjmsSession.createMessage

Purpose

Creates a **JMS message**.

Syntax

```
public javax.jms.Message createMessage()
                                throws JMSException
```

Usage Notes

Use this **ADT** to store any or all of the JMS message types: bytes messages (JMSBytes), map messages (JMSMap), stream messages (JMSStream), object messages (JMSObject), or text messages (JMSText).

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.

AQjmsSession.createAdtMessage

Purpose

Creates an ADT message.

Syntax

```
public oracle.jms.AdtMessage createAdtMessage()
```

```
                                   throws JMSException
```

Usage Notes

This method can be used only if the queue table that contains the queue/topic was created with an Oracle ADT payload_type (not one of the SYS.AQ$_JMS* types).

An ADT message 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 ADTs can be generated using the Jpublisher tool.

AQjmsMessage.setJMSCorrelationID

Purpose

Specifies message correlation ID.

Syntax

```
public void setJMSCorrelationID(java.lang.String correlationID)
                             throws JMSException
```

Specifying JMS Message Property

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 Message:

- JMSXAppID (String)
- JMSXGroupID (string)
- JMSXGroupSeq (int)
- JMS_OracleExcpQ (String) - **exception queue**
- JMS_OracleDelay (int) - message delay (seconds)

The following properties can be set on AdtMessage

- JMS_OracleExcpQ (String) - exception queue - specified as "*schema*.queue_name"
- JMS_OracleDelay (int) - message delay (seconds)

This section contains these topics:

- AQjmsMessage.setBooleanProperty
- AQjmsMessage.setStringProperty
- AQjmsMessage.setIntProperty
- AQjmsMessage.setDoubleProperty
- AQjmsMessage.setFloatProperty
- AQjmsMessage.setByteProperty
- AQjmsMessage.setLongProperty
- AQjmsMessage.setShortProperty
- AQjmsMessage.getObjectProperty

AQjmsMessage.setBooleanProperty

Purpose

Specifies message property as `Boolean`.

Syntax

```
public void setBooleanProperty(java.lang.String name,
                               boolean value)
                        throws JMSException
```

Parameters

name

Name of the Boolean property.

value

Boolean property value to set in the message.

AQjmsMessage.setStringProperty

Purpose

Specifies message property as `String`.

Syntax

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value)
                       throws JMSException
```

Parameters

name
Name of the string property.

value
String property value to set in the message.

AQjmsMessage.setIntProperty

Purpose
Specifies message property as `Int`.

Syntax

```
public void setIntProperty(java.lang.String name,
                           int value)
                    throws JMSException
```

Parameters

name
Name of the integer property.

value
Integer property value to set in the message.

AQjmsMessage.setDoubleProperty

Purpose
Specifies message property as `Double`.

Syntax

```
public void setDoubleProperty(java.lang.String name,
                              double value)
                       throws JMSException
```

Parameters

name
Name of the double property.

value
Double property value to set in the message.

AQjmsMessage.setFloatProperty

Purpose

Specifies message property as `Float`.

Syntax

```
public void setFloatProperty(java.lang.String name,
                             float value)
                      throws JMSException
```

Parameters

name
Name of the float property.

value
Float property value to set in the message.

AQjmsMessage.setByteProperty

Purpose

Specifies message property as `Byte`.

Syntax

```
public void setByteProperty(java.lang.String name,
                            byte value)
                     throws JMSException
```

Parameters

name
Name of the byte property.

value
Byte property value to set in the message.

AQjmsMessage.setLongProperty

Purpose

Specifies message property as `Long`.

Syntax

```
public void setLongProperty(java.lang.String name,
                            long value)
                     throws JMSException
```

Parameters

name
Name of the long property.

value
Long property value to set in the message.

AQjmsMessage.setShortProperty

Purpose

Specifies message property as `Short`.

Syntax

```
public void setShortProperty(java.lang.String name,
                             short value)
                      throws JMSException
```

Parameters

name
Name of the short property.

value
Short property value to set in the message.

AQjmsMessage.setObjectProperty

Purpose

Specifies message property as `Object`.

Syntax

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value)
                       throws JMSException
```

Parameters

name
Name of the Java object property.

value
Java object property value to set in the message.

Usage Notes

Only objectified primitive values are supported: `Boolean`, `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double` and `String`.

Setting Default TimeToLive for All Messages Sent by a MessageProducer

Purpose

Sets default `TimeToLive` for all messages sent by a message **producer**.

Syntax

```
public void setTimeToLive(long timeToLive)
                   throws JMSException
```

Parameters

timeToLive

Message time to live in milliseconds. Zero (the default) is unlimited.

Usage Notes

TimetoLive is specified in milliseconds. It is calculated after the message is in ready state (i.e after message delay has taken effect).

Example

```
/* 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

Purpose

Sets default `Priority` for all messages sent by a `MessageProducer`.

Syntax

```
public void setPriority(int priority)
                 throws JMSException
```

Parameters

priority
Message priority for this `MessageProducer`. The default is 4.

Usage Notes

Priority values can be any integer. A smaller number indicates higher priority. If a priority value is explicitly specified during the **send** operation, then it overrides the producer's default value set by this method.

Example 1

```
/* Set default priority value to 2 for all messages sent by the QueueSender*/
QueueSender sender;
sender.setPriority(2);
```

Example 2

```
/* Set default priority value to 2 for all messages sent by the TopicPublisher*/
TopicPublisher publisher;
publisher.setPriority(1);
```

Creating an AQjms Agent

Purpose

Creates an `AQjmsAgent`.

Syntax

```
public void createAQAgent(java.lang.String agent_name,
                          boolean enable_http,
                   throws JMSException
```

Parameters

agent_name
Name of the AQ agent.

enable_http
If set to true, then this agent is allowed to access AQ through HTTP.

Receiving a Message Synchronously

You can receive a message synchronously two ways:

- Using a Message Consumer by Specifying Timeout
- Using a Message Consumer Without Waiting

Using a Message Consumer by Specifying Timeout

Purpose

Receives a message using a message **consumer** by specifying timeout.

Syntax

```
public javax.jms.Message receive(long timeout)
                          throws JMSException
```

Parameters

timeout

Timeout value (in milliseconds).

Examples

```
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);
```

Using a Message Consumer Without Waiting

Purpose

Receives a message using a message consumer without waiting.

Syntax

```
public javax.jms.Message receiveNoWait()
                                throws JMSException
```

Specifying the Navigation Mode for Receiving Messages

Purpose

Specifies the navigation mode for receiving messages.

Syntax

```
public void setNavigationMode(int mode)
                       throws JMSException
```

Parameters

mode

New value of the navigation mode.

Example

```
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());
subscriber1.setNavigationMode(AQjmsConstants.NAVIGATION_FIRST_MESSAGE);
/* get message for the subscriber, returning immediately if there was no
message */
Message = subscriber.receive();
```

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

Purpose

Specifies a message listener at the message consumer.

Syntax

```
public void setMessageListener(javax.jms.MessageListener myListener)
                        throws JMSException
```

Parameters

myListener

Set the consumer's message listener.

Example

```
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 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

Purpose

Specifies a message listener at the session.

Syntax

```
public void setMessageListener(javax.jms.MessageListener listener)
                        throws JMSException
```

Parameters

listener

Message listener to associate with this session.

Getting Message ID

This section contains these topics:

- AQjmsMessage.getJMSCorrelationID
- AQjmsMessage.getJMSMessageID

AQjmsMessage.getJMSCorrelationID

Purpose

Gets the correlation ID of a message.

Syntax

```
public java.lang.String getJMSCorrelationID()
          throws JMSException
```

AQjmsMessage.getJMSMessageID

Purpose

Gets the message ID of a message as bytes or a string.

Syntax

```
public byte[] getJMSCorrelationIDAsBytes()
          throws JMSException
```

Getting the JMS Message Properties

This section contains these topics:

- AQjmsMessage.getBooleanProperty
- AQjmsMessage.getStringProperty
- AQjmsMessage.getIntProperty
- AQjmsMessage.getDoubleProperty
- AQjmsMessage.getFloatProperty
- AQjmsMessage.getByteProperty
- AQjmsMessage.getLongProperty
- AQjmsMessage.getShortProperty
- AQjmsMessage.getObjectProperty

AQjmsMessage.getBooleanProperty

Purpose

Gets the message property as `Boolean`.

Syntax

```
public boolean getBooleanProperty(java.lang.String name)
          throws JMSException
```

Parameters

name

Name of the Boolean property.

AQjmsMessage.getStringProperty

Purpose

Gets the message property as `String`.

Syntax

```
public java.lang.String getStringProperty(java.lang.String name)
          throws JMSException
```

Parameters

name

Name of the string property.

AQjmsMessage.getIntProperty

Purpose

Gets the message property as `Int`.

Syntax

```
public int getIntProperty(java.lang.String name)
          throws JMSException
```

Parameters

name
Name of the integer property.

AQjmsMessage.getDoubleProperty

Purpose

Gets the message property as `Double`.

Syntax

```
public double getDoubleProperty(java.lang.String name)
                         throws JMSException
```

Parameters

name
Name of the double property.

AQjmsMessage.getFloatProperty

Purpose

Gets the message property as `Float`.

Syntax

```
public float getFloatProperty(java.lang.String name)
          throws JMSException
```

Parameters

name
Name of the float property.

AQjmsMessage.getByteProperty

Purpose

Gets the message property as `Byte`.

Syntax

```
public byte getByteProperty(java.lang.String name)
          throws JMSException
```

Parameters

name
Name of the byte property.

AQjmsMessage.getLongProperty

Purpose

Gets the message property as `Long`.

Syntax

```
public long getLongProperty(java.lang.String name)
          throws JMSException
```

Parameters

name
Name of the long property.

AQjmsMessage.getShortProperty

Purpose

Gets the message property as `Short`.

Syntax

```
public short getShortProperty(java.lang.String name)
                       throws JMSException
```

Parameters

name
Name of the short property.

AQjmsMessage.getObjectProperty

Purpose

Gets the message property as `Object`.

Syntax

```
public java.lang.Object getObjectProperty(java.lang.String name)
                                   throws JMSException
```

Parameters

name

Name of the Java object property.

Examples

```
TextMessage message;
message.getObjectProperty("empid", new Integer(1000);
```

Closing and Shutting Down

This section contains these topics:

- AQjmsProducer.close
- AQjmsConsumer.close
- AQjmsConnection.stop
- AQjmsSession.close
- AQjmsConnection.close

AQjmsProducer.close

Purpose

Closes a `MessageProducer`.

Syntax

```
public void close()
            throws JMSException
```

AQjmsConsumer.close

Purpose
Closes a message consumer.

Syntax
```
public void close()
          throws JMSException
```

AQjmsConnection.stop

Purpose
Stops a JMS connection.

Syntax
```
public void stop()
         throws JMSException
```

Usage Notes
This method is used to temporarily stop a connection's delivery of incoming messages.

AQjmsSession.close

Purpose
Closes a JMS session.

Syntax
```
public void close()
          throws JMSException
```

AQjmsConnection.close

Purpose
Closes a JMS connection.

Syntax

```
public void close()
          throws JMSException
```

Usage Notes

This method closes the 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 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:

- AQjmsException.getErrorCode
- AQjmsException.getErrorNumber
- AQjmsException.getLinkString
- AQjmsException.printStackTrace
- AQjmsConnection.setExceptionListener
- AQjmsConnection.getExceptionListener

AQjmsException.getErrorCode

Purpose

Gets the error code for the JMS exception.

Syntax

```
public java.lang.String getErrorCode()
```

AQjmsException.getErrorNumber

Purpose

Gets the error number for the JMS exception.

Note: This method will be deprecated in a future release. Use `getErrorCode()` instead.

Syntax

```
public int getErrorNumber()
```

AQjmsException.getLinkString

Purpose

Gets the exception linked to the JMS exception.

Syntax

```
public java.lang.String getLinkString()
```

Usage Notes

This method is used to get the exception linked to this JMS exception. In general, this contains the SQL exception raised by the database.

AQjmsException.printStackTrace

Purpose

Prints the stack trace for the JMS exception.

Syntax

```
public void printStackTrace(java.io.PrintStream s)
```

AQjmsConnection.setExceptionListener

Purpose

Specifies an exception listener for the connection.

Syntax

```
public void setExceptionListener(javax.jms.ExceptionListener listener)
                                  throws JMSException
```

Parameters

listener
Exception listener.

Usage Notes

If a serious problem is detected for the connection, then the connection's `ExceptionListener`, if one has been registered, is informed. This is accomplished by calling the listener's `onException()` method, passing it a `JMSException` describing the problem. This allows a JMS client to be asynchronously notified of a problem. Some connections only consume messages, so they have no other way to learn the connection has failed.

Example

```
//register an exception listener
Connection jms_connection;
jms_connection.setExceptionListener(
    new ExceptionListener() {
        public void onException (JMSException jmsException) {
            System.out.println("JMS-EXCEPTION: " + jmsException.toString());
        }
    };
  );
```

AQjmsConnection.getExceptionListener

Purpose

Gets the exception listener for the connection.

Syntax

```
public javax.jms.ExceptionListener getExceptionListener()
                                                  throws JMSException
```

Example

```
//Get the exception listener
Connection jms_connection;
ExceptionListener el = jms_connection.getExceptionListener();
```

16

Oracle Streams AQ JMS Types Examples

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 Streams AQ JMS Type Examples
- JMS Bytes Message Examples
- JMS Stream Message Examples
- JMS Map Message Examples
- More Oracle Streams AQ JMS Examples

How to Run the Oracle Streams AQ JMS Type Examples

To run Example 16–2 through Example 16–7 follow these steps:

1. Run the `setup.sql` script as follows

   ```
   sqlplus /NOLOG @setup.sql
   ```

2. Login into SQL*Plus as `jmsuser/jmsuser` and run the corresponding pair of SQL scripts for each type of **message**.
3. 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 setup.sql: Setting Up the Environment for Running the JMS Types Examples

The following SQL example performs the necessary setup for the JMS types examples. This setup applies to examples 1-2 through 1-7.

```
connect sys/change_on_install as sysdba;

Rem
Rem Create the JMS user: jmsuser
Rem

DROP USER jmsuser CASCADE;
CREATE USER jmsuser IDENTIFIED BY jmsuser;
GRANT CONNECT, RESOURCE, 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/jmsuser

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$_JMS_STREAM_MESSAG
Rem SYS.AQ$_JMS_MAP_MESSAGE
```

```
Rem SYS.AQ$_JMS_MESSAGE
Rem

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
    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$_JMS_EXCEPTION data in SQLPLUS
Rem
Rem When application receives an ORA-24197 error, It means the JAVA stored
Rem procedures has thrown some exceptions that could not be catergorized. The
Rem user can use GET_EXCEPTION procedure of SYS.AQ$_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 Bytes Message Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS bytes message.

Example 16–2 enqu_bytes_message.sql: Populating and Enqueuing a Bytes_ Message

This SQL example shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS bytes message 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.

```
connect jmsuser/jmsuser

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

    -- Consturct a empty bytes message 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 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 bytes message paylaod. These functions are analogy of
JMS JAVA api's.
    -- See the document for detail.

    -- Write a byte to the bytes message payload
    message.write_byte(id, 10);

    -- Write a RAW data as byte array to the bytes message payload
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

    -- Write a portion of the RAW data as byte array to bytes message 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 bytes message payload
    message.write_char(id, 'A');

    -- Write a double to the bytes message payload
    message.write_double(id, 9999.99);

    -- Write a float to the bytes message payload
    message.write_float(id, 99.99);

    -- Write a int to the bytes message payload
    message.write_int(id, 12345);
```

```
    -- Write a long to the bytes message payload
    message.write_long(id, 1234567);

    -- Write a short to the bytes message payload
    message.write_short(id, 123);

    -- Write a String to the bytes message 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 paylaod 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   Dequeuing and Retrieving JMS Bytes Message Data

This SQL example illustrates how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS bytes message represented as `sys.aq$_jms_bytes_message` type in the database. This message might be enqueued by a Java OJMS client.

```
connect jmsuser/jmsuser

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 messges 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 bytes message paylaod. These fns. are analogy of JMS Java
-- API's. See the JMS Types chapter for detail.
    dbms_output.put_line('Payload:');

    -- read a byte from the bytes message payload
    dbms_output.put_line('read_byte:' || message.read_byte(id));

    -- read a byte array into a blob object from the bytes message 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 bytes message payload
    dbms_output.put_line('read_char:'|| message.read_char(id));

    -- read a double from the bytes message payload
    dbms_output.put_line('read_double:'|| message.read_double(id));

    -- read a float from the bytes message payload
    dbms_output.put_line('read_float:'|| message.read_float(id));

    -- read a int from the bytes message payload
    dbms_output.put_line('read_int:'|| message.read_int(id));

    -- read a long from the bytes message payload
    dbms_output.put_line('read_long:'|| message.read_long(id));

    -- read a short from the bytes message payload
```

```
    dbms_output.put_line('read_short:'|| message.read_short(id));

    -- read a String from the bytes message 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 paylaod 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;
```

JMS Stream Message Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS stream message.

Example 16–4 enqu_stream_message.sql: Populating and Enqueuing a JMS Stream Message

This SQL example shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS stream message represented as `sys.aq$_jms_stream_message` type in the database. This message later can be dequeued by a JAVA OJMS client.

```
connect jmsuser/jmsuser

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

    -- Consturct a empty stream message 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 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 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 paylaod. These functions are analogy of JMS
JAVA api's.
```

```
    -- See the document for detail.

    -- Write a byte to the stream message payload
    message.write_byte(id, 10);

    -- Write a RAW data as byte array to the stream message payload
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

    -- Write a portion of the RAW data as byte array to the stream message
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 stream message payload
    message.write_char(id, 'A');

    -- Write a double to the stream message payload
    message.write_double(id, 9999.99);

    -- Write a float to the stream message payload
    message.write_float(id, 99.99);

    -- Write a int to the stream message payload
    message.write_int(id, 12345);

    -- Write a long to the stream message payload
    message.write_long(id, 1234567);

    -- Write a short to the stream message payload
    message.write_short(id, 123);

    -- Write a String to the stream message 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 paylaod population on this message anymore
    sys.aq$_jms_stream_message.clean_all();
    --message.clean(id);

    -- Enqueue this message into AQ queue using DBMS_AQ package
```

```
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_stream_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 16–5 dequ_sream_message.sql: Dequeuing and Retrieving Data From a JMS Stream Message

This SQL example shows how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS stream message represented as `sys.aq$_jms_stream_message` type in the database. This message might be enqueued by a JAVA OJMS client.

```
connect jmsuser/jmsuser

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 aq$_jms_stream_message

    -- The prepare call send the content in the PL/SQL aq$_jms_stream_message
object to
    -- JAVA stored procedure(Jserv) in the form of byte array.
    -- Passing -1 reserve a new slot within the message store of sys.aq$_jms_
stream_message.
    -- The maximum number of sys.aq$_jms_stream_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.prepare(-1);


    -- Assume the users know the types of data in the stream message 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 stream message payload
    dbms_output.put_line('read_byte:' || message.read_byte(id));

    -- Read a byte array into a blob object from the stream message 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 stream message
payload
    dbms_output.put_line('read_bytes:');
    message.read_bytes(id, blob_data);
    display_blob(blob_data);

    -- Read a char from the stream message payload
    dbms_output.put_line('read_char:'|| message.read_char(id));

    -- Read a double from the stream message payload
    dbms_output.put_line('read_double:'|| message.read_double(id));

    -- Read a float from the stream message payload
    dbms_output.put_line('read_float:'|| message.read_float(id));

    -- Read a int from the stream message payload
    dbms_output.put_line('read_int:'|| message.read_int(id));

    -- Read a long from the stream message payload
    dbms_output.put_line('read_long:'|| message.read_long(id));

    -- Read a short from the stream message payload
    dbms_output.put_line('read_short:'|| message.read_short(id));

    -- Read a String into a clob data from the stream message 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 stream message
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;

      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 paylaod 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 Map Message Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS map message.

Example 16–6 enqu_map_message.sql: Populating and Enqueuing a JMS Map Message

This SQL example shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS map message represented as `sys.aq$_jms_map_message` type in the database. This message later can be dequeued by a JAVA OJMS client.

```
connect jmsuser/jmsuser

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

    -- Consturct 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 paylaod. These functions are analogy of JMS
JAVA api's.
    -- See the document for detail.

    -- Set a byte entry in map message payload
    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 paylaod 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   dequ_map_message.sql: Dequeuing and Retrieving Data From a JMS Map Message

This SQL example illustrates how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS map message represented as `sys.aq$_jms_map_message` type in the database. This message can be enqueued by a Java OJMS client.

```
connect jmsuser/jmsuser

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'));
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 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.prepare(-1);

-- Assume the users know the names and types in the map message payload.
-- The user can use names to get the corresponsing 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 demostration 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;
    END IF;
  ELSE
    dbms_output.put_line('item not exists:'||name_arr(i));
  END IF;

END LOOP;
```

```
    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod population on this message anymore
    message.clean(id);
    -- sys.aq$_jms_map_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;
```

More Oracle Streams AQ JMS Examples

Example 16–8   Enqueuing Through the Oracle JMS Administrative Interface

The following sample program enqueues a large text message (along with JMS user properties) in an Oracle Streams AQ queue created through the OJMS administrative interfaces to hold JMS `TEXT` messages. Both the text and bytes messages enqueued in this example can be dequeued using OJMS Java clients.

```
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 following sample program enqueues a large bytes message.

```
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,
                   payload => message,
                   msgid => msgid);

    dbms_lob.freetemporary(lob_loc => body);
END;
```

Part VI

Internet Access to Oracle Streams AQ

Part VI describes how to access the Internet using Oracle Streams Advanced Queuing (AQ).

This part contains the following chapter:

- Chapter 17, "Internet Access to Oracle Streams AQ"

17

Internet Access to Oracle Streams AQ

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 Streams AQ operations. IDAP defines XML **message** structure for the body of the SOAP request. An IDAP-structured message is transmitted over the Internet using HTTP.

This chapter contains these topics:

- Overview of Oracle Streams AQ Operations over the Internet
- Internet Data Access Presentation (IDAP)
- IDAP Documents
- SOAP and Oracle Streams AQ XML Schemas
- Deploying the Oracle Streams AQ XML Servlet
- Using HTTP to Access the Oracle Streams AQ XML Servlet
- Using HTTP and HTTPS for Oracle Streams AQ Propagation
- Customizing the Oracle Streams AQ Servlet
- Frequently Asked Questions: Using Oracle Streams AQ and the Internet

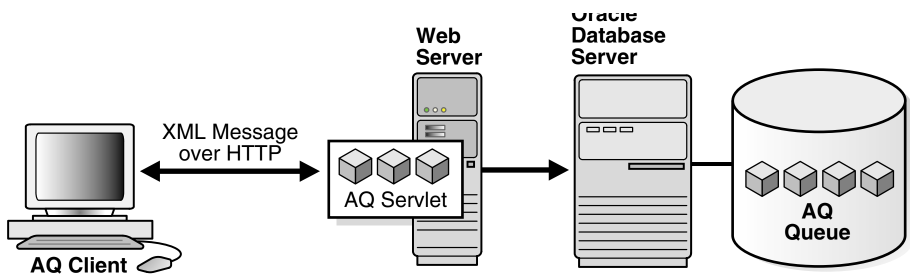
Overview of Oracle Streams AQ Operations over the Internet

Figure 17–1 shows the architecture for performing Oracle Streams AQ operations over HTTP. The major components are:

- Oracle Streams AQ client program
- Web server/Servlet Runner hosting the Oracle Streams AQ **servlet**
- Oracle Database server

The Oracle Streams AQ client program sends XML messages (conforming to IDAP) to the Oracle Streams AQ servlet. Any HTTP client, for example Web browsers, can be used. The Web server/Servlet Runner hosting the Oracle Streams AQ servlet interprets the incoming XML messages. Examples include Apache/Jserv or Tomcat. The Oracle Streams AQ servlet connects to the Oracle Database server and performs operations on the users' queues.

Figure 17–1 Architecture for Performing Oracle Streams AQ Operations Using HTTP



See Also:

- "Using HTTP to Access the Oracle Streams AQ XML Servlet" on page 17-53
- "Using HTTP and HTTPS for Oracle Streams AQ Propagation" on page 17-57

Internet Data Access Presentation (IDAP)

The 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 Streams AQ 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; for the requests with incorrect namespaces, the receiver returns an invalid request error.
- 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 (the root or top element in an XML tree))
- SOAP header (first element under the root)
- SOAP body (the Oracle Streams AQ XML document)

The SOAP Envelope

The tag of this root element 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 `SOAP:encodingStyle` 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 Headers

The tag of this first element under the root is `SOAP:Header`. A SOAP header passes necessary information, such as the transaction ID, with the request. 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.

The SOAP Body

The SOAP body, tagged SOAP:Body, 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.

For performing Oracle Streams AQ operations, the SOAP body must contain an Oracle Streams AQ XML document. The Oracle Streams AQ 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 consists of a URI followed by a "#", followed by a method name (which must not include the "#" character), as follows:

```
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.

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 content is an XML document that consists of the root and mandatory body elements. This XML document is referred to as the SOAP payload in the rest of this chapter.

The 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 Streams AQ 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:

1. 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.
2. 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.
3. 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.
4. 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.

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 Streams AQ Internet Access is supported only for 8.1or higher style queues. 8.0-compatible queues cannot be accessed using IDAP.

IDAP Client Requests for Enqueue

Client **send** and publish requests use the following methods:

- `AQXmlSend`—to enqueue to a single-consumer **queue**
- `AQXmlPublish`—to enqueue to multiconsumer queues/topics

`AQXmlSend` and `AQXmlPublish` take the arguments and argument attributes shown in Table 17–1. Required arguments are shown in bold.

Table 17–1 Client Requests for Enqueue—Arguments and Attributes for AQXmlSend and AQXmlPublish

Argument	Attribute
producer_options	**destination**—specify the queue/topic to which messages are to be sent. The destination element has an attribute `lookup_type`, which determines how the destination element value is interpreted. ■ `DATABASE (default)`—destination is interpreted as `schema.queue_name` ■ `LDAP`—the LDAP server is used to resolve the destination
-	`visibility` ■ `ON_COMMIT`—The enqueue is part of the current transaction. The operation is complete when the transaction commits. This is the default case. ■ `IMMEDIATE`—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`—the PL/SQL **transformation** to be invoked before the message is enqueued
message_set—contains one or more messages.	Each message consists of a `message_header` and `message_payload`
`message_header`	`message_id`—unique identifier of the message, supplied during dequeue
-	`correlation`—correlation identifier of the message
-	`expiration`—duration in seconds that a message is available for dequeuing. This parameter is an offset from the delay. By default messages never expire. If the message is not dequeued before it expires, then it is moved to the **exception queue** in the `EXPIRED` state
-	`delay`—duration in seconds after which a message is available for processing
-	`priority`—the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
-	**sender_id**—the application-specified identifier ■ `agent_name`, `address`, `protocol` ■ `agent_alias`—if specified, resolves to a name, address, protocol using LDAP

Table 17–1 (Cont.) Client Requests for Enqueue—Arguments and Attributes for AQXmlSend and AQXmlPublish

Argument	Attribute
-	`recipient_list`—list of recipients; overrides the default subscriber list. Each recipient consists of: ▪ `agent_name`, `address`, `protocol` ▪ `agent_alias`—if specified, resolves to a name, address, protocol using LDAP
-	`message_state`— state of the message is filled in automatically during dequeue `0` (the message is ready to be processed) `1` (the message delay has not yet been reached) `2` (the message has been processed and is retained) `3` (the message has been moved to the exception queue)
-	`exception_queue`—in case of exceptions the name of the queue to which the message is moved if it cannot be processed successfully. Messages are moved in two cases: 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. The default is the exception queue associated with the **queue table**. 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 file. If the default exception queue is used, then the parameter returns a `NULL` value at dequeue time.
message_payload	This can have different sub-elements based on the payload type of the destination queue/topic. The different payload types are described in the next section
`AQXmlCommit`	This is an empty element—if specified, the user transaction is committed at the end of the request

See Also: "Internet Integration and Internet Data Access Presentation" on page 1-16 for an explanation of IDAP message payloads

The following examples show enqueue requests using different message and queue types.

Example 17–1 IDAP Enqueue Request: Sending an ADT Message to a Single-Consumer Queue

The queue `QS.NEW_ORDER_QUE` has a payload of type `ORDER_TYP`.

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

         <message_set>
           <message_count>1</message_count>

           <message>
             <message_number>1</message_number>

             <message_header>
               <correlation>ORDER1</correlation>
               <sender_id>
                 <agent_name>scott</agent_name>
               </sender_id>
             </message_header>

             <message_payload>

              <ORDER_TYP>
                    <ORDERNO>100</ORDERNO>
                    <STATUS>NEW</STATUS>
                    <ORDERTYPE>URGENT</ORDERTYPE>
                    <ORDERREGION>EAST</ORDERREGION>
                    <CUSTOMER>
                       <CUSTNO>1001233</CUSTNO>
                       <CUSTID>MA1234555623212</CUSTID>
                       <NAME>AMERICAN EXPRESS</NAME>
                       <STREET>EXPRESS STREET</STREET>
                       <CITY>REDWOOD CITY</CITY>
                       <STATE>CA</STATE>
                       <ZIP>94065</ZIP>
                       <COUNTRY>USA</COUNTRY>
                    </CUSTOMER>
                    <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
                    <ITEMS>
                       <ITEMS_ITEM>
```

```
                            <QUANTITY>10</QUANTITY>
                            <ITEM>
                               <TITLE>Perl</TITLE>
                               <AUTHORS>Randal</AUTHORS>
                               <ISBN>ISBN20200</ISBN>
                               <PRICE>19</PRICE>
                            </ITEM>
                            <SUBTOTAL>190</SUBTOTAL>
                         </ITEMS_ITEM>
                         <ITEMS_ITEM>
                            <QUANTITY>20</QUANTITY>
                            <ITEM>
                               <TITLE>XML</TITLE>
                               <AUTHORS>Micheal</AUTHORS>
                               <ISBN>ISBN20212</ISBN>
                               <PRICE>59</PRICE>
                            </ITEM>
                            <SUBTOTAL>590</SUBTOTAL>
                         </ITEMS_ITEM>
                      </ITEMS>
                      <CCNUMBER>NUMBER01</CCNUMBER>
                      <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
                </ORDER_TYP>
              </message_payload>
             </message>
           </message_set>
         </AQXmlSend>
       </Body>
</Envelope>
```

Example 17–2 IDAP Enqueue Request: Publishing an ADT Message to a Multiconsumer Queue

The multiconsumer queue `AQUSER.EMP_TOPIC` has a payload of type `EMP_TYP`. `EMP_TYP` has the following structure:

```
 CREATE OR REPLACE TYPE emp_typ AS object (
     empno NUMBER(4),
     ename VARCHAR2(10),
     job VARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal   NUMBER(7,2),
     comm  NUMBER(7,2)
     deptno NUMBER(2));
```

A PUBLISH request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">

      <Body>
        <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

              <message_header>
                 <correlation>NEWEMP</correlation>
                 <sender_id>
                    <agent_name>scott</agent_name>
                 </sender_id>
              </message_header>

              <message_payload>
              <EMP_TYP>
                <EMPNO>1111</EMPNO>
                <ENAME>Mary</ENAME>
                <MGR>5000</MGR>
                <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                <SAL>10000</SAL>
                <COMM>100.12</COMM>
                <DEPTNO>60</DEPTNO>
              </EMP_TYP>
             </message_payload>
            </message>
          </message_set>
        </AQXmlPublish>
      </Body>
</Envelope>
```

Example 17–3 IDAP Enqueue Request: Sending a Message to a JMS Queue

The **Java Message Service** (JMS) queue `AQUSER.JMS_TEXTQ` has payload type JMS Text message (SYS.AQ$_JMS_TEXT_MESSAGE). The send request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>

        <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.JMS_TEXTQ</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

              <message_header>
                <correlation>text_msg</correlation>
                  <sender_id>
                     <agent_name>john</agent_name>
                  </sender_id>
              </message_header>

              <message_payload>

              <jms_text_message>
                <oracle_jms_properties>
                 <appid>AQProduct</appid>
                 <groupid>AQ</groupid>
                </oracle_jms_properties>

                <user_properties>
                  <property>
                    <name>Country</name>
                    <string_value>USA</string_value>
                  </property>
                  <property>
                     <name>State</name>
                    <string_value>California</string_value>
                  </property>
                 </user_properties>
```

```
                <text_data>All things bright and beautiful</text_data>
              </jms_text_message>
            </message_payload>
           </message>
         </message_set>
       </AQXmlSend>
     </Body>
</Envelope>
```

Example 17–4 IDAP Enqueue Request: Publishing a Message to a JMS Topic

The **JMS topic** `AQUSER.JMS_MAP_TOPIC` has payload type JMS Map message (`SYS.AQ$_JMS_MAP_MESSAGE`). The publish request has the following format:

```
<?xml version="1.0"?>

<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>

        <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">

          <producer_options>
            <destination>AQUSER.JMS_MAP_TOPIC</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

              <message_header>
                  <correlation>toyota</correlation>
                  <sender_id >
                     <agent_name>john</agent_name>
                  </sender_id>
                  <recipient_list>
                     <recipient>
                        <agent_name>scott</agent_name>
                     </recipient>
                     <recipient>
                        <agent_name>aquser</agent_name>
                     </recipient>
                     <recipient>
```

```
                  <agent_name>jmsuser</agent_name>
               </recipient>
            </recipient_list>
        </message_header>

        <message_payload>

          <jms_map_message>
            <oracle_jms_properties>
              <reply_to>
                 <agent_name>oracle</agent_name>
              </reply_to>
             <groupid>AQ</groupid>
           </oracle_jms_properties>

           <user_properties>
             <property>
               <name>Country</name>
               <string_value>USA</string_value>
             </property>
             <property>
               <name>State</name>
               <string_value>California</string_value>
             </property>
            </user_properties>

           <map_data>
            <item>
             <name>Car</name>
             <string_value>Toyota</string_value>
            </item>
            <item>
              <name>Color</name>
              <string_value>Blue</string_value>
            </item>
            <item>
              <name>Price</name>
              <int_value>20000</int_value>
            </item>
          </map_data>
         </jms_map_message>
       </message_payload>
      </message>
    </message_set>
  </AQXmlPublish>
```

```
      </Body>
</Envelope>
```

Example 17–5 IDAP Enqueue Request: Sending a Message to a Queue with a RAW Payload

The queue `AQUSER.RAW_MSGQ` has a payload of type RAW. The `SEND` request has the following format:

```
<?xml version="1.0"?>
   <Envelope  xmlns =  "http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
        <producer_options>
          <destination>AQUSER.RAW_MSGQ</destination>
        </producer_options>
        <message_set>
          <message_count>1</message_count>

          <message>
             <message_number>1</message_number>

                 <message_header>
                   <correlation>TKAXAS11</correlation>
                   <sender_id>
                      <agent_name>scott</agent_name>
                   </sender_id>
                 </message_header>
                 <message_payload>

<RAW>426C6F622064617461202D20626C6F622064617461202D20626C6F6220646174612042 6C6F6
22064617461202D20626C6F622064617461202D20626C6F62206461746120426</RAW>
                 </message_payload>
               </message>
            </message_set>
          </AQXmlSend>
      </Body>
</Envelope>
```

Example 17–6 IDAP Enqueue Request: Sending/Publishing and Committing the Transaction

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">

      <Body>
        <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>

              <message_header>
          <correlation>NEWEMP</correlation>
          <sender_id>
             <agent_name>scott</agent_name>
          </sender_id>
              </message_header>

              <message_payload>
                <EMP_TYP>
                  <EMPNO>1111</EMPNO>
                  <ENAME>Mary</ENAME>
                  <MGR>5000</MGR>
                  <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                  <SAL>10000</SAL>
                  <COMM>100.12</COMM>
                  <DEPTNO>60</DEPTNO>
                 </EMP_TYP>
             </message_payload>
            </message>
          </message_set>

        <AQXmlCommit/>

        </AQXmlPublish>
      </Body>
</Envelope>
```

IDAP Client Requests for Dequeue

Client requests for dequeue use the AQXmlReceive method, which takes the arguments and argument attributes shown in Table 17–2. Required arguments are shown in bold.

Table 17–2 Client Requests for Dequeue—Arguments and Attributes for AQXmlReceive

Argument	Attribute
consumer_options	**destination**—specify the queue/topic from which messages are to be received. The destination element has an attribute lookup_type, which determines how the destination element value is interpreted ■ DATABASE (default)—destination is interpreted as schema.queue_name ■ LDAP—the LDAP server is used to resolve the destination
-	consumer_name—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—the time (in seconds) to wait if there is currently no message available which matches the search criteria
-	selector—criteria used to select the message, specified as one of: ■ correlation—the correlation identifier of the message to be dequeued. ■ message_id— the message identifier of the message to be dequeued ■ condition—dequeue message that satisfy this condition. A condition is specified as a Boolean expression using syntax 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 (as specified in the where clause of a SQL query). Message properties include priority, corrid and other columns in the queue table 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 parameter cannot exceed 4000 characters.

Table 17–2 (Cont.) Client Requests for Dequeue—Arguments and Attributes for AQXmlReceive

Argument	Attribute
-	visibility ■ ON_COMMIT (default)—The dequeue is part of the current transaction. The operation is complete when the transaction commits. ■ IMMEDIATE—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—Specifies the locking action associated with the dequeue. The dequeue_mode can be specified as one of: ■ REMOVE (default): Read the message and delete it. The message can be retained in the queue table based on the retention properties. ■ BROWSE: Read the message without acquiring any lock on the message. This is equivalent to a select statement. ■ LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This is equivalent to a select for update statement.
-	navigation_mode—Specifies the position of the message that is retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. The navigation_mode can be specified as one of: ■ FIRST_MESSAGE: Retrieves the first message which is available and matches the search criteria. This resets the position to the beginning of the queue. ■ NEXT_MESSAGE (default): Retrieve the next message which is available and matches the search criteria. If the previous message belongs to a message group, then Oracle Streams AQ retrieves the next available message which matches the search criteria and belongs to the message group. This is the default. ■ NEXT_TRANSACTION: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue.
-	transformation—the PL/SQL transformation to be invoked after the message is dequeued
AQXmlCommit	This is an empty element—if specified, the user transaction is committed at the end of the request

The following examples show dequeue requests using different attributes of AQXmlReceive.

Example 17–7 IDAP Dequeue Request: Receiving Messages from a Single-Consumer Queue

Using the single-consumer queue `QS.NEW_ORDERS_QUE`, the receive request has the following format:

```
<?xml version="1.0"?>

<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

Example 17–8 IDAP Dequeue Request: Receiving Messages from a Multiconsumer Queue

Using the multiconsumer queue `AQUSER.EMP_TOPIC` with **subscriber** `APP1`, the receive request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
            <consumer_name>APP1</consumer_name>
            <wait_time>0</wait_time>
            <navigation_mode>FIRST_MESSAGE</navigation_mode>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

Example 17–9 IDAP Dequeue Request: Receiving Messages from a Specific Correlation ID

Using the single consumer queue `QS.NEW_ORDERS_QUE`, to receive messages with correlation ID `NEW`, the receive request has the following format:

```
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
```

```
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
            <selector>
                 <correlation>NEW</correlation>
            </selector>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

Example 17–10 IDAP Dequeue Request: Receiving Messages that Satisfy a Specific Condition

Using the multiconsumer queue `AQUSER.EMP_TOPIC` with subscriber `APP1` and condition `deptno=60`, the receive request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>AQUSER.EMP_TOPIC</destination>
            <consumer_name>APP1</consumer_name>
            <wait_time>0</wait_time>
            <selector>
                 <condition>tab.user_data.deptno=60</condition>
            </selector>
          </consumer_options>
        </AQXmlReceive>
      </Body>
</Envelope>
```

Example 17–11 IDAP Dequeue Request: Receiving Messages and Committing

In the dequeue request examples, if you include `AQXmlCommit` at the end of the `RECEIVE` request, then the transaction is committed upon completion of the operation. In "IDAP Dequeue Request: Receiving Messages from a Multiconsumer Queue" on page 17-19, the receive request can include the commit flag as follows:

```
 <?xml version="1.0"?>

<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
```

```
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
          </consumer_options>

          <AQXmlCommit/>

        </AQXmlReceive>
      </Body>
</Envelope>
```

Example 17–12 IDAP Dequeue Request: Browsing Messages

Messages are dequeued in `REMOVE` mode by default. To receive messages from `QS.NEW_ORDERS_QUE` in `BROWSE` mode, modify the receive request as follows:

```
<?xml version="1.0"?>

<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <wait_time>0</wait_time>
            <dequeue_mode>BROWSE</dequeue_mode>
          </consumer_options>
         </AQXmlReceive>
      </Body>
</Envelope>
```

IDAP Client Requests for Registration

Client requests for registration use the AQXmlRegister method, which takes the arguments and argument attributes shown in Table 17–3. Required arguments are shown in bold.

Table 17–3 Client Registration—Arguments and Attributes for AQXmlRegister

Argument	Attribute
register_options	**destination**—specify the queue or topic on which notifications are registered. The destination element has an attribute `lookup_type`, which determines how the destination element value is interpreted ▪ `DATABASE (default)`—destination is interpreted as `schema.queue_name` ▪ `LDAP`—the LDAP server is used to resolve the destination
-	`consumer_name`—the consumer name for multiconsumer queues or topics. For single consumer queues, this parameter must not be specified
-	**notify_url**—where notification is sent when a message is enqueued. The form can be `http://`*`url`* or `mailto://`*`email address`* or `plsql://`*`pl/sql procedure`*.

Example 17–13 IDAP Register Request: Registering for Notification at an E-mail Address

To notify an e-mail address of messages enqueued for consumer `APP1` in queue `AQUSER.EMP_TOPIC`, the register request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>

        <AQXmlRegister xmlns = "http://ns.oracle.com/AQ/schemas/access">

          <register_options>
            <destination>AQUSER.EMP_TOPIC</destination>
            <consumer_name>APP1</consumer_name>
            <notify_url>mailto:app1@hotmail.com</notify_url>
          </register_options>

          <AQXmlCommit/>

        </AQXmlRegister>
      </Body>
</Envelope>
```

IDAP Client Requests to Commit a Transaction

A request to commit all actions performed by the user in a session uses the `AQXmlCommit` method.

Example 17–14 IDAP Commit Request Example

A commit request has the following format.

```
<?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 Rollback a Transaction

A request to roll back all actions performed by the user in a session uses the `AQXmlRollback` method. Actions performed with `IMMEDIATE` visibility are not rolled back.

Example 17–15 IDAP Rollback Request Example

An IDAP client rollback request has the following format:

```
<?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 Enqueue

The response to an enqueue request to a single-consumer queue uses the `AQXmlSendResponse` method. The components of the response are shown in Table 17–4.

Table 17–4 IDAP Server Response to an Enqueue to a Single-Consumer Queue (AQXmlSendResponse)

Response	Attribute
`status_response`	`status_code`—indicates success (0) or failure (-1) `error_code`—Oracle code for the error `error_message`—description of the error
`send_result`	`destination`—where the message was sent `message_id`—identifier for every message sent

Example 17–16 Server Request: Enqueuing a Single Message to a Single-Consumer Queue

The result of a SEND request to the single consumer queue QS.NEW_ORDERS_QUE has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <AQXmlSendResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <send_result>
            <destination>QS.NEW_ORDERS_QUE</destination>
            <message_id>12341234123412341234</message_id>
         </send_result>
      </AQXmlSendResponse>
   </Body>
</Envelope>
```

The response to an enqueue request to a multiconsumer queue or topic uses the AQXmlPublishResponse method. The components of the response are shown in Table 17–5.

Table 17–5 IDAP Server Response to an Enqueue to a Multiconsumer Queue or Topic (AQXmlPublishResponse)

Response	Attribute
status_response	status_code—indicates success (0) or failure (-1) error_code—Oracle code for the error error_message—description of the error
publish_result	destination—where the message was sent message_id—identifier for every message sent

Example 17–17 IDAP Server Request: Enqueuing to a Multiconsumer Queue

The result of a SEND request to the multiconsumer queue AQUSER.EMP_TOPIC has the following format:

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

```
            <status_response>
               <status_code>0</status_code>
            </status_response>
            <publish_result>
               <destination>AQUSER.EMP_TOPIC</destination>
               <message_id>2343443543545654654654654</message_id>
            </publish_result>
         </AQXmlPublishResponse>
      </Body>
   </Envelope>
```

IDAP Server Response to a Dequeue Request

The response to a dequeue request uses the AQXmlReceiveResponse method. The components of the response are shown in Table 17–6.

Table 17–6 IDAP Server Response to a Dequeue from a Queue or Topic (AQXmlReceiveResponse)

Response	Attribute
status_response	status_code—indicates success (0) or failure (-1) error_code—Oracle code for the error error_message—description of the error
receive_result	destination—where the message was sent message_set—the set of messages dequeued

Example 17–18 IDAP Dequeue Response: Receiving Messages from an ADT Queue (AQXmlReceiveResponse)

The result of a RECEIVE request on the queue AQUSER.EMP_TOPIC with a payload of type EMP_TYP has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <AQXmlReceiveResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <receive_result>
            <destination>AQUSER.EMP_TOPIC</destination>
            <message_set>
               <message_count>1</message_count>
               <message>
```

```
                        <message_number>1</message_number>
                        <message_header>
                           <message_id>1234344545565667</message_id>
                           <correlation>TKAXAP10</correlation>
                           <priority>1</priority>
                           <delivery_count>0</delivery_count>
                           <sender_id>
                              <agent_name>scott</agent_name>
                           </sender_id>
                           <message_state>0</message_state>
                        </message_header>
                        <message_payload>
                          <EMP_TYP>
                            <EMPNO>1111</EMPNO>
                            <ENAME>Mary</ENAME>
                            <MGR>5000</MGR>
                            <HIREDATE>1996-01-01 0:0:0</HIREDATE>
                            <SAL>10000</SAL>
                            <COMM>100.12</COMM>
                            <DEPTNO>60</DEPTNO>
                          </EMP_TYP>
                       </message_payload>
                     </message>
                  </message_set>
               </receive_result>
            </AQXmlReceiveResponse>
   </Body>
</Envelope>
```

Example 17–19 IDAP Dequeue Response: Receiving Messages from a JMS Queue

The result of a `RECEIVE` request on a queue with a payload of type JMS Text message has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
      <AQXmlReceiveResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <receive_result>
            <destination>AQUSER.JMS_TEXTQ</destination>
            <message_set>
               <message_count>1</message_count>
```

```
                  <message>
                     <message_number>1</message_number>
                     <message_header>
                        <message_id>12233435454656567</message_id>
                        <correlation>TKAXAP01</correlation>
                        <delay>0</delay>
                        <priority>1</priority>
                        <message_state>0</message_state>
                        <sender_id>
                           <agent_name>scott</agent_name>
                        </sender_id>
                     </message_header>
                     <message_payload>
                        <jms_text_message>
                           <oracle_jms_properties>
                              <reply_to>
                                 <agent_name>oracle</agent_name>
                                 <address>redwoodshores</address>
                                 <protocol>100</protocol>
                              </reply_to>
                              <userid>AQUSER</userid>
                              <appid>AQProduct</appid>
                              <groupid>AQ</groupid>
                              <timestamp>01-12-2000</timestamp>
                              <recv_timestamp>12-12-2000</recv_timestamp>
                           </oracle_jms_properties>
                           <user_properties>
                              <property>
                                 <name>Country</name>
                                 <string_value>USA</string_value>
                              </property>
                              <property>
                                 <name>State</name>
                                 <string_value>California</string_value>
                              </property>
                           </user_properties>
                           <text_data>All things bright and beautiful</text_data>
                        </jms_text_message>
                     </message_payload>
                  </message>
               </message_set>
            </receive_result>
         </AQXmlReceiveResponse>
      </Body>
   </Envelope>
```

IDAP Server Response to a Register Request

The response to a register request uses the `AQXmlRegisterResponse` method, which consists of `status_response`. (See Table 17–6 for a description of `status_response`.)

IDAP Commit Response

The response to a commit request uses the AQXmlCommitResponse method, which consists of `status_response`. (See Table 17–6 for a description of `status_response`.)

Example 17–20 IDAP Commit Response

The response to a commit request has the following format:

```
<?xml version = '1.0'?>
<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 the `AQXmlRollbackResponse` method, which consists of `status_response`. (See Table 17–6 for a description of `status_response`.)

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. `AQXmlNotification` consists of:

- `notification_options`, which has
  - `destination`—the destination queue/topic on which the event occurred
  - `consumer_name`—in case of multiconsumer queues/topics, this refers to the consumer name for which the event occurred

- `message_set`—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 consists of:

- `faultcode` - error code for fault
- `faultstring` - indicates a client error or a server error. A client error means that the request is not valid. Server error indicates that the Oracle Streams AQ servlet has not been set up correctly
- `detail`, which consists of
  - `status_response`

Example 17–21 IDAP Response in Case of Error

A `FAULT` message has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <Fault xmlns="http://schemas.xmlsoap.org/soap/envelope/">
         <faultcode>100</faultcode>
         <faultstring>Server Fault</faultstring>
         <detail>
            <status_response>
               <status_code>-1</status_code>
               <error_code>410</error_code>
               <error_message>JMS-410: XML SQL Excetpion
ORA-24031: invalid value, OWNER_NAME should be non-NULL
ORA-06512: at "SYS.DBMS_AQJMS", line 177
ORA-06512: at line 1
</error_message>
            </status_response>
         </detail>
      </Fault>
   </Body>
</Envelope>
```

SOAP and Oracle Streams AQ XML Schemas

IDAP exposes the SOAP **schema** and the Oracle Streams AQ XML schema to the client. All documents sent are validated against these schemas:

- SOAP schema—`http://schemas.xmlsoap.org/soap/envelope/`
- Oracle Streams AQ XML schema—`http://ns.oracle.com/AQ/schemas/access`

SOAP Schema

The SOAP schema describes the structure of a document: envelope, header, and body.

```
<?xml version='1.0'?>
<!-- XML Schema for SOAP v 1.1 Envelope -->
<schema xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:tns='http://schemas.xmlsoap.org/soap/envelope/'
        targetNamespace='http://schemas.xmlsoap.org/soap/envelope/'>

  <!-- SOAP envelope, header and body -->

  <element name="Envelope" type="tns:Envelope"/>
  <complexType name='Envelope'>
   <sequence>
    <element ref='tns:Header' minOccurs='0'/>
    <element ref='tns:Body' minOccurs='1'/>
    <any minOccurs='0' maxOccurs='*'/>
   </sequence>
    <anyAttribute/>
  </complexType>

  <element name="Header" type="tns:Header"/>
  <complexType name='Header'>
   <sequence>
    <any minOccurs='0' maxOccurs='*'/>
   </sequence>
    <anyAttribute/>
  </complexType>

  <element name="Body" type="tns:Body"/>
  <complexType name='Body'>
   <sequence>
    <any minOccurs='0' maxOccurs='*'/>
   </sequence>
```

```
  <anyAttribute/>
</complexType>

<!-- Global Attributes.  The following attributes are intended
     to be usable through qualified attribute names on any complex type
     referencing them.  -->

<attribute name="mustUnderstand" type="tns:mutype" use="optional" value="0"/>
</attribute>

<simpleType  name="mutype">
   <restriction base="string">
<enumeration value="0"/>
      <enumeration value="1"/>
   </restriction>
</simpleType>

<attribute name='actor' type='anyURI'/>

<!-- 'encodingStyle' indicates any canonicalization conventions followed
     in the contents of the containing element.  For example, the value
     'http://schemas.xmlsoap.org/soap/encoding/' indicates
     the pattern described in SOAP specification. -->

<simpleType name='encodingStyle'>
  <list itemType='anyURI'/>
</simpleType>
<attributeGroup name='encodingStyle'>
  <attribute name='encodingStyle' type='tns:encodingStyle'/>
</attributeGroup>

<!-- SOAP fault reporting structure -->
<complexType name='Fault' final='extension'>
 <sequence>
  <element name='faultcode' type='QName'/>
  <element name='faultstring' type='string'/>
  <element name='faultactor' type='anyURI' minOccurs='0'/>
  <element name='detail' type='tns:detail' minOccurs='0'/>
 </sequence>
</complexType>

<complexType name='detail'>
 <sequence>
  <any minOccurs='0' maxOccurs='*'/>
 </sequence>
```

```
    <anyAttribute/>
  </complexType>

</schema>
```

IDAP Schema

The IDAP schema describes the contents of the IDAP body for Internet access to Oracle Streams AQ features.

```
<?xml version="1.0"?>

<!-- ****************** Oracle Streams AQ xml schema  ****************** -->

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
        targetNamespace = "http://ns.oracle.com/AQ/schemas/access"
        xmlns:aq = "http://ns.oracle.com/AQ/schemas/access"
        xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

<import namespace = "http://schemas.xmlsoap.org/soap/envelope/"
        schemaLocation = "soap_env.xsd" />



<!-- ****************** Oracle Streams AQ xml client operations
****************** -->

   <element name="AQXmlSend">
      <complexType mixed="true">
        <sequence>
           <element ref="aq:producer_options" minOccurs="1" maxOccurs="1" />
           <element ref="aq:message_set" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:AQXmlCommit" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>

   <element name="AQXmlPublish">
      <complexType mixed="true">
        <sequence>
           <element ref="aq:producer_options" minOccurs="1" maxOccurs="1" />
           <element ref="aq:message_set" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:AQXmlCommit" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
```

```
    </element>


   <element name="AQXmlReceive">
      <complexType mixed="true">
        <sequence>
           <element ref="aq:consumer_options" minOccurs="1" maxOccurs="1" />
           <element ref="aq:AQXmlCommit" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
   </element>


   <element name="AQXmlRegister">
      <complexType mixed="true">
        <sequence>
           <element ref="aq:register_options" minOccurs="1" maxOccurs="1" />
           <element ref="aq:AQXmlCommit" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
   </element>

   <element name="AQXmlCommit">
      <complexType>
      </complexType>
   </element>

   <element name="AQXmlRollback">
      <complexType>
      </complexType>
   </element>



   <!-- ****************** Oracle Streams AQ xml server responses
****************** -->

   <element name="AQXmlSendResponse">
      <complexType mixed="true">
        <sequence>
           <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:send_result" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
```

```
<element name="AQXmlPublishResponse">
   <complexType mixed="true">
     <sequence>
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:publish_result" minOccurs="0" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>


<element name="AQXmlReceiveResponse">
   <complexType mixed="true">
     <sequence>
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
        <element ref="aq:receive_result" minOccurs="0" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>


<element name="AQXmlRegisterResponse">
   <complexType mixed="true">
     <sequence>
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>

<element name="AQXmlCommitResponse">
   <complexType mixed="true">
     <sequence>
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>

<element name="AQXmlRollbackResponse">
   <complexType mixed="true">
     <sequence>
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>
```

```
<element name="destination">
  <complexType>
   <simpleContent>
     <extension base='string'>
       <attribute name="lookup_type" type="aq:dest_lookup_type"
        default="DATABASE"/>
     </extension>
  </simpleContent>
</complexType>
</element>

<!-- **** destination lookup type ******* -->
<!-- lookup_type can be specified to either lookup LDAP or use  -->
<simpleType name="dest_lookup_type">
 <restriction base="string">
   <enumeration value="DATABASE"/>
   <enumeration value="LDAP"/>
  </restriction>
 </simpleType>

<!-- ****************** Producer Options ****************** -->
<element name="producer_options">
   <complexType mixed="true">
     <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:visibility" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:transformation" minOccurs="0" maxOccurs="1"/>
     </sequence>
   </complexType>
 </element>

<!-- ****************** Consumer Options ****************** -->
<element name="consumer_options">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:wait_time" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:selector" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:batch_size" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:visibility" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:dequeue_mode" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:navigation_mode" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:transformation" minOccurs="0" maxOccurs="1"/>
```

```
          </sequence>
         </complexType>
      </element>


     <!-- ****************** Register Options ****************** -->
     <element name="register_options">
        <complexType mixed="true">
          <sequence>
          <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
          <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:notify_url" minOccurs="1" maxOccurs="1"/>
          </sequence>
        </complexType>
      </element>

     <element name="recipient_list">
       <complexType mixed="true">
        <sequence>
  <element ref="aq:recipient" minOccurs="1" maxOccurs="*"/>
        </sequence>
       </complexType>
     </element>


     <!-- ****************** Message Set ************************* -->
     <element name="message_set">
        <complexType mixed="true">
         <sequence>
          <element ref="aq:message_count" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:message" minOccurs="0" maxOccurs="*"/>
         </sequence>
        </complexType>
     </element>


     <!-- ****************** Message ************************* -->
     <element name="message">
        <complexType mixed="true">
         <sequence>
          <element ref="aq:message_number" minOccurs="0" maxOccurs="1"/>
          <element ref="aq:message_header" minOccurs="1" maxOccurs="1"/>
          <element ref="aq:message_payload" minOccurs="0" maxOccurs="1"/>
         </sequence>
        </complexType>
```

```
</element>


<!-- ****************** Message header ****************** -->
<element name="message_header">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:correlation" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:delay" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:expiration" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:priority" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:delivery_count" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:sender_id" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:recipient_list" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:message_state" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:exception_queue" minOccurs="0" maxOccurs="1"/>
   </sequence>
  </complexType>
 </element>


<!-- ****************** Oracle JMS properties ****************** -->
<element name="oracle_jms_properties">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:type" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:reply_to" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:userid" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:appid" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:groupid" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:group_sequence" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:timestamp" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:recv_timestamp" minOccurs="0" maxOccurs="1"/>
    </sequence>
   </complexType>
 </element>


<!-- ****************** Message payload ****************** -->
<element name="message_payload">
  <complexType>
     <choice>
       <element ref="aq:raw" minOccurs="0" maxOccurs="1"/>
       <element ref="aq:jms_text_message" minOccurs="0" maxOccurs="1"/>
```

```
       <element ref="aq:jms_map_message" minOccurs="0" maxOccurs="1"/>
       <element ref="aq:jms_bytes_message" minOccurs="0" maxOccurs="1"/>
       <element ref="aq:jms_object_message" minOccurs="0" maxOccurs="1"/>
  <any minOccurs="0" maxOccurs="*" processContents="skip"/>
     </choice>
  </complexType>
</element>


<!-- ****************** User-defined properties ****************** -->
<element name="user_properties">
   <complexType mixed="true">
     <sequence>
     <element ref="aq:property" minOccurs="0" maxOccurs="*"/>
     </sequence>
   </complexType>
</element>

<!-- ****************** Property  ****************** -->
<element name="property">
   <complexType mixed="true">
     <sequence>
 <element ref="aq:name" minOccurs="1" maxOccurs="1"/>
 <choice>
   <element ref="aq:int_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:string_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:long_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:double_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:boolean_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:float_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:short_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:byte_value" minOccurs="1" maxOccurs="1"/>
 </choice>
     </sequence>
   </complexType>
 </element>


<!-- ****************** Status response ****************** -->
<element name="status_response">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:acknowledge" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:status_code" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:error_code" minOccurs="0" maxOccurs="1"/>
```

```
     <element ref="aq:error_message" minOccurs="0" maxOccurs="1"/>
    </sequence>
   </complexType>
</element>


<!-- ****************** Send result ****************** -->
<element name="send_result">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
   </sequence>
   </complexType>
</element>

<!-- ****************** Publish result ****************** -->
<element name="publish_result">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
    </sequence>
   </complexType>
</element>


<!-- ****************** Receive result ****************** -->
<element name="receive_result">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
    </sequence>
   </complexType>
</element>


<!-- ****************** Notification ************************** -->
<element name="notification_options">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
    </sequence>
```

```
   </complexType>
</element>


<element name="priority" type="integer"/>
<element name="expiration" type="integer"/>
<element name="consumer_name" type="string"/>
<element name="wait_time" type="integer"/>
<element name="batch_size" type="integer"/>

<element name="notify_url" type="string"/>
<element name="message_id" type="string"/>
<element name="message_state" type="string"/>


<element name="message_number" type="integer"/>
<element name="message_count" type="integer"/>

<element name="correlation" type="string"/>
<element name="delay" type="integer"/>
<element name="delivery_count" type="integer"/>
<element name="exception_queue" type="string"/>
<element name="agent_alias" type="string"/>

<element name="type" type="string"/>
<element name="userid" type="string"/>
<element name="appid" type="string"/>
<element name="groupid" type="string"/>
<element name="group_sequence" type="integer"/>
<element name="timestamp" type="date"/>
<element name="recv_timestamp" type="date"/>

<element name="recipient">
  <complexType>
    <choice>
      <sequence>
        <element ref="aq:agent_name" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:address" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:protocol" minOccurs="0" maxOccurs="1"/>
      </sequence>
      <element ref="aq:agent_alias" minOccurs="1" maxOccurs="1"/>
    </choice>
  </complexType>
</element>
```

```
<element name="sender_id">
  <complexType>
    <choice>
      <sequence>
        <element ref="aq:agent_name" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:address" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:protocol" minOccurs="0" maxOccurs="1"/>
      </sequence>
      <element ref="aq:agent_alias" minOccurs="1" maxOccurs="1"/>
    </choice>
  </complexType>
</element>


<element name="reply_to">
  <complexType>
    <choice>
      <sequence>
        <element ref="aq:agent_name" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:address" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:protocol" minOccurs="0" maxOccurs="1"/>
      </sequence>
      <element ref="aq:agent_alias" minOccurs="1" maxOccurs="1"/>
    </choice>
  </complexType>
</element>


<element name="selector">
  <complexType>
<choice>
         <element ref="aq:correlation" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
         <element ref="aq:condition" minOccurs="0" maxOccurs="1"/>
     </choice>
  </complexType>
</element>


<element name="condition" type="string"/>

<element name="visibility">
 <simpleType>
  <restriction base="string">
```

```
      <enumeration value="ON_COMMIT"/>
      <enumeration value="IMMEDIATE"/>
     </restriction>
    </simpleType>
    </element>

   <simpleType name="del_mode_type">
     <restriction base="string">
      <enumeration value="PERSISTENT"/>
      <enumeration value="NONPERSISTENT"/>
     </restriction>
    </simpleType>

   <element name="dequeue_mode">
   <simpleType>
    <restriction base="string">
      <enumeration value="BROWSE"/>
      <enumeration value="LOCKED"/>
      <enumeration value="REMOVE"/>
      <enumeration value="REMOVE_NODATA"/>
    </restriction>
    </simpleType>
   </element>

   <element name="navigation_mode">
   <simpleType>
    <restriction base="string">
      <enumeration value="FIRST_MESSAGE"/>
      <enumeration value="NEXT_MESSAGE"/>
      <enumeration value="NEXT_TRANSACTION"/>
    </restriction>
    </simpleType>
   </element>

   <element name="transformation" type="string"/>

   <element name="acknowledge">
     <complexType>
     </complexType>
   </element>
   <element name="status_code" type="string"/>
   <element name="error_code" type="string"/>
   <element name="error_message" type="string"/>

   <element name="name" type="string"/>
```

```
<element name="int_value" type="integer"/>
<element name="string_value" type="string"/>
<element name="long_value" type="long"/>
<element name="double_value" type="double"/>
<element name="boolean_value" type="boolean"/>
<element name="float_value" type="float"/>
<element name="short_value" type="short"/>
<element name="byte_value" type="byte"/>

<element name="agent_name" type="string"/>
<element name="address" type="string"/>
<element name="protocol" type="integer"/>


<!-- ****************** RAW message *********************** -->
<element name="raw" type="string"/>


<!-- ****************** JMS text message ****************** -->
<element name="jms_text_message">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:text_data" minOccurs="1" maxOccurs="1"/>
   </sequence>
   </complexType>
</element>

<element name="text_data" type="string"/>


<!-- ****************** JMS map message ****************** -->
<element name="jms_map_message">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:map_data" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
</element>

<!-- ****************** Map data ****************** -->
<element name="map_data">
```

```
   <complexType mixed="true">
     <sequence>
     <element ref="aq:item" minOccurs="0" maxOccurs="*"/>
     </sequence>
   </complexType>
</element>

<!-- ****************** Map Item  ****************** -->
<element name="item">
   <complexType mixed="true">
     <sequence>
 <element ref="aq:name" minOccurs="1" maxOccurs="1"/>
 <choice>
   <element ref="aq:int_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:string_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:long_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:double_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:boolean_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:float_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:short_value" minOccurs="1" maxOccurs="1"/>
   <element ref="aq:byte_value" minOccurs="1" maxOccurs="1"/>
 </choice>
     </sequence>
   </complexType>
 </element>


<!-- ****************** JMS bytes message ****************** -->
<element name="jms_bytes_message">
   <complexType mixed="true">
    <sequence>
     <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:bytes_data" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
</element>

<element name="bytes_data" type="string"/>


<!-- ****************** JMS object message ****************** -->
<element name="jms_object_message">
   <complexType mixed="true">
     <sequence>
```

```
        <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:ser_object_data" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
   </element>

   <element name="ser_object_data" type="string"/>

</schema>
```

Deploying the Oracle Streams AQ XML Servlet

The Oracle Streams AQ XML servlet is a Java class that extends the `oracle.AQ.xml.AQxmlServlet` class. The `AQxmlServlet` class extends the `javax.servlet.http.HttpServlet` class.

> **Note:** Demos for the Oracle Streams AQ XML servlet can be found in `$ORACLE_HOME/rdbms/demo/`. Check the `aqxmlREADME.txt` file for details.

The Oracle Streams AQ XML Servlet accepts requests with Content-Type `"text/xml"` or `application/x-www-form-urlencoded`. When the Content-Type request is set to `application/x-www-form-urlencoded`, you must set the parameter name to `aqxmldoc` and the value must be the URL-encoded Oracle Streams AQ XML document.

Creating the Oracle Streams AQ XML Servlet Class

The Oracle Streams AQ servlet creates a JDBC **Oracle Call Interface** (OCI) connection pool to connect to the Oracle Database server. The `init()` method of the servlet must specify an `AQxmlDataSource` object that encapsulates the database connection parameters and the username and password. See the *Oracle XML API Reference* for information on the `AQxmlDataSource` class.

The user specified in the AQxmlDataSource is the Oracle Streams AQ servlet super-user. This user must have `CREATE SESSION` privilege and `EXECUTE` privilege on the `DBMS_AQIN` package.

Example 17–22 Creating Oracle Streams AQ XML Servlet Class

Create a user `AQADM` as the Oracle Streams AQ servlet super-user as follows:

```
connect sys/change_on_install as sysdba;
GRANT CONNECT, RESOURCE to aqadm IDENTIFIED BY aqadm;
grant create session to aqadm;
GRANT EXECUTE ON DBMS_AQJMS TO aqadm;
```

A sample servlet can be created using this super-user as follows:

```
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.AQ.xml.*;

/**
 * This is a sample Oracle Streams AQ Servlet.
 */
public class AQTestServlet extends oracle.AQ.xml.AQxmlServlet
{

  /* The init method must be overloaded to specify the AQxmlDataSource */
  public void init()
  {
      AQxmlDataSource  db_drv = null;

      try
      {
        /* Create data source with username, password, sid, host, port */
        db_drv = new AQxmlDataSource("AQADM", "AQADM", "test_db", "sun-248",
"5521");

        this.setAQDataSource(db_drv);
      }
      catch (Exception ex)
      {
          System.out.println("Exception in init: " + ex);
      }
}
```

The superclass `oracle.AQ.xml.AQxmlServlet` implements the `doPost()` and `doGet()` methods in `javax.servlet.http.HttpServlet.` The `doPost()` method handles incoming SOAP requests and performs the requested Oracle Streams AQ operations.

Note: The example assumes that the Oracle Streams AQ servlet is installed in a Web server that implements Javasoft's Servlet2.2 specification (such as Tomcat 3.1). For a Web server that implements the Servlet 2.0 specification (such as Apache Jserv), you should extend the `oracle.AQ.xml.AQxmlServlet20` class instead of the `AQxmlServlet` class and override the appropriate `write()` method.

Compiling the Oracle Streams AQ XML Servlet

The Oracle Streams AQ servlet can be deployed with any Web server or servlet-runner that implements Javasoft's Servlet2.0 or Servlet2.2 interfaces (for example, Apache Jserv or Tomcat).

Because the servlet uses JDBC OCI drivers to connect to the Oracle Database server, the Oracle Database client libraries must be installed on the computer hosting the servlet.

The `LD_LIBRARY_PATH` must contain `$ORACLE_HOME/lib`.

The servlet can be compiled using JDK 1.2.x, JDK 1.3.x or JDK 1.4.x libraries.

For JDK 1.4.x, the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/ojdbc14.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$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
```

For JDK 1.3.x, the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$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
```

For JDK 1.2.x, the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jdbc/lib/orai18n.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jlib/jta.jar
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/lib/servlet.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
```

After setting the `CLASSPATH`, compile the servlet using `javac` or any other Java compiler.

> **Note:** If you are using the Oracle Streams AQ XML Servlet or the Oracle Streams AQ JMS **API** with JDK1.2, versions 1.2.2_05a or higher, then you must turn off the JIT compiler. Set `JAVA_COMPILER = none` to avoid problems in multithreaded applications.

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 Streams AQ servlet. The Oracle Streams AQ 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.

Using HTTP

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

Example 17–23 Restricting Access To Servlets in Apache

In Apache, the following can be used to restrict access (using basic authentication) to servlets installed under `aqserv/servlet`. In this example, all users sending `POST` requests to the servlet are authenticated using the `users` file in `/apache/htdocs/userdb`.

```
<Location /aqserv/servlet>
  <Limit POST>
    AuthName "AQ restricted stuff"
    AuthType Basic
    AuthUserFile /apache/htdocs/userdb/users
    require valid-user
  </Limit>
</Location>
```

Authorizing Users to Perform Operations with Oracle Streams AQ Servlet

After authenticating the users who connect to the Oracle Streams AQ servlet, you establish which operations the users are authorized to perform by doing the following:

1. Register the Oracle Streams AQ agent for Internet access.
2. Map the Oracle Streams AQ agent to one or more database users.

Registering the Oracle Streams AQ Agent

To register the Oracle Streams AQ agent for Internet access, use `DBMS_AQADM.CREATE_AQ_AGENT`. The `CREATE_AQ_AGENT` procedure takes an `agent_name`.

Example 17–24 Creating and Registering an Oracle Streams AQ Agent

Create an Oracle Streams AQ agent `JOHN` to access the Oracle Streams AQ servlet using HTTP.

```
DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'JOHN', enable_http => true);
```

The procedures ALTER_AQ_AGENT and DROP_AQ_AGENT for altering and dropping Oracle Streams AQ agents function similarly to CREATE_AQ_AGENT. These procedures are documented in the *PL/SQL Packages and Types Reference.*

Mapping the Oracle Streams AQ Agent to Database Users

To map an Oracle Streams AQ agent to one or more database users, use DBMS_AQADM.ENABLE_DB_ACCESS. With the ENABLE_DB_ACCESS procedure, you give an Oracle Streams AQ agent the privileges of a particular database user. This allows the agent to access all queues that are visible to the database users to which the agent is mapped.

Example 17–25 Mapping Oracle Streams AQ Agent to Database Users

Map the Oracle Streams AQ Internet agent JOHN to database users OE (Overseas Shipping) and CBADM (Customer Billing administrator).

```
DBMS_AQADM.ENABLE_DB_ACCESS(agent_name =>'JOHN', db_username => 'OE');
DBMS_AQADM.ENABLE_DB_ACCESS(agent_name =>'JOHN', db_username => 'CBADM');
```

Database Sessions

When the user sends a POST request to the servlet, the servlet parses the request to determine which queue/topic the user is trying to access. Accordingly, the Oracle Streams AQ servlet creates a database session as one of the database users (db_user) that maps to the Oracle Streams AQ agent. The db_user selected has privileges to access the queue specified in the request. For example:

Oracle Streams AQ agent JOHN sends an enqueue request to OE.OE_NEW_ORDERS_QUE. The servlet sees that JOHN can map to db_users OE and CBADM. Because OE.OE_NEW_ORDERS_QUE is in the OE schema, it does a CREATE SESSION as OE to perform the requested operation.

The Oracle Streams AQ servlet creates a connection pool to the Oracle Database server using the Oracle Streams AQ servlet super-user. This super-user creates sessions on behalf of db_users that the Oracle Streams AQ Internet agent maps to. Hence the super-user must have privileges to create proxy sessions for all the users specified in the ENABLE_DB_ACCESS call.

> **See Also:** "Creating the Oracle Streams AQ XML Servlet Class" on page 17-45

Example 17–26 Granting Connect to Oracle Streams AQ Servlet Super-User

The Oracle Streams AQ servlet super-user can be granted `CONNECT THROUGH` session privileges as follows:

```
connect sys/change_on_install as sysdba
rem grant super-user AQADM privileges to create proxy sessions as OE
alter user OE grant CONNECT THROUGH AQADM;

rem grant super-user AQADM privileges to create proxy sessions as CBADM
alter user CBADM grant CONNECT THROUGH AQADM;
```

If an Oracle Streams AQ Internet agent is mapped to more than one `db_user`, then all the `db_users` must have the `FORCE ANY TRANSACTION` privilege:

```
grant FORCE ANY TRANSACTION to OE;
grant FORCE ANY TRANSACTION to CBADM;
```

To disable the mapping between an agent and a database user, use `DBMS_AQADM.DISABLE_DB_ACCESS`.

The `SYSTEM.AQ$INTERNET_USERS` view lists Oracle Streams AQ agents, the protocols they are enabled for, and the mapping between Oracle Streams AQ agents and database users. Example entries in this view are shown in Table 17–7.

Table 17–7 SYSTEM_AQ$INTERNET_USERS View

agent_name	db_username	http_enabled
scott	cbadmin	YES
scott	buyer	YES
aqadmin	OE	YES
aqadmin	seller	YES
bookstore	-	NO

Using an LDAP Server with an Oracle Streams AQ XML Servlet

An LDAP server is required if:

- The `lookup_type` destination attribute is specified as LDAP. In this case the destination name is resolved to a `schema.queue_name` using the LDAP server.
- You use `agent_alias` instead of `(agent_name, address, protocol)`. If an `agent_alias` is specified in a client request, then it is resolved to `agent_name, address, protocol` using the LDAP server.

Example 17–27 Specifying the LDAP Server Context for Oracle Streams AQ XML Servlet

The LDAP context must be specified by the `setLDAPContext(DirContext)` call, as follows:

```
public void init()
{
      Hashtable env = new Hashtable(5, 0.75f);
      AQxmlDataSource  db_drv = null;

      try
      {
           /* Create data source with username, password, sid, host, port */
           db_drv = new AQxmlDataSource("AQADM", "AQADM", "test_db",
                                         "sun-248", "5521");
           this.setAQDataSource(db_drv);

           env.put(Context.INITIAL_CONTEXT_FACTORY,
                   "com.sun.jndi.ldap.LdapCtxFactory");
           env.put(Context.PROVIDER_URL, "ldap://yow:389");
           env.put(SEARCHBASE, "cn=server1,cn=dbservers,cn=wei");
           env.put(Context.SECURITY_AUTHENTICATION, "simple");
           env.put(Context.SECURITY_PRINCIPAL, "cn=orcladmin");
           env.put(Context.SECURITY_CREDENTIALS, "welcome");

           DirContext inictx = new InitialDirContext(env);
           String searchbase = (String)env.get("server_dn");
           lctx = (DirContext)inictx.lookup(searchbase);

           // Set up LDAP context
           setLdapContext(lctx);

           // Set the EMAIL server address (if any)
```

```
            setEmailServerAddr("144.25.186.236");
         }
         catch (Exception ex)
         {
            System.err.println("Servlet init exception: " +ex) ;
         }
     }
```

Using HTTP to Access the Oracle Streams AQ XML Servlet

The procedures for an Oracle Streams AQ client to make a request to the Oracle Streams AQ servlet using HTTP and for the Oracle Streams AQ servlet to process the request are as follows:

Oracle Streams AQ Client Request to the Oracle Streams AQ Servlet Using HTTP

1. The client opens an HTTP(S) connection to the server.

   For example,
   `https://aq.us.oracle.com:8000/aqserv/servlet/AQTestServlet`

   This opens a connection to port 8000 on `aq.us.oracle.com`.

2. The client logs in to the server by either:

   - HTTP basic authentication (with or without SSL)
   - SSL certificate-based client authentication

3. The client constructs the XML message representing the Send, Publish, Receive or Register request.

   Example:

```
<?xml version="1.0"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <Body>

        <AQXmlSend  xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>OE.OE_NEW_ORDERS_QUE</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>
            <message>
```

```
                    <message_number>1</message_number>
                    <message_header>
                       <correlation>XML_ADT_SINGLE_ENQ</correlation>
                       <sender_id>
                           <agent_name>john</agent_name>
                       </sender_id>
                    </message_header>
                    <message_payload>
                    <ORDER_TYP>
                           <ORDERNO>100</ORDERNO>
                           <STATUS>NEW</STATUS>
                           <ORDERTYPE>NORMAL</ORDERTYPE>
                           <ORDERREGION>EAST</ORDERREGION>
                           <CUSTOMER>
                              <CUSTNO>1001233</CUSTNO>
                              <CUSTID>JOHN</CUSTID>
                              <NAME>AMERICAN EXPRESS</NAME>
                              <STREET>EXPRESS STREET</STREET>
                              <CITY>REDWOOD CITY</CITY>
                              <STATE>CA</STATE>
                              <ZIP>94065</ZIP>
                              <COUNTRY>USA</COUNTRY>
                           </CUSTOMER>
                           <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
                           <ITEMS>
                              <ITEMS_ITEM>
                                 <QUANTITY>10</QUANTITY>
                                 <ITEM>
                                    <TITLE>Perl</TITLE>
                                    <AUTHORS>Randal</AUTHORS>
                                    <ISBN>ISBN20200</ISBN>
                                    <PRICE>19</PRICE>
                                 </ITEM>
                                 <SUBTOTAL>190</SUBTOTAL>
                              </ITEMS_ITEM>
                            </ITEMS>
                           <CCNUMBER>NUMBER01</CCNUMBER>
                           <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
                     </ORDER_TYP>
                   </message_payload>
                  </message>
                </message_set>
                </AQXmlSend>
            </Body>
      </Envelope>
```

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

   See the $ORACLE_HOME/demo directory for sample code of POST requests using HTTP.

Oracle Streams AQ Servlet Processes a Request Using HTTP

1. The server accepts the client HTTP(S) connection.
2. The server authenticates the user (Oracle Streams AQ agent) specified by the client.
3. The server receives the POST request.
4. The Oracle Streams AQ servlet is invoked.

   If this is the first request being serviced by this servlet, then the servlet is initialized—its init() method is invoked. The init () method creates a connection pool to the Oracle Database server using the AQxmlDataSource parameters (SID, host, port, Oracle Streams AQ servlet super-username, password) provided by the client.

5. The servlet processes the message as follows:

   - 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. This is described in detail in the next section.
   - The servlet determines which object (queue and topic) the agent is trying to perform operations on:

     For example, in the client request (step 3 in "Oracle Streams AQ Client Request to the Oracle Streams AQ Servlet Using HTTP"), the agent JOHN is trying to access OE.OE_NEW_ORDERS_QUE.

   - The servlet looks through the list of database users that map to this Oracle Streams AQ agent (using the AQ$INTERNET_USERS view). If any one of these db_users has privileges to access the queue/topic specified in the request, then the Oracle Streams AQ servlet super-user creates a session on behalf of this db_user.
   - For example, where the agent JOHN is mapped to the database user OE using the DBMS_AQADM.ENABLE_DB_ACCESS call, the servlet creates a session for the agent JOHN with the privileges of database user OE.

See Also: "Mapping the Oracle Streams AQ Agent to Database Users" on page 17-50

- A new database transaction is started if no transaction is active in the HTTP session. Subsequent requests in the session are part of the same transaction until an explicit `COMMIT` or ROLLBACK request is made.
- The requested operation (`SEND/PUBLISH/RECEIVE/REGISTER/COMMIT/ROLLBACK`) is performed.
- The response is formatted as an XML message and sent back the client.

  For example, the response for the preceding request can be as follows:

```
 <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <AQXmlSendResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
         <send_result>
            <destination>OE.OE_NEW_ORDERS_QUE</destination>
            <message_id>12341234123412341234123412341234</message_id>
         </send_result>
      </AQXmlSendResponse>
   </Body>
</Envelope>
```

- The response also includes the session ID in the HTTP headers as a cookie. For example, Tomcat sends back session IDs as JSESSIONID=239454ds2343. If the operation does not commit the transaction, then the transaction remains active until an explicit commit/rollback call is received. 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.

User Sessions and Transactions

After a client is authenticated and connects to the Oracle Streams AQ 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 you are using a Java or C client to post requests, then this must be accomplished programmatically. An example of a Java program used to post requests as part of the same session is given in `$ORACLE_HOME/demo` directory.

An explicit commit or rollback must be applied to end the transaction. The commit or rollback requests can also be included as part of other Oracle Streams AQ operations (`Send, Publish, Receive, Register`).

Each HTTP session has a default timeout of 120 seconds. If the user does not commit or rollback the transaction in 120 seconds after the last request that session, then the transaction is automatically terminated. This timeout can be modified in the init() method of the servlet by using `setSessionMaxInactiveTime()`.

See Also: "Customizing the Oracle Streams AQ Servlet" on page 17-60

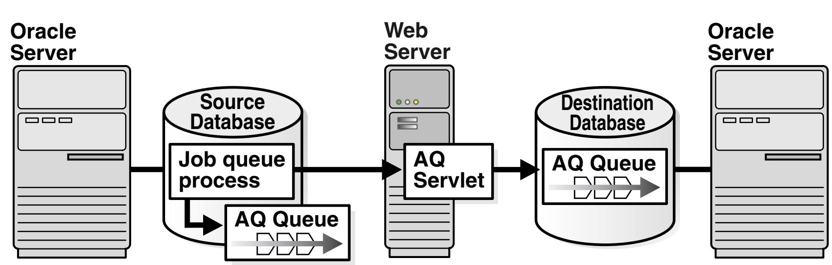
Using HTTP and HTTPS for Oracle Streams AQ Propagation

Using Oracle Streams AQ **propagation** in Oracle Database, 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.

High-Level Architecture

HTTP Oracle Streams AQ propagation uses the infrastructure for Internet access to Oracle Streams AQ as its basis. The background process doing propagation pushes messages to an Oracle Streams AQ Servlet that enqueues them into the destination database, as shown in Figure 17–2.

Figure 17–2 HTTP Oracle Streams AQ Propagation



Because HTTP propagation is different from Net Services in only the transport, most of the setup is the same as for Net Services propagation. The additional steps and differences are outlined in the following section.

Setting Up for HTTP Propagation

1. The database link at the source database must be created differently. The connect string should specify the protocol as HTTP and specify the host and port of the Web server running the Oracle Streams AQ servlet. The username and password of the database link are used for authentication with the Web server/servlet runner.
2. An Oracle Streams AQ servlet that connects to the destination database should be deployed.
3. The source database must be enabled for running Java and XML.

The rest of the steps for propagation remain the same. The administrator must use DBMS_AQADM.SCHEDULE_PROPAGATION to start propagation. Propagation can be disabled with the DBMS_AQADM.DISABLE_PROPAGATION _SCHEDULE and re-enabled using DBMS_AQADM.ENABLE_PROPAGATION _SCHEDULE. The background processes, the job queue processes propagate the messages to the destination database.The job_queue_processes parameters must be at least 2 for propagation to take place.

Any application can be easily set up to use Oracle Streams AQ HTTP propagation without any change to the existing code, by following steps 1-3. Similarly an application using Oracle Streams AQ 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.

Setting Up for Oracle Streams AQ Propagation over HTTP

1. The source database must be created for running Java and XML.
2. Create the database link with protocol as HTTP and the host and port of the Web server running the Oracle Streams AQ servlet, with the username and password for authentication with the Web Server/Servlet Runner.

   For example, if the Web Server is running on the computer `webdest.oracle.com` and listening for requests on port 8081, then the connect string of the database is as follows:

   ```
   (DESCRIPTION=(ADDRESS=(PROTOCOL=http)(HOST=webdest.oracle.com)(PORT=8081))
   ```

   If SSL is used, then specify HTTPS as the protocol in the connect string.

The database link is created as follows:

```
create public database link dba connect to john IDENTIFIED BY welcome using
'(DESCRIPTION=(ADDRESS=(PROTOCOL=http)(HOST=webdest.oracle.com)(PORT=8081)
)';
```

Where user `john` with password `welcome` is used to authenticate with the Web server and is also known by the term Oracle Streams AQ HTTP agent.

> **Note:** You cannot use `net_service_name` in `tnsnames.ora` with the database link. Doing so results in error ORA-12538.

3. You can optionally set a proxy to use for all HTTP requests from the database. Use the `UTL_HTTP.SET_PROXY` procedure, as described in *PL/SQL Packages and Types Reference.*
4. If HTTP over SSL is used, then a database wallet must be created for the source database. The wallet must be open for the duration of propagation. If HTTPS is used for propagation, then communication between the source database and the Oracle Streams AQ servlet is encrypted and the HTTPS server is authenticated with the source database. The database uses the database link username-password to authenticate itself with the HTTPS server.
5. Deploy the Oracle Streams AQ Servlet.

   Create a class AQPropServlet that extends AQxmlServlet as described in [create the Oracle Streams AQ XML Servlet Class]. This servlet must connect to the destination database. The servlet must be deployed on the Web server in the path `aqserv/servlet.`

   In Oracle Database, the propagation servlet name and deployment path are fixed; that is, they must be `AQPropServlet` and the servlet, respectively.
6. Make sure that the Oracle Streams AQ HTTP agent (John) is authorized to perform Oracle Streams AQ operations. This is accomplished at the destination database:

   a. Register the Oracle Streams AQ agent as follows:

   ```
   DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'John', enable_http => true);
   ```

   b. Map the Oracle Streams AQ agent to a database user as follows:

   ```
   DBMS_AQADM.ENABLE_DB_ACCESS(agent_name =>'John', db_username =>'CBADM')'
   ```
7. Start propagation at the source site by calling:

```
dbms_aqdm.schedule_propagation.
DBMS_AQADM.SCHEDULE_PROPAGATION('src_queue', 'dba');
```

Customizing the Oracle Streams AQ Servlet

The oracle.AQ.xml.AQxmlServlet provides the API to set the connection pool size, session timeout, style sheet, and callbacks before and after Oracle Streams AQ operations.

Setting the Connection Pool Size

The Oracle Streams AQ data source is used the specify the back-end database to which the servlet connects to perform Oracle Streams AQ operations. It contains the database SID, host name, listener port and the username/password of the Oracle Streams AQ servlet super-user.

The data source is represented by the `AQxmlDataSource` class, which can be set using the `setAQDataSource` method in the servlet. See the *Oracle XML API Reference* for more information.

The Oracle Streams AQ data source creates a pool of connections to the database server. By default the maximum size of the pool is set to 50 and the minimum is set to 1. The number of connections in the pool grows and shrinks dynamically based on the number of incoming requests. If you want to change the maximum limit on the number of connections, then you must specify a cache size using the `AQxmlDataSource.setCacheSize(size)` method.

Setting the Session Timeout

After a client is authenticated and connects to the Oracle Streams AQ 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.

Each HTTP session has a default timeout of 120 seconds. If the user does not commit or rollback the transaction in 120 seconds after the last request for that session, then the transaction is automatically terminated. This timeout can be specified in the `init()` method of the servlet by using `setSessionMaxInactiveTime()` method.

Example 17–28 Initializing the Oracle Streams AQ Servlet for HTTP: Setting the HTTP Session Timeout

The servlet is initialized as follows:

```
public class AQTestServlet extends oracle.AQ.xml.AQxmlServlet
{
  /* The init method must be overloaded to specify the AQxmlDataSource */
  public void init()
  {
      AQxmlDataSource  db_drv = null;

      try
      {
        /* Create data source with username, password, sid, host, port */
        db_drv = new AQxmlDataSource("AQADM", "AQADM",
                             "test_db", "sun-248", "5521");

        /* Set the minimum cache size to 10 connections */
        db_drv.getCacheSize(10);

        this.setAQDataSource(db_drv);

        /* Set the transaction timeout to 180 seconds */
        this.setSessionMaxInactiveTime(180);
      }
      catch (Exception ex)
      {
          System.out.println("Exception in init: " + ex);
      }
}
```

Specifying the Style Sheet for All Responses from the Servlet

Oracle Streams AQ servlet sends back XML responses. The servlet administrator can specify a style sheet that is to be set for all responses sent back from this servlet. This can be accomplished by invoking the `setStyleSheet(type,href)` or the `setStyleSheetProcessingInstr(proc_instr)` in the `init()` method of the servlet.

Example 17–29 Specifying a Stylesheet for Use in AQ XML Servlet Responses

For example, to include the following style sheet instruction for all responses, do the following:

```
<?xml-stylesheet type="text/xsl"
href="http://sun-248/stylesheets/bookOrder.xsl"?>
```

The servlet is initialized as follows:

```
public class AQTestServlet extends oracle.AQ.xml.AQxmlServlet
{
  /* The init method must be overloaded to specify the AQxmlDataSource */
  public void init()
  {
      AQxmlDataSource  db_drv = null;

      try
      {
        /* Create data source with username, password, sid, host, port */
        db_drv = new AQxmlDataSource("AQADM", "AQADM",
                                     "test_db", "sun-248", "5521");

        this.setAQDataSource(db_drv);

        /* Set the bookOrder.xsl style sheet for all responses */
        setStyleSheet("text/xsl",
"http://sun-248:8000/stylesheets/bookOrder.xsl");
      }
      catch (Exception ex)
      {
          System.out.println("Exception in init: " + ex);
      }
}
```

Callbacks Before and After Oracle Streams AQ Operations

Using the Oracle Streams AQ servlet, you can register callbacks that are invoked before and after Oracle Streams AQ operations are performed. This allows users to perform Oracle Streams AQ operations and other operations in the same transaction.

To receive callbacks, users register an object that implements the `oracle.AQ.xml.AQxmlCallback` interface.

Example 17–30 Registering Callbacks Invoked Before and After Performing Oracle Streams AQ Operations

The `AQxmlCallback` interface has the following methods:

```
public interface AQxmlCallback
{

  /** Callback invoked before any Oracle Streams AQ operations are performed by
  the servlet */
  public void beforeAQOperation(HttpServletRequest request,
                                HttpServletResponse response,
                                AQxmlCallbackContext ctx);

  /** Callback invoked after any Oracle Streams AQ operations are performed by
  the servlet */
  public void afterAQOperation(HttpServletRequest request,
                               HttpServletResponse response,
                               AQxmlCallbackContext ctx);
}
```

The callbacks are passed in the HTTP request and response streams and an `AQxmlCallbackContext` object. The object has the following methods:

- `The java.sql.Connection getDBConnection()` method gives a handle to the database connection that is used by the servlet for performing Oracle Streams AQ operations. Users can perform other SQL operations in the callback functions using this connection object.
- You cannot call `close()`, `commit()`, or `rollback() methods` on this connection object.
- `org.w3c.org.Document parseRequestStream()` gives a DOM document representing the parsed request stream.
- `The void setStyleSheet(String type,String href)` method allows the user to set the style sheet for a particular call. So instead of specifying a single style sheet for all responses from this servlet, users can set style sheets for specific responses.

  The style sheet specified in the callback overrides the style sheet (if any) specified for the servlet in the `init()` method

Example 17–31 Inserting a Row in the EMP Table by Creating a Callback Class and Associating it with a Servlet

Before any Oracle Streams AQ operation in the servlet, you want to insert a row in the EMP table. Do this by creating a callback class and associating it with a particular servlet as follows:

```
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.AQ.xml.*;
import java.sql.*;
import javax.jms.*;

/**
 * This is a sample Oracle Streams AQ Servlet callback
 */
public class TestCallback implements oracle.AQ.xml.AQxmlCallback
{

  /** Callback invoked before any Oracle Streams AQ operations are performed by
  the servlet */
  public void beforeAQOperation(HttpServletRequest request,
                                HttpServletResponse response,
                                AQxmlCallbackContext ctx)
  {
      Connection conn = null;
      System.out.println("Entering BeforeAQ Callback ...");

      try
      {
            // Get the connection object from the callback context
            conn = ctx.getDBConnection();

            // Insert value in the EMP table
            PreparedStatement pstmt = conn.prepareStatement (
               "insert into EMP (EMPNO, ENAME) values (100, 'HARRY')");
            pstmt.execute ();
            pstmt.close();
      }
      catch (Exception ex)
      {
          System.out.println("Exception ex: " + ex);
      }
  }

  /** Callback invoked after any Oracle Streams AQ operations are performed by
```

```
  the servlet */
  public void afterAQOperation(HttpServletRequest request,
                             HttpServletResponse response,
                             AQxmlCallbackContext ctx)
  {
      System.out.println("Entering afterAQ Callback ...");

      try
      {
         // Set style sheet for response
         ctx.setStyleSheetProcessingInstr(
             "type='text/xsl href='http://sun-248/AQ/xslt23.html'");
      }
      catch (Exception aq_ex)
      {
          System.out.println("Exception: " + ex);
      }
  }
}

/* Sample Oracle Streams AQ servlet - using user-defined callbacks */
public class AQTestServlet extends oracle.AQ.xml.AQxmlServlet
{
  /* The init method must be overloaded to specify the AQxmlDataSource */
  public void init()
  {
      AQxmlDataSource  db_drv = null;
      AQxmlCallback    serv_cbk  = new TestCallback();

      try
      {
        /* Create data source with username, password, sid, host, port */
        db_drv = new AQxmlDataSource("AQADM", "AQADM", "test_db", "sun-248",
             "5521");

        this.setAQDataSource(db_drv);

        /* Set Callback */
        setUserCallback(serv_cbk);
      }
      catch (Exception ex)
      {
          System.out.println("Exception in init: " + ex);
      }
}
```

Frequently Asked Questions: Using Oracle Streams AQ and the Internet

The following frequently asked questions cover using Oracle Streams AQ and the Internet and Oracle Internet Directory.

Internet Access Questions

What is IDAP?

IDAP is Internet Data Access Presentation. IDAP defines the message structure for the body of a SOAP request. An IDAP message encapsulates the Oracle Streams AQ request and response in XML. IDAP is used to perform Oracle Streams AQ 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.

Which Web servers are supported for Oracle Streams AQ Internet access functionality? Must I use Apache or can I use any Web server? Which servlet engines are supported for Oracle Streams AQ Internet access? Can I use Tomcat?

Internet access functionality for Oracle Streams AQ is supported on Apache. 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.

How does an Internet agent tie to an Oracle Streams AQ agent stored in Oracle Internet Directory?

You can create an alias to an Oracle Streams AQ agent in Oracle Internet Directory. You can use these Oracle Streams AQ agent aliases in the IDAP document sent over the Internet to perform Oracle Streams AQ operations. Using aliases prevents exposing the internal name of the Oracle Streams AQ agent.

Can I use my own authentication framework for authentication?

Yes, you can use your own authentication framework for authentication. HTTP `POST` requests to the Oracle Streams AQ Servlet for Oracle Streams AQ operations must be authenticated by the Web server. For example, in Apache, the following can be used to restrict access (using basic authentication) to servlets installed under aqserv/servlet. In this example, all users sending `POST` requests to the servlet are authenticated using the users file in `/apache/htdocs/userdb`.

```
<Location /aqserv/servlet>
<Limit POST>
AuthName "Restrict AQ Servlet Access"
AuthType Basic
AuthUserFile /apache/htdocs/userdb/users
require valid-user
</Limit>
</Location>
```

Oracle Internet Directory Questions

Which events can be registered in Oracle Internet Directory?

All types of events—system events, user events, and notifications on queues—can be registered 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.

How do I use agent information stored in an Oracle Internet Directory?

You can create aliases for an Oracle Streams AQ agent in Oracle Internet Directory. These aliases can be specified while performing Oracle Streams AQ operations-enqueue, dequeue, and notifications. This is specifically useful while performing Oracle Streams AQ operations over the Internet when you do not want to expose an internal agent name. An alias can be used in an Oracle Streams AQ operation (IDAP request).

Part VII

Using Messaging Gateway

Part VII describes Messaging Gateway and how to use it.

This part contains the following chapters:

- Chapter 18, "Introducing Oracle Messaging Gateway"
- Chapter 19, "Getting Started with Oracle Messaging Gateway"
- Chapter 20, "Working with Oracle Messaging Gateway"
- Chapter 21, "Oracle Messaging Gateway Message Conversion"
- Chapter 22, "Monitoring Oracle Messaging Gateway"

18

Introducing Oracle Messaging Gateway

This chapter introduces Oracle Messaging Gateway (MGW) features and functionality.

This chapter contains these topics:

- Introducing Oracle Messaging Gateway
- Oracle Messaging Gateway Features
- Oracle Messaging Gateway Architecture
- Propagation Processing Overview

Introducing Oracle Messaging Gateway

MGW enables communication between applications based on non-Oracle messaging systems and Oracle Streams AQ.

Oracle Streams AQ provides **propagation** between two Oracle Streams AQ queues to enable e-business (HTTP through **IDAP**). MGW extends this to applications based on non-Oracle messaging systems.

Because MGW is integrated with Oracle Streams AQ and Oracle Database, it offers reliable **message** delivery. MGW guarantees that messages are delivered once and only once between Oracle Streams AQ 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 Streams AQ.

This release of MGW supports the integration of Oracle Streams AQ with applications based on WebSphere MQ 5.3, TIB/Rendezvous 6.9 and TIB/Rendezvous 7.2.

Oracle Messaging Gateway Features

MGW provides the following features:

- Extends Oracle Streams AQ message propagation

  MGW propagates messages between Oracle Streams AQ and non-Oracle messaging systems. Messages sent by Oracle Streams AQ applications can be received by non-Oracle messaging system applications. Conversely, messages published by non-Oracle messaging system applications can be consumed by Oracle Streams AQ applications.

  See Also: "Propagation Processing Overview" on page 18-6 and Chapter 21, "Oracle Messaging Gateway Message Conversion"

- Support for **Java Message Service** (JMS) messaging systems

  MGW propagates messages between Oracle Java Message Service (Oracle JMS) and WebSphere MQ Java Message Service (WebSphere MQ JMS).

- Native message format support

  MGW supports the native message formats of messaging systems. Oracle Streams AQ 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

  MGW facilitates message conversion between Oracle Streams AQ messages and non-Oracle messaging system messages. Messages are converted through either automatic routines provided by MGW or customized message **transformation** functions that you provide.

  > **Note:** MGW does not support message propagation between JMS and non-JMS messaging systems.

  > **See Also:** "Converting Oracle Messaging Gateway Non-JMS Messages" on page 21-2

- Integration with Oracle Database

  MGW is managed through a PL/SQL interface similar to that of Oracle Streams AQ. 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 MGW 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

  MGW supports client authentication of Oracle Database and non-Oracle messaging systems.

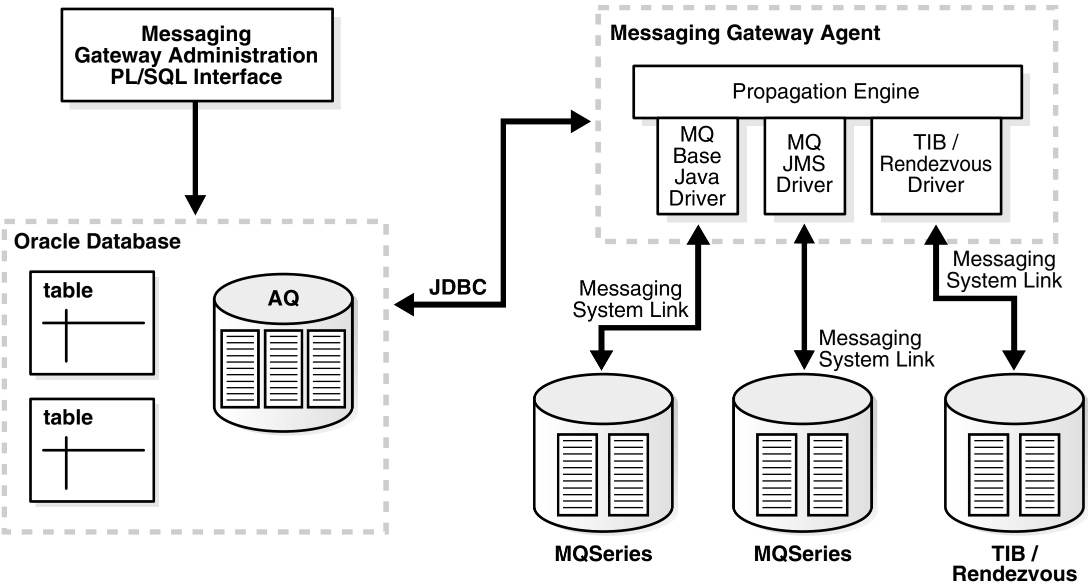
Oracle Messaging Gateway Architecture

MGW has two main components:

- Administration Package DBMS_MGWADM
- MGW Agent

Figure 18–1 shows how these components work together with Oracle Database and non-Oracle messaging systems.

Figure 18–1 MGW Architecture



Administration Package DBMS_MGWADM

The MGW administration package `DBMS_MGWADM` provides an interface for managing the MGW agent, creating messaging system links, registering non-Oracle queues, and setting up propagation.

Users call the procedures in the package to make configuration changes regardless of whether the MGW agent is running. If the MGW 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 MGW agent receives and processes notifications in the same order as they are made.

> **See Also:** *PL/SQL Packages and Types Reference* for more information on `DBMS_MGWADM`

Oracle Messaging Gateway Agent

The MGW 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 MGW 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, MGW provides a mechanism of message propagation between Oracle Streams AQ and non-Oracle messaging systems. Oracle Streams AQ is involved in every propagation job as either propagation source or propagation destination.

MGW is managed through the PL/SQL administration package DBMS_MGWADM. All configuration information and execution state information of MGW are stored in Oracle Database and can be accessed through database views.

The MGW 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 MGW agent connects to non-Oracle messaging systems through messaging system links. Messaging system links are communication channels between the MGW 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 MGW. 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 20-12

Propagation Processing Overview

To propagate messages, propagation jobs must be created. A propagation job consists of a propagation **subscriber** and a propagation schedule. A propagation subscriber defines the source queue and destination queue of a propagation job. A propagation schedule controls when the propagation job is processed.

If the propagation source is a queue (point-to-point), then the MGW agent moves all messages in the queue to the destination. If the propagation source is a topic (**publish/subscribe**), then the MGW 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 its schedule is enabled. Disabling a propagation schedule stops propagation job processing, but does not stop message subscription.

When the MGW 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. MGW 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 MGW 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 MGW 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, MGW requires logging queues in messaging systems that support transactions and persistent messages. The MGW 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 20-15

19

Getting Started with Oracle Messaging Gateway

This chapter describes Oracle Messaging Gateway (MGW) prerequisites and how to load, set up, and unload MGW. 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

MGW requires two job queue processes in addition to those used for other purposes. You can set the number of job queue processes in the `init`*`sid`*`.ora` file, where *`sid`* is the Oracle system ID of the database instance used for MGW:

```
JOB_QUEUE_PROCESSES = num_of_processes
```

Loading and Setting Up Oracle Messaging Gateway

Perform the following procedures before running MGW:

- Loading Database Objects into the Database
- Modifying listener.ora for the External Procedure (Solaris Operating System 32-Bit Only)
- Modifying tnsnames.ora for the External Procedure (Solaris Operating System 32-Bit Only)
- 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

Note: These setup instructions are specific to 32-bit versions of the Windows and Solaris operating systems. The tasks apply to both Windows and Solaris operating systems, except where "Windows Operating System Only" or "Solaris Operating System Only" is indicated. For other operating systems, see *Oracle Database Installation Guide Release 10.1 for UNIX Systems: AIX-Based Systems, HP Tru64 UNIX, HP 9000 Series HP-UX, Linux Intel and Sun Solaris*.

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 MGW, including roles, tables, views, object types, and PL/SQL packages. It creates public synonyms for MGW 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 (Solaris Operating System 32-Bit Only)

Static service information for the listener is not necessary on the Windows operating system.

You must modify `listener.ora` so that the MGW 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 19–1 are important to MGW and must be specified according to your own situation.

Table 19–1 SID_DESC Parameters

Parameter	Description
`SID_NAME`	The SID that is specified in the net service name in `tnsnames.ora`. In the following example, the `SID_NAME` is `mgwextproc`.
`ENVS`	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`*`/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 19-7.
`ORACLE_HOME`	Your Oracle home directory. Using `$ORACLE_HOME` does not work.
`PROGRAM`	The name of the external procedure agent, which is `extproc`

Note: *JRE_HOME* represents the root directory of a JRE installation, just as *ORACLE_HOME* represents the root directory of an Oracle installation. Oracle recommends that you use the JRE installed with Oracle Database.

Example 19–1 adds `SID_NAME mgwextproc` to a listener.ora file.

Example 19–1 Adding Static Service Information for a Listener

```
# Add a SID_DESC
SID_LIST_LISTENER= (SID_LIST=
(SID_DESC =
   (SID_NAME= mgwextproc)
   (ENVS="LD_LIBRARY_PATH=JRE_HOME/lib/sparc/server:ORACLE_HOME/lib")
   (ORACLE_HOME=ORACLE_HOME)
   (PROGRAM = extproc))
      .
      .
      .
```

Modifying tnsnames.ora for the External Procedure (Solaris Operating System 32-Bit 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 19–2. 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 19–2 Configuring MGW_AGENT

```
MGW_AGENT =
(DESCRIPTION=
   (ADDRESS_LIST= (ADDRESS= (PROTOCOL=IPC)(KEY=EXTPROC)))
   (CONNECT_DATA= (SID=mgwextproc) (PRESENTATION=RO)))
```

> **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 MGW initialization file *`ORACLE_HOME`*`/mgw/admin/mgw.ora` is a text file. The MGW external procedure uses it to get initialization parameters to start the MGW 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 MGW:

1. **Windows Operating System Only:** Set the `MGW_PRE_PATH` variable. Its value is the path to the `jvm.dll` library:

   ```
   set MGW_PRE_PATH = JRE_HOME\bin\client
   ```

   This variable is prepended to the path inherited by the MGW agent process.

2. Set `CLASSPATH` to include at least the following:

   - MGW classes:

     ```
     ORACLE_HOME/mgw/classes/mgw.jar
     ```

   - JRE runtime classes:

     ```
     JRE_HOME/lib/rt.jar
     ```

   - Oracle JDBC classes:

     ```
     ORACLE_HOME/jdbc/lib/ojdbc14.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/aqapi13.jar

- Java transaction **API**

  ORACLE_HOME/jlib/jta.jar

- Any additional classes needed for MGW to access non-Oracle messaging systems

  See Also: "Setting Up Non-Oracle Messaging Systems" on page 19-7

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 MGW administration work, a database user must be created with MGW_ADMINISTRATOR_ROLE privileges, as shown in Example 19–3.

Example 19–3 Creating a MGW Administrator User

```
CREATE USER admin_user IDENTIFIED BY admin_password;
GRANT CONNECT, RESOURCE to admin_user;
GRANT MGW_ADMINISTRATOR_ROLE to admin_user;
```

Creating an Oracle Messaging Gateway Agent User

To establish the MGW agent connection back to the database, a database user with MGW_AGENT_ROLE privileges must be created, as shown in Example 19–4.

Example 19–4 Creating an MGW Agent User

```
CREATE USER agent_user IDENTIFIED BY agent_password;
GRANT CONNECT, RESOURCE to agent_user;
GRANT MGW_AGENT_ROLE to agent_user;
```

Configuring Oracle Messaging Gateway Connection Information

After the MGW agent user is created, the administration user uses `DBMS_MGWADM.DB_CONNECT_INFO` to configure MGW with the username, password, and database connect string used by the MGW agent to connect back to the database, as shown in Example 19–5. Use the MGW username and password that you created in "Creating an Oracle Messaging Gateway Agent User" on page 19-6. 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 Oracle Database release 10.1, always specify a not `NULL` value for the database connect string parameter when calling `DBMS_MGWADM.DB_CONNECT_INFO`.

Example 19–5 Configuring MGW Connection Information

```
connect admin_user/admin_password
exec dbms_mgwadm.db_connect_info('agent_user','agent_password', 'agent_
database');
```

Configuring Oracle Messaging Gateway in a RAC Environment

Setting the RAC Instance for the MGW Agent

The `DBMS_MGWADM.STARTUP` procedure submits a job queue job that starts the MGW agent external process when the job is executed. You can use the `instance` and `force` parameters to control the job and instance affinity. By default the job is set up so that it can be run by any instance.

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 MGW agent requires TIB/Rendezvous software to be installed on the computer where the MGW agent runs. In this section *TIBRV_HOME* refers to the installed TIB/Rendezvous software location.

Modifying listener.ora

On the Solaris operating system, LD_LIBRARY_PATH in the entry for MGW must include *TIBRV_HOME*/lib for the agent to access TIB/Rendezvous shared library files.

> **See Also:** "Modifying listener.ora for the External Procedure (Solaris Operating System 32-Bit Only)" on page 19-3

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 19-10

Example 19–6 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 MGW agent runs. In this section *MQ_HOME* refers to the location of the installed client. On the Solaris 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 MGW 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 19-5):

- *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 19-5):

- *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 MGW 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 MGW agent user.

   Run sqlplus *agent_user*/*agent_password*@*agent_database*.

   If it is successful, then the MGW agent is able to connect to the database.

3. **Solaris 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 MGW 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 MGW 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 MGW:

1. Shut down MGW.
2. Remove any user-created queues whose payload is an MGW **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 MGW, including roles, tables, views, packages, object types, and synonyms.

4. Remove entries for MGW created in `listener.ora` and `tnsnames.ora`.

Understanding the mgw.ora Initialization File

MGW reads initialization information from a text file named `mgw.ora` when the MGW agent starts. The `mgw.ora` file is located in *`ORACLE_HOME`*`/mgw/admin`.

The MGW initialization file `mgw.ora` 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.

This section contains these topics:

- mgw.ora Initialization Parameters
- mgw.ora Environment Variables
- mgw.ora Java Properties
- mgw.ora Comment Lines

Note: Each of the following examples must consist of only one line in the initialization file, although it can appear otherwise in this document.

---

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 MGW 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 MGW agent. The logging level can be dynamically changed by the `dbms_mgwadm.set_log_level` API while the MGW 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 MGW 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 MGW agent are `CLASSPATH`, `MGW_PRE_PATH`, and `ORACLE_SID`.

Environment variables such as `CLASSPATH` and `MGW_PRE_PATH` are set so the MGW 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 MGW agent for **propagation** between Oracle Streams AQ and non-Oracle messaging systems.

Format: `set CLASSPATH=`*`value`*

Example: `set CLASSPATH=`*`ORACLE_HOME`*`/jdbc/lib/ojdbc14.jar:`*`JRE_HOME`*`/lib/rt.jar:`*`ORACLE_HOME`*`/sqlj/lib/runtime12.jar:`*`ORACLE_HOME`*`/jlib/orai18n.jar:`*`ORACLE_HOME`*`/mgw/classes/mgw.jar:`*`ORACLE_HOME`*`/rdbms/jlib/jmscommon.jar:`*`ORACLE_HOME`*`/rdbms/jlib/aqapi13.jar:`*`ORACLE_HOME`*`/jlib/jta.jar:/opt/mqm/java/lib/com.ibm.mq.jar:/opt/mqm/java/lib/com.ibm.mqjms.jar:/opt/mqm/java/lib/connector.jar`

MGW_PRE_PATH

Usage: Appended to the front of the path inherited by the MGW 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 MGW.

Format: `set ORACLE_SID=`*`value`*

Example: `set ORACLE_SID=my_sid`

mgw.ora Java Properties

You must specify Java system properties for the MGW JVM when working with TIB/Rendezvous subjects. You can use the `setJavaProp` parameter of the MGW 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 MGW **subscriber** option, `MsgBatchSize`, is not specified. If altered from the default, then consideration should be given to the expected **message** size and the MGW 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.

> **See Also:** "DBMS_MGWADM" in *PL/SQL Packages and Types Reference*

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 MGW 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`

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 MGW is used, which writes system advisory messages into MGW 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`

mgw.ora Comment Lines

Comment lines are designated with a # character as the first character of the line.

20

Working with Oracle Messaging Gateway

After Oracle Messaging Gateway (MGW) is loaded and set up, it is ready to be configured and run. This chapter describes how to manage the MGW agent and how to configure **propagation**.

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

Note: All commands in the examples must be run as a user granted `MGW_ADMINISTRATOR_ROLE`.

See Also: "DBMS_MGWADM" and "DBMS_MGWMSG" in *PL/SQL Packages and Types Reference*

Configuring the Oracle Messaging Gateway Agent

Messages are propagated between Oracle Streams AQ and non-Oracle messaging systems by the MGW agent. The MGW agent runs as an external process of the Oracle Database server.

You must set the following information in order for the agent to start:

- Database Connection
- Resource Limits

Database Connection

The MGW agent runs as a process external to the database. To access Oracle Streams AQ and the MGW packages, the MGW agent needs to establish connections to the database. You can use `DBMS_MGWADM.DB_CONNECT_INFO` to set the username, password and the database connect string that the MGW agent will use for creating database connections. The user must be granted the role `MGW_AGENT_ROLE` before the MGW agent can be started.

You can call `DBMS_MGWADM.DB_CONNECT_INFO` to alter connection information when the MGW agent is running.

Example 20–1 shows MGW being configured for user `mgwagent` with password `mgwagent_password` using net service name `mydatabase`.

Example 20–1 Setting Database Connection Information

```
SQL> exec dbms_mgwadm.db_connect_info(username => 'mgwagent',
                                      password => 'mgwagent_password',
                                      database => 'mydatabase');
```

Resource Limits

You can use `DBMS_MGWADM.ALTER_AGENT` to set the maximum number of messaging connections used by the MGW agent, the heap size of the MGW agent process, and the number of **propagation** threads in the agent process. The default values are one connection, 64 MB of memory heap, and one propagation thread.

Example 20–2 sets the number of database connections to two, the heap size to 64MB, and the number of propagation threads to two.

Example 20–2 Setting the Resource Limits

```
SQL> exec dbms_mgwadm.alter_agent(max_connections => 2,
                                  max_memory => 64,
                                  max_threads => 2);
```

You can alter the maximum number of connections when the MGW agent is running. The memory heap size and the number of propagation threads cannot be altered when the MGW agent is running. Example 20–3 updates the maximum number of connections to three but leaves the maximum memory and the number of propagation threads unchanged.

Example 20–3 Updating the Maximum Connection Number

```
SQL> exec dbms_mgwadm.alter_agent(max_connection => 3);
```

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 Job Queue Job
- Running the Oracle Messaging Gateway Agent on RAC

Starting the Oracle Messaging Gateway Agent

After the MGW agent is configured, you can start the agent with `DBMS_MGWADM.STARTUP`, as shown in Example 20–4.

Example 20–4 Starting the MGW Agent

```
SQL> exec dbms_mgwadm.startup;
```

You can use the `MGW_GATEWAY` view to check the status of the MGW agent, as described in Chapter 22, "Monitoring Oracle Messaging Gateway".

Shutting Down the Oracle Messaging Gateway Agent

You can use DBMS_MGWADM.SHUTDOWN to shut down the MGW agent, as shown in Example 20–5.

Example 20–5 Shutting Down the MGW Agent

```
SQL> exec dbms_mgwadm.shutdown;
```

You can use the MGW_GATEWAY view to check if the MGW agent has shut down successfully, as described in Chapter 22, "Monitoring Oracle Messaging Gateway".

Oracle Messaging Gateway Agent Job Queue Job

MGW uses a job queue job to start the MGW 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 MGW 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

The DBMS_JOB package creates a repeatable job with a repeat interval of two minutes. The job is owned by SYS. A repeatable job enables the MGW 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 an MGW agent job runs at a given time.

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.

MGW uses a database shutdown trigger. If the MGW 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 an MGW 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, the job identifier, and the database instance on which the MGW agent is currently running.

See Also:

- "DBMS_JOB" in *PL/SQL Packages and Types Reference*
- Chapter 22, "Monitoring Oracle Messaging Gateway"

Running the Oracle Messaging Gateway Agent on RAC

While the MGW job startup and shutdown principles are the same for RAC and non-RAC environments, there are some things to keep in mind for a RAC environment.

Only one MGW agent process can be running at a given time, even in a RAC environment. The job scheduler determines which database instance will run a job based on parameters specified when the job is created. The `DBMS_MGWADM.STARTUP` procedure has two optional parameters, `instance` and `force`, that can be used to set the instance affinity of the MGW agent job.

When a database instance is shut down in a RAC environment, the MGW shutdown trigger will notify the agent to shut down only if the MGW 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.

See Also: "DBMS_MGWADM" and "DBMS_JOB" in *PL/SQL Packages and Types Reference*

Configuring Messaging System Links

Running as a client of non-Oracle messaging systems, the MGW agent communicates with non-Oracle messaging systems through messaging system links. A messaging system link is a set of connections between the MGW 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.

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 MGW 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.

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 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_MSGSYSTEM_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 20–6 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`.

Example 20–6 Configuring a WebSphere MQ Base Java Link

```
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', properties => v_prop, options => v_options );
end;
```

See Also:

- "Understanding the mgw.ora Initialization File" on page 19-10 for information on setting the CLASSPATH of the MGW agent
- "DBMS_MGWADM" in *PL/SQL Packages and Types Reference* for information on WebSphere MQ system properties and supported options

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. Each type of connection can be used only for operations involving that JMS destination type:

  - A WebSphere MQ JMS queue link must be created with interface type `DBMS_MGWADM.JMS_QUEUE_CONNECTION` to access WebSphere MQ JMS queues
  - A WebSphere MQ JMS topic link must be created with interface type `DBMS_MGWADM.JMS_TOPIC_CONNECTION` to access WebSphere MQ JMS topics

- 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

  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 match the link type:

  - For a WebSphere MQ JMS 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 queue 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 20–7 configures an MGW 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.

Example 20–7 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', properties => v_prop, options => v_options );
end;
```

See Also:

- "Registering a WebSphere MQ JMS Queue or Topic" on page 20-13 for more information on JMS queues and topics
- "DBMS_MGWADM" in *PL/SQL Packages and Types Reference* for information on `DBMS_MGWADM.create_msgsystem_link`

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, MGW can only guarantee at-most-once message delivery.

Example 20–8 configures a TIB/Rendezvous link named `'rvlink'` that connects to the `rvd` daemon on the local computer.

Example 20–8 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;
```

See Also: "DBMS_MGWADM" in *PL/SQL Packages and Types Reference* for information on TIB/Rendezvous system properties and supported options

Altering a Messaging System Link

Some link information can be altered after the link is created. You can alter link information with the MGW agent running or shut down. Example 20–9 alters the link `'mqlink'` to change the `max_connections` and `password` properties.

Example 20–9 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();
  v_prop.max_connections := 2;
  v_prop.password := 'newpasswd';

  dbms_mgwadm.alter_msgsystem_link(
    linkname => 'mqlink', properties => v_prop);
end;
```

> **See Also:** "DBMS_MGWADM" in *PL/SQL Packages and Types Reference* for restrictions on changes when the MGW agent is running

Removing a Messaging System Link

You can remove an MGW 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 MGW agent running or shut down. Example 20–10 removes the link `'mqlink'`.

Example 20–10 Removing an MGW Link

```
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 20–11.

Example 20–11 Listing All MGW Links

```
SQL> select link_name, link_type from MGW_LINKS;

LINK_NAME     LINK_TYPE
------------------------
MQLINK        MQSERIES
RVLINK        TIBRV
```

You can use the MGW_MQSERIES_LINK and MGW_TIBRV_LINKS views to check messaging system type-specific configuration information, as shown in Example 20–12.

Example 20–12 Checking Messaging System Link Configuration Information

```
SQL> select link_name, queue_manager, channel, hostname from mgw_mqseries_link;

LINK_NAME   QUEUE_MANAGER      CHANNEL     HOSTNAME
----------------------------------------------------------
MQLINK      my.queue.manager  mychannel  myhost.mydomain

SQL> select link_name, service, network, daemon from mgw_tibrv_links;

LINK_NAME   SERVICE      NETWORK     DAEMON
-----------------------------------------------------
RVLINK
```

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 MGW administration interface. You do not need to register Oracle Streams AQ 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

Registering a non-Oracle queue provides information for the MGW 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 MGW 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: "DBMS_MGWADM" in *PL/SQL Packages and Types Reference* for optional foreign queue configuration properties

Example 20–13 shows how to register the WebSphere MQ Base Java queue `my_mq_queue` as an MGW queue `'destq'`.

Example 20–13 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 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 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

The domain of a registered TIB/Rendezvous queue must be `DBMS_MGWADM.DOMAIN_TOPIC` or NULL. The `provider_queue` of the queue is a TIB/Rendezvous subject.

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 MGW 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 subscribers or schedules referencing it.

Example 20–14 unregisters the queue `'destq'` of the link `'mqlink'`.

Example 20–14 Unregistering a Non-Oracle Queue

```
begin
  dbms_mwgadm.unregister_foreign_queue(name =>'destq', link_name=>'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 20–15.

Example 20–15 Checking Which Queues Are Registered

```
SQL> select name, link_name, provider_queue from MGW_FOREIGN_QUEUES;

NAME   LINK_NAME   PROVIDER_QUEUE
-----------------------------------
DESTQ  MQLINK      my_mq_queue
```

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. A propagation job consists of a propagation subscriber and a propagation schedule. The propagation subscriber specifies the source and destination queues, while the propagation schedule specifies when the propagation job is processed. A propagation schedule is associated with a propagation subscriber that has the same propagation source, destination, and type.

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. MGW does not support message propagation between a JMS destination and a non-JMS queue.

This section contains these topics:

- Propagation Subscriber Overview
- Creating an Oracle Messaging Gateway Propagation Subscriber
- Creating an Oracle Messaging Gateway Propagation Schedule
- Enabling and Disabling a Propagation Job
- Resetting a Propagation Job
- Altering a Propagation Subscriber and Schedule
- Removing a Propagation Subscriber and Schedule

Propagation Subscriber Overview

A propagation subscriber specifies what messages are propagated and how the messages are propagated.

MGW allows bi-directional message propagation. An outbound propagation moves messages from Oracle Streams AQ to non-Oracle messaging systems. An inbound propagation moves messages from non-Oracle messaging systems to Oracle Streams AQ.

If the propagation source is a queue (point-to-point), then the MGW agent moves all messages from the source queue to the destination queue. If the propagation source is a topic (publish/subscribe), then the MGW 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 MGW 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.

An MGW exception queue is different from an Oracle Streams AQ exception queue. MGW moves a message to an MGW exception queue when message conversion fails. Oracle Streams AQ moves a message to an Oracle Streams AQ exception queue after `MAX_RETRIES` dequeue attempts on the message.

Messages moved to an Oracle Streams AQ exception queue may result in unrecoverable failures on the associated MGW subscriber. To avoid the problem, the `MAX_RETRIES` parameter of any Oracle Streams AQ queue that is used as the propagation source of an MGW 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 subscriber options for certain types of propagation subscribers to control how messages are propagated, such as options for **JMS message** delivery mode and TIB/Rendezvous queue policies.

MGW provides `MGW_SUBSCRIBERS` and `MGW_SCHEDULES` views for users to check configuration and status of MGW subscribers and schedules.

See Also: Chapter 22, "Monitoring Oracle Messaging Gateway"

Creating an Oracle Messaging Gateway Propagation Subscriber

MGW subscribers are created by `DBMS_MGWADM.ADD_SUBSCRIBER`.

If the propagation source for non-JMS propagation is an Oracle Streams AQ queue, then the queue must be a multiconsumer queue. MGW creates a corresponding Oracle Streams AQ subscriber `'MGW_subscriber_id'` for the messaging system subscriber `subscriber_id` when `DBMS_MGWADM.ADD_SUBSCRIBER` 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_subscriber_id'` is created on the topic in the source messaging system by the MGW 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 20–16 creates MGW propagation subscriber `sub_aq2mq`.

Example 20–16 Creating a Propagation Subscriber

```
begin
  dbms_mgwadm.add_subscriber(
    subscriber_id => 'sub_aq2mq',
    propagation_type => dbms_mgwadm.outbound_propagation,
    queue_name => 'mgwuser.srcq',
    destination => 'destq@mqlink');
end;
```

Note: If a WebSphere MQ JMS topic is involved in a propagation job, then a durable subscriber `MGL_subscriber_id` is created on the log topic. The durable subscriber is removed when the MGW subscriber is successfully removed.

Creating an Oracle Messaging Gateway Propagation Schedule

A propagation subscriber is not processed until an associated propagation schedule is created and enabled. A propagation schedule is associated with a propagation subscriber when the propagation type, source and destination match.

The `latency` parameter in a propagation schedule controls the polling interval of a propagation job. The polling interval determines how soon the agent can discover the available messages to propagate in the propagation source queue. The default polling interval is 5 seconds or the value set for `oracle.mgw.polling_interval` in MGW initialization file `mgw.ora`.

Example 20–17 creates MGW propagation schedule `sch_aq2mq`.

Example 20–17 Creating a Propagation Schedule

```
begin
  dbms_mgwadm.schedule_propagation(
    schedule_id => 'sch_aq2mq',
    propagation_type => dbms_mgwadm.outbound_propagation,
    source => 'mgwuser.srcq',
```

```
    destination => 'destq@mqlink',
    latency => 2);
end;
```

Enabling and Disabling a Propagation Job

A propagation job is enabled if its propagation schedule is created and enabled. A propagation job is disabled if its propagation schedule is disabled or removed. Users can call `DBMS_MGWADM.ENABLE_PROPAGATION_SCHEDULE` to enable a propagation schedule and `DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE` to disable a propagation schedule.

Example 20–18 enables the propagation schedule for propagation subscriber `sub_aq2mq`.

Example 20–18 Enabling an MGW Propagation Schedule

```
begin
  dbms_mgwadm.enable_propagation_schedule('sch_aq2mq');
end;
```

Example 20–19 disables the propagation schedule for propagation subscriber `sub_aq2mq`.

Example 20–19 Disabling an MGW Propagation Schedule

```
begin
  dbms_mgwadm.disable_propagation_schedule('sch_aq2mq');
end;
```

By default, the propagation schedule is enabled when it is first created.

To create a propagation job that is initially disabled, call the following APIs in the indicated order:

1. `DBMS_MGWADM.SCHEDULE_PROPAGATION`
2. `DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE`
3. `DBMS_MGWADM.ADD_SUBSCRIBER`

Resetting a Propagation Job

When a problem occurs with a propagation job, the MGW 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_SUBSCRIBER` to reset the failure count to zero to allow the agent to retry the failed operation immediately.

Example 20–20 resets the failure count for propagation subscriber `sub_aq2mq`.

Example 20–20 Resetting a Propagation Job

```
begin
  dbms_mgwadm.reset_subscriber('sub_aq2mq');
end;
```

Altering a Propagation Subscriber and Schedule

After the propagation subscriber and schedule of a propagation job are created, you can alter the selection rule, transformation, exception queue, subscriber options, and latency of the propagation job. Subscribers and schedules can be altered with the MGW agent running or shut down.

Example 20–21 adds an exception queue for subscriber `sub_aq2mq`.

Example 20–21 Altering Propagation Subscriber by Adding an Exception Queue

```
begin
  dbms_mgwadm.alter_subscriber(
    subscriber_id => 'sub_aq2mq',
    exception_queue => 'mgwuser.my_ex_queue');
end;
```

Example 20–22 changes the polling interval for schedule `sch_aq2mq`.

Example 20–22 Altering Propagation Subscriber by Changing the Polling Interval

```
begin
  dbms_mgwadm.alter_propagation_schedule(
    subscriber_id => 'sch_aq2mq',
    latency => 1);
end;
```

Removing a Propagation Subscriber and Schedule

You can remove an MGW propagation subscriber with DBMS_MGWADM.REMOVE_SUBSCRIBER.

Before removing the MGW subscriber from the MGW configuration, MGW does the following cleanup:

- Removes from the messaging system the associated subscriber that may have been created by MGW
- Removes propagation log records from log queues for the subscriber being removed

MGW may fail to do the cleanup because:

- The MGW agent is not running
- Non-Oracle messaging system is not running
- The MGW agent is unable to interact with the source or destination messaging system

If MGW cleanup fails for any reason, then the MGW subscriber being removed is placed in the DELETE_PENDING state. The MGW agent tries to clean up subscribers in DELETE_PENDING state when:

- DBMS_MGWADM.REMOVE_SUBSCRIBER is called and the MGW agent is running
- The MGW agent is starting and finds a subscriber in DELETE_PENDING state

You can specify DBMS_MGWADM.FORCE when calling DBMS_MGWADM.REMOVE_SUBSCRIBER to force MGW to remove the MGW subscriber from the MGW configuration without placing it in the DELETE_PENDING mode in case of cleanup failures.

Calling DBMS_MGWADM.REMOVE_SUBSCRIBER with DBMS_MGWADM.FORCE may result in obsolete log records in the log queues and subscriptions in messaging systems, which may cause unnecessary message accumulation. Oracle recommends not using DBMS_MGWADM.FORCE when calling DBMS_MGWADM.REMOVE_SUBSCRIBER, if possible.

Example 20–23 removes propagation subscriber sub_aq2mq.

Example 20–23 Removing a Propagation Subscriber

```
begin
  dbms_mgwadm.remove_subscriber(subscriber_id =>'sub_aq2mq');
end;
```

You can remove propagation schedules with DBMS_MGWADM.UNSCHEDULE_PROPAGATION. Removing a propagation schedule results in disabling the associated propagation job. It does not remove any subscriptions in messaging systems.

Example 20–24 removes propagation schedule sch_aq2mq.

Example 20–24 Removing a Propagation Schedule

```
begin
  dbms_mgwadm.unschedule_propagation(schedule_id => 'sch_aq2mq');
end;
```

21

Oracle Messaging Gateway Message Conversion

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 Streams AQ 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 Streams AQ for the conversion.

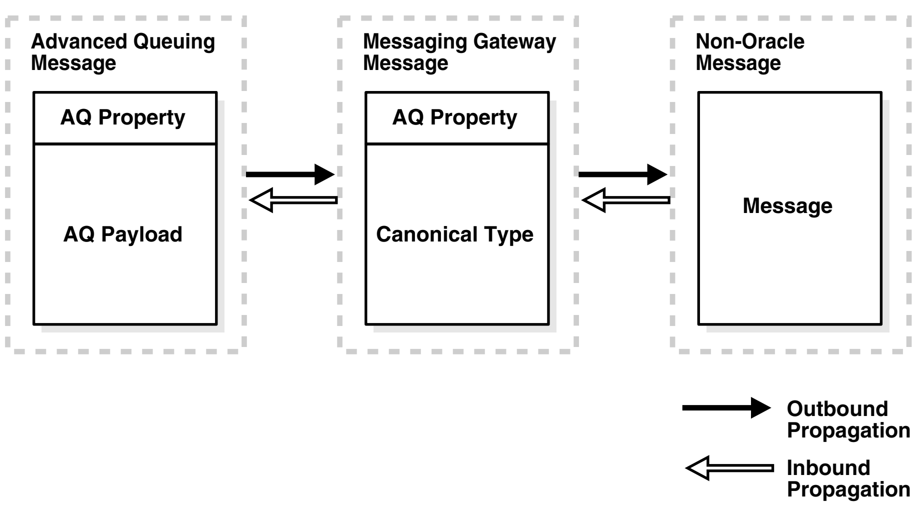
Overview of the Non-JMS Message Conversion Process

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 Streams AQ and the fixed header in WebSphere MQ. The body contains message contents, such as the Oracle Streams AQ payload, the WebSphere MQ message body, or the entire TIB/Rendezvous message. MGW converts both message header and message body components.

Figure 21–1 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.

Figure 21–1 Non-JMS Message Conversion



The MGW agent uses an internal message format consisting of a header that is similar to the Oracle Streams AQ 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 Streams AQ 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 `MGW_BASIC_MSG_T` and `RAW`. TIB/Rendezvous propagation supports the canonical types `MGW_TIBRV_MSG_T` and `RAW`.

> **See Also:** "DBMS_MGWMSG" in *PL/SQL Packages and Types Reference* for
>
> - Syntax and attribute information for `MGW_BASIC_MSG_T`
> - Syntax and attribute information for `MGW_TIBRV_MSG_T`

Message Header Conversion

MGW provides default mappings between Oracle Streams AQ message properties and non-Oracle message header fields that have a counterpart in Oracle Streams AQ 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.

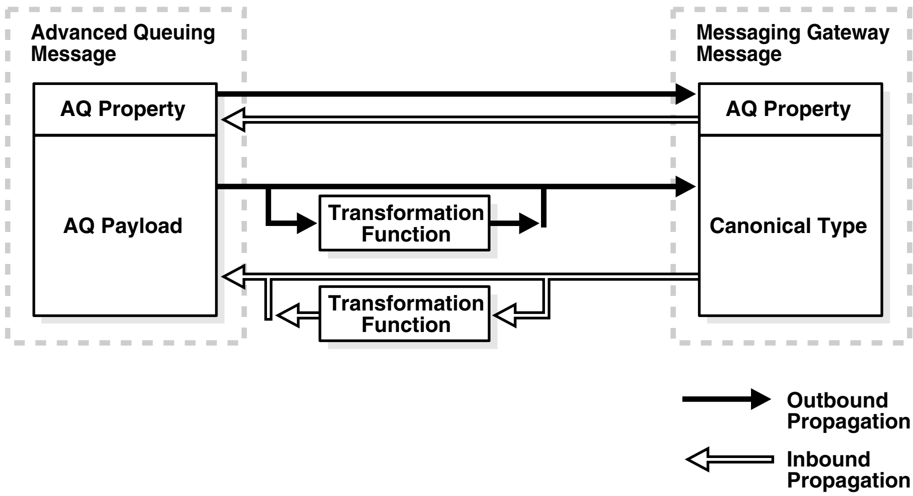
Handling Arbitrary Payload Types Using Message Transformations

When converting to or from Oracle Streams AQ messages, the MGW agent uses only its canonical types. Arbitrary payload types are supported, however, with the assistance of user-defined Oracle Streams AQ message transformations to convert between an Oracle Streams AQ queue payload and an MGW canonical type.

For MGW to propagate messages from an Oracle Streams AQ 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 Streams AQ queue. Similarly, for MGW to propagate messages to an Oracle Streams AQ 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 Streams AQ queue.

Figure 21–2 Oracle Streams AQ Message Conversion



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.DB_CONNECT_INFO`) must have `EXECUTE` privileges on the transformation function and the Oracle Streams AQ 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 an MGW **subscriber** 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.ADD_SUBSCRIBER` to create an MGW subscriber using the transformation, or `DBMS_MGWADM.ALTER_SUBSCRIBER` to alter an existing subscriber.

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 21–1.

> **Note:** All commands in the examples must be run as a user granted `MGW_ADMINISTRATOR_ROLE`, except for the commands to create transformations.

Example 21–1 Transformation Function Signature

```
FUNCTION trans_sampleadt_to_mgw_basic(in_msg IN mgwuser.sampleADT)
RETURN SYS.MGW_BASIC_MSG_T;
```

You can create a transformation using `DBMS_TRANSFORM.CREATE_TRANSFORMATION`, as shown in Example 21–2.

Example 21–2 Creating a Transformation

```
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 subscriber. Example 21–3 creates subscriber `sub_aq2mq`, for whom messages are propagated from Oracle Streams AQ queue `mgwuser.srcq` to non-Oracle messaging system queue `destq@mqlink` using transformation `mgwuser.sample_adt_to_mgw_basic`.

Example 21–3 Registering a Transformation

```
begin
  DBMS_MGWADM.ADD_SUBSCRIBER(
    subscriber_id    => 'sub_aq2mq',
    propagation_type => dbms_mgwadm.outbound_propagation,
    queue_name       => 'mgwuser.srcq',
```

```
    destination      =>'destq@mqlink',
    transformation   => 'mgwuser.sample_adt_to_mgw_basic'),
    exception_queue  => 'mgwuser.excq');
end;
```

> **See Also:** "DBMS_MGWADM", "DBMS_MGWMSG", and "DBMS_TRANSFORM" in *PL/SQL Packages and Types Reference*

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 `SYS.ANYDATA`. An LCR is propagated as an XML string stored in the appropriate message type.

> **Note:** The XDB package must be loaded for LCR propagation.

Because Oracle Streams uses `SYS.ANYDATA` queues to store LCRs, a `SYS.ANYDATA` queue is the source for outbound propagation. The transformation must first convert the `SYS.ANYDATA` object containing an LCR into an XMLType object using the MGW routine `DBMS_MGWMSG.LCR_TO_XML`. If the `SYS.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 XMLType and placed in the appropriate MGW canonical type (`MGW_BASIC_MSG_T` or `MGW_TIBRV_MSG_T`).

Example 21–4 illustrates a simplified transformation used for LCR outbound propagation. The transformation converts a `SYS.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 21–4 Outbound LCR Transformation

```
create or replace function any2tibrv(adata in sys.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 (`MGW_BASIC_MSG_T` or `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 a `SYS.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 21–5 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 a `SYS.ANYDATA` object. The string representing the LCR as an XML document is taken from a field named `'ORACLE_LCR'`.

Example 21–5 Inbound LCR Transformation

```
create or replace function tibrv2any(tdata in sys.mgw_tibrv_msg_t)
return sys.anydata is
    v_field   sys.mgw_tibrv_field_t;
    v_xml     XMLType;
    v_text    varchar2(2000);
    v_any     sys.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;
```

See Also:

- "DBMS_MGWMSG" in *PL/SQL Packages and Types Reference*
- `ORACLE_HOME/mgw/samples/lcr` for complete examples of LCR transformations

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 `MGW_BASIC_MSG_T` and `RAW`.

Figure 21–3 Message Conversion for WebSphere MQ Using MGW_BASIC_MSG_T

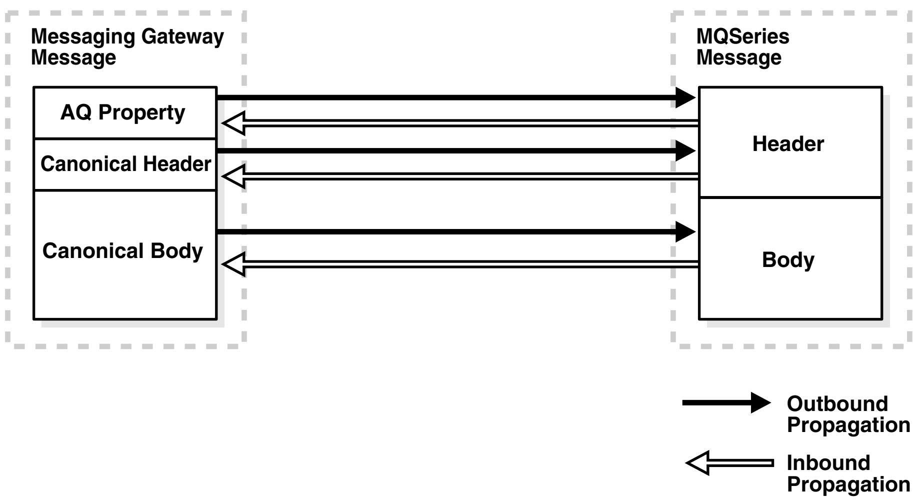


Figure 21–3 illustrates the message conversion performed by the MGW WebSphere MQ driver when using the canonical type `MGW_BASIC_MSG_T`. For outbound propagation, the driver maps the Oracle Streams AQ message properties and canonical message to a WebSphere MQ message having a fixed header and a message body. For inbound propagation, the driver maps a native message to a set of Oracle Streams AQ 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 `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 `MGW_BASIC_MSG_T` as the canonical type. Table 21–1 defines the MGW {name,value} pairs used to describe the WebSphere MQ header properties. The first column refers to valid string values for the `MGW_NAME_VALUE_T.NAME` field in the `MGW_BASIC_MSG_T` header. The second column refers to the `MGW_NAME_VALUE_T.TYPE` value corresponding to the name. (Refer to "Notes on Table 21–1" on page 21-11 for explanations of the numbers in parentheses.)

> **See Also:** "DBMS_MGWMSG" in *PL/SQL Packages and Types Reference*

When a message is dequeued from the WebSphere MQ messaging system, the WebSphere MQ driver generates {name,value} pairs based on the dequeued message header and stores them in the header part of the canonical message of the `MGW_BASIC_MSG_T` type. When a message is enqueued to WebSphere MQ, 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 `MGW_BASIC_MSG_T` canonical message.

Table 21–1 MGW Names for WebSphere MQ Header Values

MGW Name	MGW Type	WebSphere MQ Property Name	Used For
`MGW_MQ_priority`	`INTEGER_VALUE`	`priority`	Enqueue, Dequeue
`MGW_MQ_expiry`	`INTEGER_VALUE`	`expiry`	Enqueue, Dequeue
`MGW_MQ_correlationId`	`RAW_VALUE` (size 24)	`correlationId`	Enqueue (1), Dequeue
`MGW_MQ_persistence`	`INTEGER_VALUE`	`persistence`	Dequeue
`MGW_MQ_report`	`INTEGER_VALUE`	`report`	Enqueue (1), Dequeue
`MGW_MQ_messageType`	`INTEGER_VALUE`	`messageType`	Enqueue, Dequeue
`MGW_MQ_feedback`	`INTEGER_VALUE`	`feedback`	Enqueue, Dequeue

Table 21–1 (Cont.) MGW Names for WebSphere MQ Header Values

MGW Name	MGW Type	WebSphere MQ Property Name	Used For
MGW_MQ_encoding	INTEGER_VALUE	encoding	Enqueue, Dequeue
MGW_MQ_characterSet	INTEGER_VALUE	characterSet	Enqueue, Dequeue
MGW_MQ_format	TEXT_VALUE (size 8)	format	Enqueue (1), Dequeue
MGW_MQ_backoutCount	INTEGER_VALUE	backoutCount	Dequeue
MGW_MQ_replyToQueueName	TEXT_VALUE (size 48)	replyToQueueName	Enqueue, Dequeue
MGW_MQ_replyToQueueManagerName	TEXT_VALUE (size 48)	replyToQueueManagerName	Enqueue, Dequeue
MGW_MQ_userId	TEXT_VALUE (size 12)	userId	Enqueue, Dequeue
MGW_MQ_accountingToken	RAW_VALUE (size 32)	accountingToken	Enqueue (1), Dequeue
MGW_MQ_applicationIdData	TEXT_VALUE (size 32)	applicationIdData	Enqueue (1), Dequeue
MGW_MQ_putApplicationType	INTEGER_VALUE	putApplicationType	Enqueue (1), Dequeue
MGW_MQ_putApplicationName	TEXT_VALUE (size 28)	putApplicationName	Enqueue (1), Dequeue
MGW_MQ_putDateTime	DATE_VALUE	putDateTime	Dequeue
MGW_MQ_applicationOriginData	TEXT_VALUE (size 4)	applicationOriginData	Enqueue (1), Dequeue
MGW_MQ_groupId	RAW_VALUE (size 24)	groupId	Enqueue (1), Dequeue
MGW_MQ_messageSequenceNumber	INTEGER_VALUE	messageSequenceNumber	Enqueue, Dequeue
MGW_MQ_offset	INTEGER_VALUE	offset	Enqueue, Dequeue
MGW_MQ_messageFlags	INTEGER_VALUE	messageFlags	Enqueue, Dequeue
MGW_MQ_originalLength	INTEGER_VALUE	originalLength	Enqueue, Dequeue
MGW_MQ_putMessageOptions	INTEGER_VALUE	putMessageOptions (2)	Enqueue (1)

Notes on Table 21–1

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_MGWADM.REGISTER_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 Streams AQ message properties and WebSphere MQ header fields.

One maps the Oracle Streams AQ 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 Streams AQ 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 Streams AQ message property priority with the WebSphere MQ header field priority. It is described in Table 21–2.

Table 21–2 Default Priority Mappings for Propagation

Propagation Type	Message System	Priority Values									
Outbound	Oracle Streams AQ	0	1	2	3	4	5	6	7	8	9
Outbound	WebSphere MQ	9	8	7	6	5	4	3	2	1	0
Inbound	Oracle Streams AQ	9	8	7	6	5	4	3	2	1	0
Inbound	WebSphere MQ	0	1	2	3	4	5	6	7	8	9

Note: For outbound propagation, Oracle Streams AQ priority values less than 0 are mapped to WebSphere MQ priority 9, and Oracle Streams AQ 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 Streams AQ 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 `MGW_BASIC_MSG_T`, but the source ADT can be any ADT supported by Oracle Streams AQ.

If the Oracle Streams AQ 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 Streams AQ queue payload or transformation target ADT is `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 Streams AQ message properties and the `MGW_BASIC_MSG_T.header` attribute of the canonical message, as described in "WebSphere MQ Message Header Mappings" on page 21-9.
- 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`, and 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 (MGW_BASIC_MSG_T.TEXT_BODY.small_value and 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 (MGW_BASIC_MSG_T.RAW_BODY.small_value and MGW_BASIC_MSG_T.RAW_BODY.large_value not empty), then message conversion fails.

WebSphere MQ Inbound Propagation

If no message transformation is provided for inbound propagation, then the Oracle Streams AQ 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 MGW_BASIC_MSG_T, but the destination ADT can be any ADT supported by Oracle Streams AQ.

If the Oracle Streams AQ 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 Streams AQ queue payload or transformation source ADT is MGW_BASIC_MSG_T, then the WebSphere MQ message is mapped to a MGW_BASIC_MSG_T message as follows:

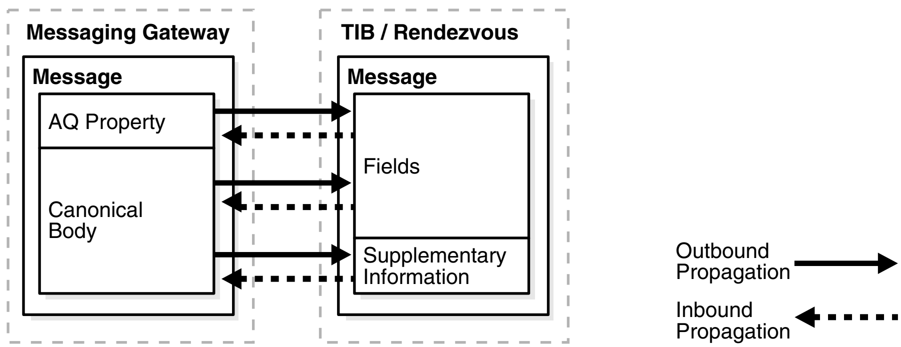
- Specific WebSphere MQ header fields are mapped to Oracle Streams AQ message properties as previously described.
- The MGW_BASIC_MSG_T.header attribute of the canonical message is set to {name,value} pairs based on the WebSphere MQ header fields, as described in Table 21–1. These values preserve the 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 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 MGW_BASIC_MSG_T.raw_body.

See Also: "WebSphere MQ Message Header Mappings" on page 21-9

Message Conversion for TIB/Rendezvous

MGW regards a TIB/Rendezvous message as a set of fields and supplementary information. Figure 21–4 shows how messages are converted between MGW and TIB/Rendezvous.

Figure 21–4 Message Conversion for 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 21–3 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 21–3 TIB/Rendezvous Datatype Mapping

TIB/Rendezvous Wire Format	Oracle Type	ADT Field Type
Bool	NUMBER	TIBRVMSG_BOOL
F32	NUMBER	TIBRVMSG_F32
F64	NUMBER	TIBRVMSG_F64
I8	NUMBER	TIBRVMSG_I8
I16	NUMBER	TIBRVMSG_I16
I32	NUMBER	TIBRVMSG_I32
I64	NUMBER	TIBRVMSG_I64
U8	not supported	not supported
U16	not supported	not supported
U32	not supported	not supported
U64	not supported	not supported
IPADDR32	VARCHAR2	TIBRVMSG_IPADDR32
IPPORT16	NUMBER	TIBRVMSG_IPPORT16
DATETIME	DATE	TIBRVMSG_DATETIME
F32ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_F32ARRAY
F64ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_ F64ARRAY
I8ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_ I8ARRAY
I16ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_I16ARRAY
I32ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_ I32ARRAY
I64ARRAY	SYS.MGW_NUMBER_ARRAY_T	TIBRVMSG_ I64ARRAY
U8ARRAY	not supported	not supported
U16ARRAY	not supported	not supported
U32ARRAY	not supported	not supported
U64ARRAY	not supported	not supported
MSG	not supported	not supported
OPAQUE	RAW or BLOB	TIBRVMSG_OPAQUE

Table 21–3 (Cont.) TIB/Rendezvous Datatype Mapping

TIB/Rendezvous Wire Format	Oracle Type	ADT Field Type
STRING	VARCHAR2 or CLOB	TIBRVMSG_STRING
XML	RAW or BLOB	TIBRVMSG_XML

For propagation between Oracle Streams AQ and TIB/Rendezvous, MGW provides direct support for the Oracle Streams AQ payload types RAW and SYS.MGW_TIBRV_MSG_T. To support any other Oracle Streams AQ payload type, you must supply a transformation.

TIB/Rendezvous Outbound Propagation

If no propagation transformation is provided for outbound propagation, then the Oracle Streams AQ 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 Streams AQ.

If the Oracle Streams AQ 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 subscriber option AQ_MsgProperties is specified with a value of TRUE, then the MGW agent generates a field for each Oracle Streams AQ message property in the TIB/Rendezvous message. Table 21–4 shows the field name strings and the corresponding values used in the TIB/Rendezvous message.

Table 21–4 TIB/Rendezvous and MGW Names for Oracle Streams AQ Message Properties

Oracle Streams AQ Message Property	MGW Name	TIB/Rendezvous Wire Format Datatype	Used For
priority	MGW_AQ_priority	TibrvMsg.I32	Enqueue, Dequeue
expiration	MGW_AQ_expiration	TibrvMsg.I32	Enqueue, Dequeue
delay	MGW_AQ_delay	TibrvMsg.I32	Enqueue, Dequeue
correlation	MGW_AQ_correlation	TibrvMsg.STRING	Enqueue, Dequeue

Table 21–4 (Cont.) TIB/Rendezvous and MGW Names for Oracle Streams AQ Message Properties

Oracle Streams AQ Message Property	MGW Name	TIB/Rendezvous Wire Format Datatype	Used For
`exception_queue`	`MGW_AQ_exception_queue`	`TibrvMsg.STRING`	Enqueue, Dequeue
`enqueue_time`	`MGW_AQ_enqueue_time`	`TibrvMsg.DATETIME`	Dequeue
`original_msgid`	`MGW_AQ_original_msgid`	`TibrvMsg.OPAQUE`	Dequeue

If the Oracle Streams AQ 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.

TIB/Rendezvous Inbound Propagation

If no propagation transformation is provided for inbound propagation, then the Oracle Streams AQ 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 Streams AQ, but the source ADT of the transformation must be `SYS.MGW_TIBRV_MSG_T`.

If the Oracle Streams AQ 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 Streams AQ queue payload is RAW, then:

- The Oracle Streams AQ 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 subscriber 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 21–4 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 Streams AQ 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 Streams AQ 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 *PL/SQL Packages and Types Reference* for the value datatypes

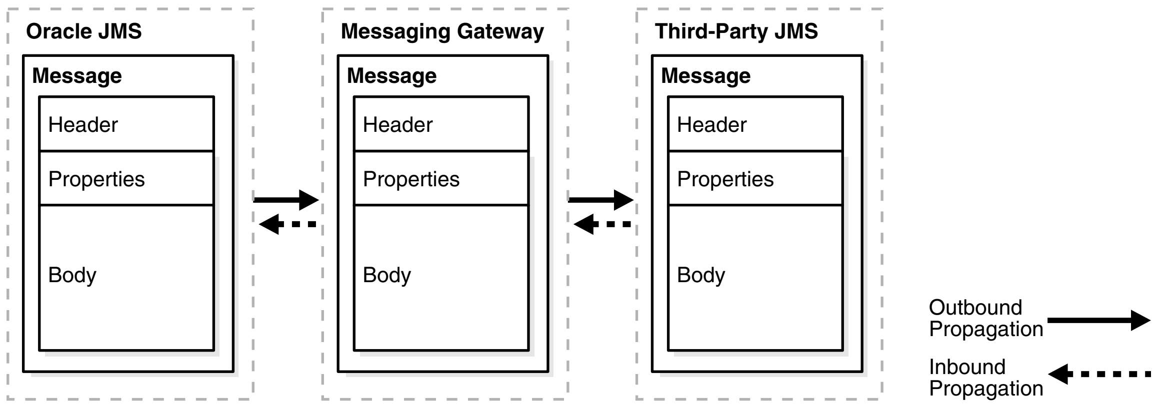
JMS Messages

MGW propagates only JMS messages between Oracle JMS and non-Oracle JMS systems, without changing the message content. Figure 21–5 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 21–5 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 21–5 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 21–5 Oracle JMS Message Conversion

JMS Message	ADT
`javax.jms.TextMessage`	`SYS.AQ$_JMS_TEXT_MESSAGE`
`javax.jms.BytesMessage`	`SYS.AQ$_JMS_BYTES_MESSAGE`
`javax.jms.MapMessage`	`SYS.AQ$_JMS_MAP_MESSAGE`
`javax.jms.StreamMessage`	`SYS.AQ$_JMS_STREAM_MESSAGE`
`javax.jms.ObjectMessage`	`SYS.AQ$_JMS_OBJECT_MESSAGE`
`javax.jms.Message`	`SYS.AQ$_JMS_MESSAGE`

When a propagation job is activated, the MGW agent checks the Oracle Streams AQ payload type for the propagation source or destination. If the type is one of those listed in Table 21–5 or `SYS.ANYDATA`, then message propagation is attempted. Otherwise an exception is logged and propagation is not attempted.

JMS Outbound Propagation

When dequeuing a message from an Oracle Streams AQ queue, Oracle JMS converts instances of the ADTs shown in Table 21–5 into JMS messages. In addition it can convert instances of `SYS.ANYDATA` into JMS messages, depending on the content.

A queue with payload type `SYS.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` ADT and a queue payload type `SYS.AQ$_JMS_BYTES_MESSAGE`.

22

Monitoring Oracle Messaging Gateway

This chapter discusses means of monitoring the Oracle Messaging Gateway (MGW) agent, abnormal situations you may experience, several sources of information about MGW errors and exceptions, and suggested remedies.

This chapter contains these topics:

- The Oracle Messaging Gateway Log File
- Monitoring the Oracle Messaging Gateway Agent Status
- Monitoring Oracle Messaging Gateway Propagation
- Oracle Messaging Gateway Agent Error Messages

The Oracle Messaging Gateway Log File

MGW agent status, history, and errors are recorded in the MGW log file. A different log file is created each time the MGW 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 is:

```
oramgw-hostname-timestamp-processid.log
```

By default the MGW log file is in *ORACLE_HOME*/mgw/log. This location can be overridden by parameter log_directory in 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 MGW agent starting. The sample log file shows that a link, a registered foreign **queue**, a **subscriber**, and a schedule have been added. The log shows that the subscriber has been activated. The last line indicates that the MGW agent is up and running.

Example 22–1   Sample MGW Log File

```
>>2003-07-22 15:04:49  MGW  C-Bootstrap  0 LOG  process-id=11080
Bootstrap program starting
>>2003-07-22 15:04:50  MGW  C-Bootstrap  0 LOG  process-id=11080
JVM created -- heapsize = 64
>>2003-07-22 15:04:53  MGW  Engine  0  200  main
MGW Agent version: 10.1.0.2
>>2003-07-22 15:04:53  MGW  AdminMgr  0  LOG  main
Connecting to database using connect string = jdbc:oracle:oci8:@INST1
>>2003-07-22 15:05:00  MGW  Engine  0  200  main
MGW Component version: 10.1.0.2.0
>>2003-07-22 15:05:01  MGW  Engine  0  200  main
MGW job number: 125, MGW job sid: 10, MGW database instance: 1
>>2003-07-22 15:05:09  MGW  Engine  0  1  main
Agent is initializing.
>>2003-07-22 15:05:09  MGW  Engine  0  23  main
The number of worker threads is set to 1.
>>2003-07-22 15:05:09  MGW  Engine  0  22  main
The default polling interval is set to 5000ms.
>>2003-07-22 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          :
  connections   : 1
  inbound logQ  : logq1
  outbound logQ : logq2
>>2003-07-22 15:05:09  MGW  Engine  0  4  main
Link MQLINK has been added.
>>2003-07-22 15:05:09  MGW  Engine  0  7  main
Queue DESTQ@MQLINK has been registered; provider queue: MGWUSER.MYQUEUE.
>>2003-07-22 15:05:09  MGW  Engine  0  9  main
Propagation Schedule SCH_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been
added.
>>2003-07-22 15:05:09  MGW  AQN  0  LOG  main
Creating AQ messaging link:
  link            : oracleMgwAq
  link type       : native
  database        : INST1
  user            : MGWAGENT
  connection type : JDBC OCI
  connections     : 1
  inbound logQ    : sys.mgw_recv_log
  outbound logQ   : sys.mgw_send_log
>>2003-07-22 15:05:10  MGW  Engine  0  19  main
MGW subscriber SUB_AQ2MQ has been activated.
>>2003-07-22 15:05:10  MGW  Engine  0  14  main
MGW subscriber SUB_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been
added.
>>2003-07-22 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 MGW 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 an MGW log file when an invalid value ('bad_service_name') was specified for the database parameter of DBMS_MGWADM.DB_CONNECT_INFO. This resulted in the MGW agent being unable to establish database connections.

Example 22–2 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
oracle.mgw.admin.MgwAdminException: [241]  Failed to connect to database. SQL
error: 12154, connect string: jdbc:oracle:oci8:@BAD_SERVICE_NAME
[ …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:

- The MGW_GATEWAY view
- Oracle Messaging Gateway Irrecoverable Error Messages
- Other Oracle Messaging Gateway Error Conditions

The MGW_GATEWAY view

The MGW_GATEWAY view monitors the progress of the MGW agent. Among the fields that can be used to monitor the agent are:

- AGENT_STATUS
- AGENT_PING
- LAST_ERROR_MSG
- AGENT_JOB
- AGENT_INSTANCE

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, MGW has detected another agent already running. This situation should never occur under normal usage.

Querying the `AGENT_PING` field pings the MGW 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 MGW agent occurs. `AGENT_INSTANCE` indicates the Oracle Database instance on which the MGW instance was started.

> **See Also:** "DBMS_MGWADM" in *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 MGW agent is not running. If the `AGENT_STATUS` is `NOT_STARTED` and the `LAST_ERROR_MSG` field is not NULL, then the MGW agent has encountered an irrecoverable error while starting or running. Check if an MGW log file has been generated and whether it indicates any errors. If a log file is not present, then the MGW 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 MGW agent will not attempt to restart itself when one of these errors occurs.

ORA-01089: Immediate shutdown in progress - no operations are permitted

The MGW 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 MGW 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 MGW 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 MGW 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.

ORA-28576: Lost RPC connection to external procedure agent

The MGW 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 MGW log file to determine if further information is available. If the problem persists, then contact Oracle Support Services for assistance.

ORA-32830: Result code -1 returned by MGW agent

An error occurred starting the **Java Virtual Machine** (JVM). Your response depends on the contents of the MGW log file:

```
Cannot create Java VM
```

If the MGW log file contains this line, then 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`

```
Could not find class oracle.mgw.engine.Agent
```

If the MGW log file contains this line, then verify that the CLASSPATH set in `mgw.ora` contains `mgw.jar`. For example:

```
ORACLE_HOME/mgw/classes/mgw.jar
```

ORA-32830: Result code -2 returned by MGW agent

An error occurred reading the initialization file `mgw.ora`. Verify that the file is readable.

ORA-32830: Result code -3 returned by MGW agent

An error occurred creating the MGW log file. Verify that the log directory can be written to. The default location is *`ORACLE_HOME`*`/mgw/log`.

ORA-32830: Result code -100 returned by MGW agent

The MGW 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 MGW agent

An irrecoverable error caused the MGW agent to shut down. Check the MGW log file for further information. Verify that the values specified in `mgw.ora` are correct.

Incorrect values can cause the MGW agent to terminate due to unusual error conditions.

ORA-32830: Result code -102 returned by MGW agent

The MGW agent shut down because the version of file *ORACLE_HOME*/mgw/classes/mgw.jar does not match the version of the MGW PL/SQL packages. Verify that all MGW components are from the same release.

ORA-32830: Result code -103 returned by MGW agent

The MGW 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 MGW agent

See previous error.

ORA-32830: Result code -105 returned by MGW agent

The MGW 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 MGW 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 MGW agent using `DBMS_MGWADM.STARTUP`.

ORA-32830: Result code -106 returned by MGW agent

See previous error.

> **See Also:** "DBMS-MGWADM" in *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.

Too Few Job Queue Processes

MGW uses job queues in Oracle Database to start the MGW agent process. When AGENT_STATUS is START_SCHEDULED, the MGW agent job is waiting to be run by the job scheduler. At least one job queue process must be configured to execute queued jobs in the background. The MGW job is scheduled to execute immediately, but will not do so until a job queue process is available. The MGW holds its job queue process for the lifetime of that MGW agent session.

If the MGW status remains START_SCHEDULED for an extended period of time, then it can indicate that the database instance has been started with no or too few job queue processes. Verify that the database instance has been started with enough job queue processes so one is available for use by MGW. You can set the number of job queue processes with init.ora parameter JOB_QUEUE_PROCESSES, or you can change the number dynamically with:

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = number;
```

Oracle recommends a minimum of two job queue processes for MGW in addition to those used for other purposes.

Multiple Errors While Starting

Another possibility is that the job queue has attempted to start the MGW agent sixteen times, each time resulting in an error. To determine if this is the case, connect as user SYS and execute the following query:

```
select g.agent_job, j.failures, j.broken from MGW_GATEWAY g, DBA_JOBS j where
j.job = g.agent_job;
```

If the job has failed sixteen times, check the last error message from the MGW_GATEWAY view and any error messages in the MGW log file, fix the problem, call DBMS_MGWADM.SHUTDOWN to remove the current job queue job, and then call DBMS_MGWADM.STARTUP to try again.

RAC Environment

If MGW is being used in a RAC environment and the agent has been configured to run on a particular instance that is currently not running, then AGENT_STATUS will remain START_SCHEDULED until that instance is running.

Monitoring Oracle Messaging Gateway Propagation

MGW **propagation** can be monitored using the MGW_SUBSCRIBERS view and the MGW 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_SUBSCRIBERS view also has dynamic information that can be used to monitor **message** propagation. Applicable fields include STATUS, PROPAGATED_MSGS, EXCEPTIONQ_MSGS, FAILURES, LAST_ERROR_MSG, LAST_ERROR_DATE, and LAST_ERROR_TIME.

STATUS can be either ENABLED or DELETE_PENDING. DELETE_PENDING means subscriber removal is pending, usually because DBMS_MGWADM.REMOVE_SUBSCRIBER has been called but certain cleanup tasks pertaining to this subscriber are still outstanding. Otherwise the subscriber is considered ENABLED.

The PROPAGATED_MSGS field of the MGW_SUBSCRIBERS view indicates how many messages have been successfully propagated. This field is reset to zero when the MGW agent is started.

If an MGW subscriber has been configured with an **exception queue**, then the MGW agent will move messages to that exception queue the first time the MGW agent encounters a propagation failure caused by a message conversion failure. A message conversion failure is indicated by oracle.mgw.common.MessageException in the MGW 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 MGW agent is started.

If an error occurs during message propagation for a subscriber, 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, MGW halts propagation attempts for this subscriber. To resume propagation attempts you must call DBMS_MGWADM.RESET_SUBSCRIBER for the subscriber in question.

If an error occurs, then examine the MGW log file for further information.

See Also: "DBMS_MGWADM" in *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_SUBSCRIBERS` view and logged to the MGW 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 MGW agent log file:

- `oracle.mgw.common.MessageException`

  This error type is logged when a message conversion failure occurs. The MGW 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 Streams AQ 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 entries in `DBMS_MGWADM.DB_CONNECT_INFO`. Either the MGW agent user or password has not been entered correctly, or the `database` parameter is incorrect or NULL.

If the `database` parameter is NULL, then check the MGW 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 MGW 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 DBMS_MGWADM.DB_CONNECT_INFO is called with a NULL value for the database parameter. If a local connection is desired, the correct ORACLE_SID value must be set in the MGW agent process. This can be done by adding the following line to mgw.ora:

```
set ORACLE_SID = sid_value
```

ORACLE_SID need not be set if DBMS_MGWADM.DB_CONNECT_INFO is called with a not NULL value for the database parameter. In this case the value should specify a net service name from tnsnames.ora.

If setting ORACLE_SID in mgw.ora does not work, then the database parameter of DBMS_MGWADM.DB_CONNECT_INFO must be set to a value that is not NULL.

[415] Missing messages from source queue of subscriber <*subscriber_id*>

Possible causes include:

- The agent partially processed persistent messages that were dequeued by someone other than the MGW agent.
- The propagation source queue was purged or re-created.
- A message was moved to the Oracle Streams AQ exception queue.

If this error occurs, then call procedure CLEANUP_GATEWAY in the DBMS_MGWADM package:

```
DBMS_MGWADM.CLEANUP_GATEWAY (
       action => DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE,
       sarg => <subscriber_id>);
```

The call takes effect only if the subscriber has encountered the missing message problem and the agent is running. The agent treats the missing messages as **nonpersistent** messages and continues processing the subscriber.

> **See Also:** "Propagation Subscriber Overview" on page 20-15 for more information on MGW exception queues

[416] Missing log records in receiving log queue for subscriber <*subscriber_id*>

Possible causes include:

- Log records were dequeued from the log queues by someone other than the MGW agent.
- The log queues were purged or re-created.

If this error occurs, then call procedure `CLEANUP_GATEWAY` in the `DBMS_MGWADM` package:

```
DBMS_MGWADM.CLEANUP_GATEWAY (
       action => DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC,
       sarg => <subscriber_id>);
```

The call takes effect only if the subscriber 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 subscriber <*subscriber_id*>

See previous error.

[421] WARNING: Unable to get connections to recover subscriber <*subscriber_id*>

This message is a warning message indicating that the MGW 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 MGW link associated with the subscriber.

See Also: "Altering a Messaging System Link" on page 20-10

[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 Streams AQ queue. The linked exceptions should give more information.

Possible causes include:

- The foreign queue was registered incorrectly, or the MGW 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 MGW link.

- The Oracle Streams AQ queue does not exist. Perhaps the queue was removed after the MGW subscriber was created.

  Check that the Oracle Streams AQ queue still exists.

[436] LOW MEMORY WARNING: total memory = < >, free_mem = < >

The MGW 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 MGW agent is running, then it must be restarted for this change to take effect.

[703] Failed to retrieve information for transformation <*transformation_id*>

The MGW agent could not obtain all the information it needs about the **transformation**. The transformation parameter of `DBMS_MGWADM.ADD_SUBSCRIBER` 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:

  ```
  SELECT TRANSFORMATION_ID, OWNER FROM DBA_TRANSFORMATIONS;
  ```

- The wrong transformation is registered with MGW. Verify that the transformation registered is the one intended.
- The MGW 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 22–3 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 22–3 No EXECUTE Privilege on Object Type

```
Errors occurred during processing of subscriber SUB_AQ2MQ_2
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…]
```

[720] AQ payload type <*type*> not supported; queue: <*queue*>

The payload type of the Oracle Streams AQ queue used by an MGW subscriber is not directly supported by MGW. For non-JMS propagation, MGW directly supports the payload types `RAW`, `SYS.MGW_BASIC_MSG_T` and `SYS.MGW_TIBRV_MSG_T`.

Possible actions include:

- Configure the MGW subscriber to use a transformation that converts the queue payload type to a supported type.
- Remove the MGW subscriber and create a new subscriber that uses an Oracle Streams AQ queue with a supported payload type.

  For **Java Message Service** (JMS) propagation, the MGW subscriber must be removed and a new subscriber added whose Oracle Streams AQ payload type is supported by **Oracle Java Message Service** (OJMS). Transformations are not supported for JMS propagation.

[721] Transformation type <*type*> not supported; queue: <*queue_name*>, transform: <*transformation*>

An MGW subscriber was configured with a transformation that uses an object type that is not one of the MGW **canonical** types.

For an outbound subscriber, the transformation `from_type` must be the Oracle Streams AQ payload type, and the `to_type` must be an MGW canonical type. For an inbound subscriber, the transformation `from_type` must be an MGW canonical type and the `to_type` must be the Oracle Streams AQ 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 Streams AQ 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 MGW agent user does not have `EXECUTE` privilege on the transformation function. This is illustrated in Example 22–4.

  It is not sufficient to grant `EXECUTE` to `MGW_AGENT_ROLE` and then grant `MGW_AGENT_ROLE` to the MGW agent user. You must grant `EXECUTE` privilege on the transformation function directly to the MGW agent user or to `PUBLIC`.

Example 22–4 No EXECUTE Privilege on Transformation Function

```
Errors occurred during processing of subscriber SUB_MQ2AQ_2
oracle.mgw.common.GatewayException: [722] Message transformation failed queue:
MGWUSER.DESTQ_SIMPLEADT, transform: MGWUSER.MGW_BASIC_MSG_TO_SIMPLEADT
[…Java stack trace here…]
[Linked-exception]
oracle.mgw.common.MessageException: [722] Message transformation failed;
queue: MGWUSER.DESTQ_SIMPLEADT, transform:
MGWUSER.MGW_BASIC_MSG_TO_SIMPLEADT
[…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 MGW 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 MGW agent user. You must grant `EXECUTE` privilege on the object type directly to the MGW 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*>

An MGW subscriber is configured for inbound propagation where the canonical message type generated by the non-Oracle messaging system link is not compatible with the Oracle Streams AQ queue payload type. For example, propagation from a TIB/Rendezvous messaging system to an Oracle Streams AQ queue with a `SYS.MGW_BASIC_MSG_T` payload type, or propagation from WebSphere MQ to an Oracle Streams AQ queue with a `SYS.MGW_TIBRV_MSG_T` payload type.

Possible actions include:

- Configure the MGW subscriber with a transformation that maps the canonical message type generated by the non-Oracle messaging link to the Oracle Streams AQ payload type.
- Remove the MGW subscriber and create a new subscriber whose Oracle Streams AQ queue payload type matches the canonical message type generated by the non-Oracle link.

[725] Text message not supported for RAW payload

An MGW subscriber is configured for inbound propagation to an Oracle Streams AQ 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 MGW subscriber and create a new subscriber to an Oracle Streams AQ destination whose payload type supports text data.

[726] Message size <*size*> too large for RAW payload; maximum size is <*size*>

An MGW subscriber is configured for inbound propagation to an Oracle Streams AQ 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 MGW subscriber and create a new subscriber to an Oracle Streams AQ 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 a `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.

Part VIII

Using Oracle Streams with Oracle Streams AQ

Part VIII describes how to use Oracle Streams with Oracle Streams Advanced Queuing (AQ).

This part contains the following chapters:

- Chapter 23, "Staging and Propagating with Oracle Streams AQ"
- Chapter 24, "Oracle Streams Messaging Example"

23

Staging and Propagating with Oracle Streams AQ

This chapter describes how to use and manage Oracle Streams AQ when staging and propagating. It describes `SYS.AnyData` queues and user messages.

This chapter contains these topics:

- Oracle Streams Event Staging and Propagation Overview
- SYS.AnyData Queues and User Messages
- Message Propagation and SYS.AnyData Queues
- Managing an Oracle Streams Messaging Environment
- Wrapping User Message Payloads in a SYS.AnyData Wrapper
- Propagating Messages Between a SYS.AnyData Queue and a Typed Queue

Oracle Streams Event Staging and Propagation Overview

Oracle Streams uses queues of type `SYS.AnyData` to stage events. There are two types of events that can be staged in an Oracle Streams **queue**:

- Logical change records (LCRs). LCRs are objects that contain information about a change to a database object.
- User messages. These are custom messages created by users or applications.

Both types of events are of type `SYS.AnyData` and can be used for information sharing within a single database or between databases.

Staged events can be consumed or propagated, or both. These events can be consumed by an apply process or by a user application that explicitly dequeues them. Even after an event is consumed, it can remain in the queue if you have also configured Oracle Streams to propagate the event to one or more other queues or if **message** retention is specified. These other queues can reside in the same database or in different databases. In either case, the queue from which the events are propagated is called the source queue, and the queue that receives the events is called the destination queue.

SYS.AnyData Queues and User Messages

Oracle Streams enables messaging with queues of type `SYS.AnyData`. `SYS.AnyData` queues can stage user messages whose payloads are of `SYS.AnyData` type. A `SYS.AnyData` payload can be a wrapper for payloads of different datatypes. Queues that can stage messages of only a particular type are called typed queues.

By using `SYS.AnyData` wrappers for message payloads, publishing applications can **enqueue** messages of different types into a single queue. Subscribing applications can then **dequeue** these messages, either explicitly using a dequeue **API** or implicitly using an apply process. If the subscribing application is remote, then the messages can be propagated to the remote site, and the subscribing application can dequeue the messages from a local queue in the remote database. Alternatively, a remote subscribing application can dequeue messages directly from the source queue using a variety of standard protocols, such as PL/SQL and **Oracle Call Interface** (OCI).

Oracle Streams interoperates with Oracle Streams AQ, which supports all the standard features of message queuing systems, including multiconsumer queues, publish and subscribe, content-based routing, internet **propagation**, transformations, and gateways to other messaging subsystems.

See Also: *Oracle Streams Concepts and Administration*

SYS.AnyData Wrapper for User Messages Payloads

You can wrap almost any type of payload in a `SYS.AnyData` payload. To do this, you use the `Convert`*`data_type`* static functions of the `SYS.AnyData` type, where *`data_type`* is the type of object to wrap. These functions take the object as input and return a `SYS.AnyData` object.

The following datatypes cannot be wrapped in a `SYS.AnyData` wrapper:

- Nested table
- **NCLOB**
- ROWID and UROWID

The following datatypes can be directly wrapped in a `SYS.AnyData` wrapper, but these datatypes cannot be present in a user-defined type payload wrapped in a `SYS.AnyData` wrapper:

- **CLOB**
- **BLOB**
- **BFILE**
- **VARRAY**

See Also: *PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

Programmatic Environments for Enqueue and Dequeue of User Messages

Your applications can use the following programmatic environments to enqueue user messages into a `SYS.AnyData` queue and dequeue user messages from a `SYS.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 a `SYS.AnyData` queue.

See Also: Chapter 4, "Oracle Streams AQ: Programmatic Environments" for more information about these programmatic interfaces

Enqueuing User Messages Using PL/SQL

To enqueue a user message containing an LCR into a `SYS.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 `SYS.AnyData.ConvertObject` function to convert the LCR into `SYS.AnyData` payload and enqueue it using the `DBMS_AQ.ENQUEUE` procedure.

To enqueue a user message containing a non-LCR object into a `SYS.AnyData` queue using PL/SQL, you use one of the `SYS.AnyData.Convert*` functions to convert the object into `SYS.AnyData` payload and enqueue it using the `DBMS_AQ.ENQUEUE` procedure.

See Also:

- *Oracle Streams Concepts and Administration*, "Managing a Streams Messaging Environment"
- Chapter 24, "Oracle Streams Messaging Example"

Enqueuing User Messages Using OCI or JMS

To enqueue a user message containing an LCR into a `SYS.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 `catxlcr.sql` script in Oracle home in the `rdbms/admin/` directory.

To enqueue a message using OCI, perform the same actions that you would to enqueue a message into a typed queue. A typed queue is a queue that can stage messages of a particular type only. To enqueue a message using JMS, a user must have `EXECUTE` privilege on `DBMS_AQ`, `DBMS_AQIN`, and `DBMS_AQJMS` packages.

Note: Enqueue of JMS types and XML types does not work with Oracle Streams `Sys.Anydata` queues unless you call `DBMS_AQADM.ENABLE_JMS_TYPES(`*`queue_table_name`*`)` after `DBMS_STREAMS_ADM.SET_UP_QUEUE()`. Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

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 a `SYS.AnyData` queue, you can use methods `QueueSender.send` or `TopicPublisher.publish`.

See Also:

- *Oracle Streams Concepts and Administration*, "Enqueue and Dequeue Events Using JMS"
- *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 `SYS.AnyData` queue using PL/SQL, you use the `DBMS_AQ.DEQUEUE` procedure and specify `SYS.AnyData` as the payload. The user message can contain an LCR or another type of object.

Dequeuing User Messages Using OCI or JMS

In a `SYS.AnyData` queue, user messages containing LCRs in XML format are represented as `oracle.xdb.XMLType`. Non-LCR messages can be one of the following formats:

- A JMS type (`javax.jms.TextMessage`, `javax.jms.MapMessage`, `javax.jms.StreamMessage`, `javax.jms.ObjectMessage`, or `javax.jms.BytesMessage`)

- A user-defined type

To dequeue a message from a `SYS.AnyData` queue using JMS, you can use methods `QueueReceiver`, `TopicSubscriber`, or `TopicReceiver`. Because the queue can contain different types of objects wrapped in a `SYS.AnyData` wrapper, you must register a list of SQL types and their corresponding Java classes in the typemap of the **JMS session**. JMS types are already preregistered in the typemap.

For example, suppose a queue contains LCR messages represented as `oracle.xdb.XMLType` and 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.util.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 message selectors with a `QueueReceiver` or `TopicPublisher`, the selector can contain any SQL92 expression that has a combination of one or more of the following:

- **JMS message** header fields or properties, including `JMSPriority`, `JMSCorrelationID`, `JMSType`, `JMSXUserI`, `JMSXAppID`, `JMSXGroupID`, and `JMSXGroupSeq`. The following is an example of a JMS message field:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- User-defined message properties, as in the following example:

  ```
  color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
  ```

- PL/SQL functions, as in the following example:

  ```
  hr.GET_TYPE(tab.user_data) = 'HR.EMPLOYEES'
  ```

To dequeue a message using OCI, perform the same actions that you would to dequeue a message from a typed queue.

See Also:

- *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 `OCIAQdeq` function in the *Oracle Call Interface Programmer's Guide* for more information about dequeuing messages using OCI

Message Propagation and SYS.AnyData Queues

`SYS.AnyData` queues can interoperate with typed queues in an Oracle Streams environment. A typed queue can stage messages of a particular type only. Table 23–1 shows the types of propagation possible between queues.

Table 23–1 Propagation Between Different Types of Queues

Source Queue	Destination Queue	Transformation
`SYS.AnyData`	`SYS.AnyData`	None
Typed	`SYS.AnyData`	Implicit **Note:** Propagation is possible only if the messages in the typed queue meet the restrictions outlined in "User-Defined Type Messages" on page 23-8.
`SYS.AnyData`	Typed	Requires a rule to filter messages and a user-defined **transformation**
Typed	Typed	Follows Oracle Streams AQ rules

To propagate messages containing a payload of a certain type from a `SYS.AnyData` source queue to a typed destination queue, you must perform a transformation. Only messages containing a payload of the same type as the typed queue can be propagated to the typed queue.

Although you cannot use **Simple Object Access Protocol** (SOAP) to interact directly with a `SYS.AnyData` queue, you can use SOAP with Oracle Streams by propagating messages between a `SYS.AnyData` queue and a typed queue. If you want to enqueue a message into a `SYS.AnyData` queue using SOAP, then you can configure propagation from a typed queue to `SYS.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 `SYS.AnyData` queue.

If you want to use SOAP to dequeue a message that is in a `SYS.AnyData` queue, then you can configure propagation from a `SYS.AnyData` queue to a typed queue. The message is propagated automatically from the `SYS.AnyData` queue to the typed queue. Then, the message would be available for access using SOAP.

Note: Certain Oracle Streams capabilities, such as capturing changes using a capture process and applying changes with an apply process, can be configured only with `SYS.AnyData` queues.

See Also: *Oracle Streams Concepts and Administration*, "Propagating Messages Between a SYS.AnyData Queue and a Typed Queue"

User-Defined Type Messages

If you plan to enqueue, propagate, or dequeue user-defined type messages in an Oracle Streams environment, then each type used in these messages must exist at every database where the message can be staged 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:

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

The object identifier need not match at each database.

See Also:

- "SYS.AnyData Wrapper for User Messages Payloads" on page 23-3 for information about wrapping user-defined type message payloads in `SYS.AnyData` messages
- *Oracle Streams Concepts and Administration* for more information about directed networks

Managing an Oracle Streams Messaging Environment

Oracle Streams enables messaging with queues of type `SYS.AnyData`. These queues stage user messages whose payloads are of `SYS.AnyData` type, and a `SYS.AnyData` payload can be a wrapper for payloads of different datatypes.

This section provides instructions for completing the following tasks:

- SYS.AnyData Wrapper for User Messages Payloads
- Propagating Messages Between a SYS.AnyData Queue and a Typed Queue

---

Note: The examples in this section assume that you have configured an Oracle Streams administrator at each database.

See Also: *PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

Wrapping User Message Payloads in a SYS.AnyData Wrapper

You can wrap almost any type of payload in a `SYS.AnyData` payload. The following sections provide examples of enqueuing messages into, and dequeuing messages from, a `SYS.AnyData` queue.

Example 23–1 Example of Wrapping a Payload in a SYS.AnyData Payload and Enqueuing It

The following steps illustrate how to wrap payloads of various types in a `SYS.AnyData` payload.

1. Connect as an administrative user who can create users, grant privileges, create tablespaces, and alter users at the `dbs1.net` database.

2. Grant EXECUTE privilege on the DBMS_AQ package to the oe user so that this user can run the ENQUEUE and DEQUEUE procedures in that package:

```
GRANT EXECUTE ON DBMS_AQ TO oe;
```

3. Connect as the Oracle Streams administrator, as in the following example:

```
CONNECT strmadmin/strmadminpw@dbs1.net
```

4. Create a SYS.AnyData queue if one does not already exist.

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'oe_q_table_any',
    queue_name   => 'oe_q_any',
    queue_user   => 'oe');
END;
/
```

   The oe user is configured automatically as a secure user of the oe_q_any queue and is given ENQUEUE and DEQUEUE privileges on the queue.

5. Add a **subscriber** to the oe_q_any queue. This subscriber performs explicit dequeues of events. The ADD_SUBSCRIBER procedure will automatically create an AQ_AGENT.

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name  =>  'strmadmin.oe_q_any',
    subscriber  =>  subscriber);
END;
/
```

6. Grant the oe user enqueue and dequeue privileges on queue strmadmin.oe_q_any.

```
BEGIN
  DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
    privilege  =>  ALL,
    queue_name => 'strmadmin.oe_q_any',
    grantee    => 'oe');
END;
/
```

7. Associate the `oe` user with the `local_agent` agent:

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'local_agent',
    db_username => 'oe');
END;
/
```

8. Connect as the `oe` user.

```
CONNECT oe/oe@dbs1.net
```

9. Create a procedure that takes as an input parameter an object of `SYS.AnyData` type and enqueues a message containing the payload into an existing `SYS.AnyData` queue.

```
CREATE OR REPLACE PROCEDURE oe.enq_proc (payload SYS.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_q_any',
    enqueue_options     =>  enqopt,
    message_properties  =>  mprop,
    payload             =>  payload,
    msgid               =>  enq_msgid);
END;
/
```

10. Run the procedure you created in Step 9 by specifying the appropriate `Convert`*`data_type`* function. The following commands enqueue messages of various types.

`VARCHAR2` type:

```
EXEC oe.enq_proc(SYS.AnyData.ConvertVarchar2('Chemicals - SW'));
COMMIT;
```

`NUMBER` type:

```
EXEC oe.enq_proc(SYS.AnyData.ConvertNumber('16'));
COMMIT;
```

User-defined type:

```
BEGIN
  oe.enq_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1646 Brazil Blvd','361168','Chennai','Tam', 'IN')));
END;
/
COMMIT;
```

> **See Also:** *Oracle Streams Concepts and Administration*, "Viewing the Contents of User-Enqueued Events in a Queue" for information about viewing the contents of these enqueued messages

Example 23–2 Example of Dequeuing a Payload That Is Wrapped in a SYS.AnyData Payload

The following steps illustrate how to dequeue a payload wrapped in a `SYS.AnyData` payload. This example assumes that you have completed the steps in "Example of Wrapping a Payload in a SYS.AnyData Payload and Enqueuing It" on page 23-9.

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*, where *queue_table_name* is the name of the **queue table**. For example, to find the consumers of the messages in the `oe_q_any` queue, run the following query:

```
CONNECT strmadmin/strmadminpw@dbs1.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ANY;
```

1. Connect as the `oe` user:

   ```
   CONNECT oe/oe@dbs1.net
   ```

2. Create a procedure that takes as an input the consumer of the messages you want to dequeue. The following example procedure dequeues messages of `oe.cust_address_typ` and prints the contents of the messages.

   ```
   CREATE OR REPLACE PROCEDURE oe.get_cust_address (
   consumer IN VARCHAR2) AS
     address         OE.CUST_ADDRESS_TYP;
     deq_address     SYS.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_q_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;
/
```

3. Run the procedure you created in Step 1 and specify the consumer of the messages you want to dequeue, as in the following example:

```
SET SERVEROUTPUT ON SIZE 100000
EXEC oe.get_cust_address('LOCAL_AGENT');
```

Propagating Messages Between a SYS.AnyData Queue and a Typed Queue

`SYS.AnyData` queues can interoperate with typed queues in an Oracle Streams environment. A typed queue is a queue that can stage messages of a particular type only. To propagate a message from a `SYS.AnyData` queue to a typed queue, the message must be transformed to match the type of the typed queue. The following sections provide examples of propagating non-LCR user messages and LCRs between a `SYS.AnyData` queue and a typed queue.

Note: The examples in this section assume that you have completed the examples in "SYS.AnyData Wrapper for User Messages Payloads" on page 23-3.

See Also: "Message Propagation and SYS.AnyData Queues" on page 23-7 for more information about propagation between `SYS.AnyData` and typed queues

Example 23–3 Example of Propagating Non-LCR User Messages to a Typed Queue

The following steps set up propagation from a `SYS.AnyData` queue named `oe_q_any` to a typed queue of type `oe.cust_address_typ` named `oe_q_address`. The source queue `oe_q_any` is at the `dbs1.net` database, and the destination queue `oe_q_address` is at the `dbs2.net` database. Both queues are owned by `strmadmin`.

1. Connect as an administrative user who can grant privileges at `dbs1.net`.
2. Grant the following privilege to `strmadmin`, if it was not already granted.

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. Grant `strmadmin` `EXECUTE` privilege on `oe.cust_address_typ` at `dbs1.net` and `dbs2.net`.

```
CONNECT oe/oe@dbs1.net
```

```
GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;

CONNECT oe/oe@dbs2.net

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
```

4. Create a typed queue at `dbs2.net`, if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs2.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'strmadmin.oe_q_table_address',
    queue_payload_type  => 'oe.cust_address_typ',
    multiple_consumers  => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name   => 'strmadmin.oe_q_address',
    queue_table  => 'strmadmin.oe_q_table_address');
  DBMS_AQADM.START_QUEUE(
    queue_name   => 'strmadmin.oe_q_address');
END;
/
```

5. Create a database link between `dbs1.net` and `dbs2.net` if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS2.NET';
```

6. Create a function called `any_to_cust_address_typ` in the `strmadmin` schema at `dbs1.net` that takes a `SYS.AnyData` payload containing a `oe.cust_address_typ` object and returns the `oe.cust_address_typ` object.

```
CREATE OR REPLACE FUNCTION strmadmin.any_to_cust_address_typ(
  in_any IN SYS.AnyData)
RETURN OE.CUST_ADDRESS_TYP
AS
  address       OE.CUST_ADDRESS_TYP;
  num_var       NUMBER;
  type_name     VARCHAR2(100);
BEGIN
  -- Get the type of object
  type_name := in_any.GetTypeName();
```

```
  -- Check if the object type is OE.CUST_ADDRESS_TYP
  IF (type_name = 'OE.CUST_ADDRESS_TYP') THEN
    -- Put the address in the message into the address variable
    num_var := in_any.GetObject(address);
    RETURN address;
  ELSE
    raise_application_error(-20101, 'Conversion failed - ' || type_name);
  END IF;
END;
/
```

7. Create a transformation at `dbs1.net` using the `DBMS_TRANSFORM` package.

```
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;
/
```

8. Create a subscriber for the typed queue if one does not already exist. The subscriber must contain a rule that ensures that only messages of the appropriate type are propagated to the destination queue.

```
DECLARE
  subscriber  SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT ('ADDRESS_AGENT_REMOTE',
                               'STRMADMIN.OE_Q_ADDRESS@DBS2.NET',
                               0);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name     => 'strmadmin.oe_q_any',
    subscriber     => subscriber,
    rule           =>
              'TAB.USER_DATA.GetTypeName()=''OE.CUST_ADDRESS_TYP''',
    transformation => 'strmadmin.anytoaddress');
END;
/
```

9. Schedule propagation between the `SYS.AnyData` queue at `dbs1.net` and the typed queue at `dbs2.net`.

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
    queue_name   => 'strmadmin.oe_q_any',
    destination  => 'dbs2.net');
END;
/
```

10. Enqueue a message of `oe.cust_address_typ` type wrapped in a `SYS.AnyData` wrapper:

```
CONNECT oe/oe@dbs1.net

BEGIN
  oe.enq_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1668 Chong Tao','111181','Beijing',NULL, 'CN')));
END;
/
COMMIT;
```

11. After allowing some time for propagation, query the queue table at `dbs2.net` to view the propagated message:

```
CONNECT strmadmin/strmadminpw@dbs2.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ADDRESS;
```

See Also: Chapter 21, "Oracle Messaging Gateway Message Conversion" for more information about transformations during propagation

Example 23–4 Example of Propagating LCRs to a Typed Queue

To propagate LCRs from a `SYS.AnyData` queue to a typed queue, you complete the same steps as you do for non-LCR events, but Oracle supplies the transformation functions. You can use the following functions in the `DBMS_STREAMS` package to transform LCRs in `SYS.AnyData` queues to messages in typed queues:

- The `CONVERT_ANYDATA_TO_LCR_ROW` function transforms `SYS.AnyData` payload containing a row LCR into `SYS.LCR$_ROW_RECORD` payload.

- The CONVERT_ANYDATA_TO_LCR_DDL function transforms SYS.AnyData payload containing a DDL LCR into SYS.LCR$_DDL_RECORD payload.

You can propagate user-enqueued LCRs to an appropriate typed queue, but propagation of captured LCRs to a typed queue is not supported.

The following example sets up propagation of row LCRs from a SYS.AnyData queue named oe_q_any to a typed queue of type SYS.LCR$_ROW_RECORD named oe_q_lcr. The source queue oe_q_any is at the dbs1.net database, and the destination queue oe_q_lcr is at the dbs3.net database.

1. Connect as an administrative user who can grant privileges at dbs1.net.
2. Grant the following privilege to strmadmin, if it was not already granted.

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. Create a queue of the LCR type if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs3.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table         => 'strmadmin.oe_q_table_lcr',
    queue_payload_type  => 'SYS.LCR$_ROW_RECORD',
    multiple_consumers  => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name   => 'strmadmin.oe_q_lcr',
    queue_table  => 'strmadmin.oe_q_table_lcr');
  DBMS_AQADM.START_QUEUE(
    queue_name   => 'strmadmin.oe_q_lcr');
END;
/
```

4. Create a database link between dbs1.net and dbs3.net if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS3.NET';
```

5. Create a transformation at dbs1.net using the DBMS_TRANSFORM package.

```
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;
/
```

6. Create a subscriber at the typed queue if one does not already exist. The subscriber specifies the `CONVERT_ANYDATA_TO_LCR_ROW` function for the transformation parameter.

```
DECLARE
  subscriber  SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT (
    'ROW_LCR_AGENT_REMOTE',
    'STRMADMIN.OE_Q_LCR@DBS3.NET',
    0);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name     => 'strmadmin.oe_q_any',
    subscriber     => subscriber,
    rule           => 'TAB.USER_DATA.GetTypeName()=''SYS.LCR$_ROW_RECORD''',
    transformation => 'strmadmin.anytolcr');
END;
/
```

7. Schedule propagation between the `SYS.AnyData` queue at `dbs1.net` and the LCR queue at `dbs3.net`.

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
    queue_name   => 'strmadmin.oe_q_any',
    destination  => 'dbs3.net');
END;
/
```

8. Create a procedure to construct and enqueue a row LCR into the `strmadmin.oe_q_any` queue:

```
CONNECT oe/oe@dbs1.net

CREATE OR REPLACE 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);
  -- Construct the LCR based on information passed to procedure
  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);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name         =>  'strmadmin.oe_q_any',
    enqueue_options    =>  eopt,
    message_properties =>  mprop,
    payload            =>  SYS.AnyData.ConvertObject(row_lcr),
    msgid              =>  enq_msgid);
END enq_row_lcr_proc;
/
```

9. Create a row LCR that inserts a row into the `oe.inventories` table and enqueue the row LCR into the `strmadmin.oe_q_any` queue.

```
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$_ROW_UNIT(
    'PRODUCT_ID',
    SYS.AnyData.ConvertNumber(3503),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
```

```
    'WAREHOUSE_ID',
    SYS.AnyData.ConvertNumber(1),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit3 := SYS.LCR$_ROW_UNIT(
    'QUANTITY_ON_HAND',
    SYS.AnyData.ConvertNumber(157),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3);
oe.enq_row_lcr_proc(
  source_dbname  =>  'DBS1.NET',
  cmd_type       =>  'INSERT',
  obj_owner      =>  'OE',
  obj_name       =>  'INVENTORIES',
  old_vals       =>  NULL,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

10. After allowing some time for propagation, query the queue table at `dbs3.net` to view the propagated message:

```
CONNECT strmadmin/strmadminpw@dbs3.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_LCR;
```

> **See Also:** "`DBMS_STREAMS`" in *PL/SQL Packages and Types Reference* for more information about the row LCR and DDL LCR conversion functions

24

Oracle Streams Messaging Example

This chapter illustrates a messaging environment that can be constructed using Oracle Streams.

This chapter contains these topics:

- Overview of Messaging Example
- Prerequisites
- Set Up Users and Create a SYS.AnyData Queue
- Create the Enqueue Procedures
- Configure an Apply Process
- Configure Explicit Dequeue
- Enqueue Events
- Dequeue Events Explicitly and Query for Applied Events
- Enqueue and Dequeue Events Using JMS

> **See Also:** *Oracle Streams Concepts and Administration* for more information about messaging and `SYS.AnyData` queues

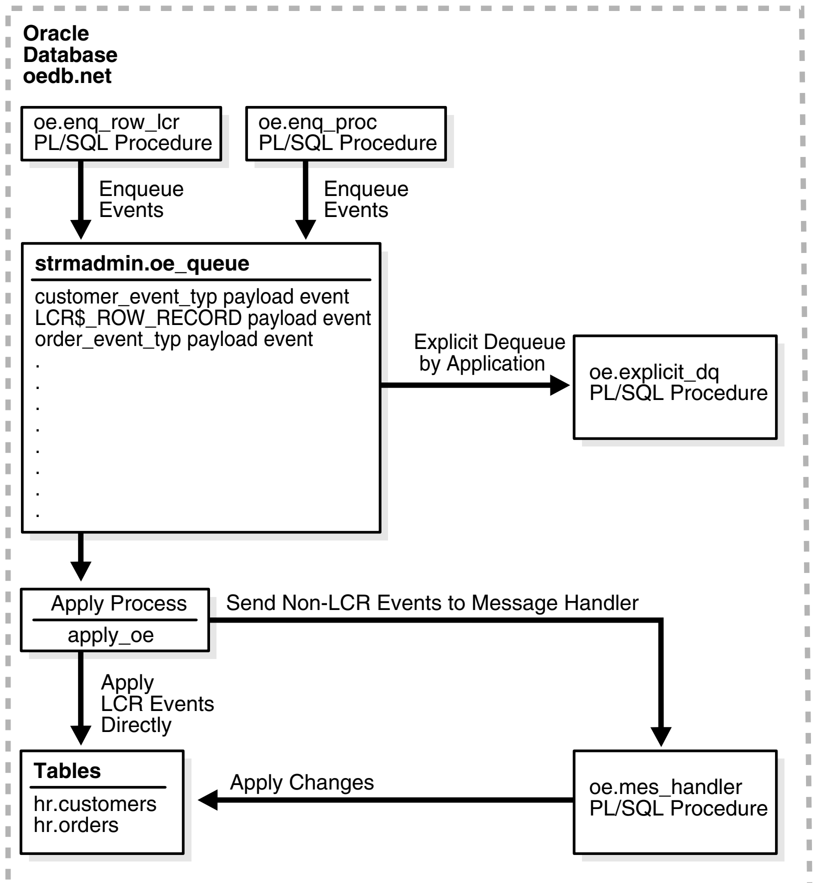
Overview of Messaging Example

This example illustrates using a single `SYS.AnyData` **queue** at a database called `oedb.net` to create a Oracle Streams messaging environment in which events containing **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 and customer payload as `SYS.Anydata` events into the queue
- Enqueuing messages containing row LCR payload as `SYS.Anydata` events into the queue
- Creating a rule set for applying the events
- Creating an evaluation context used by the rule set
- Creating a 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 24–1 provides an overview of this environment.

Figure 24–1 Example Oracle Streams Messaging Environment



Prerequisites

The following tasks must be completed before you begin the example in this section.

- Set initialization parameter `COMPATIBLE` to `9.2.0` or higher.
- Configure your network and Oracle Net so that you can access the `oedb.net` database from the client where you run these scripts.

See Also: *Oracle Net Services Administrator's Guide*

- This example creates a new user to function as the Oracle Streams administrator (`strmadmin`) and prompts you for the tablespace you want to use for this user's data. Before you start this example, either create a new tablespace or identify an existing tablespace for the Oracle Streams administrator to use. The Oracle Streams administrator should not use the `SYSTEM` tablespace.

Set Up Users and Create a SYS.AnyData Queue

Complete the following steps to set up users and create a `SYS.AnyData` queue for a Oracle Streams messaging environment.

1. Show Output and Spool Results
2. Set Up Users
3. Create the SYS.AnyData Queue
4. Grant the oe User Privileges on the Queue
5. Create an Agent for Explicit Enqueue
6. Associate the oe User with the explicit_enq Agent
7. Check the Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-8 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```
/************************* BEGINNING OF SCRIPT ******************************
```

Step 1 Show Output and Spool Results

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```
*/

SET ECHO ON
SPOOL streams_setup_message.out

/*
```

Step 2 Set Up Users

Connect to `oedb.net` as `SYS` user.

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@oedb.net AS SYSDBA

/*
```

This example uses the `oe` sample **schema**. For this example to work properly, the `oe` user must have privileges to run the subprograms in the `DBMS_AQ` package. The `oe` user is specified as the queue user when the `SYS.AnyData` queue is created in Step 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 events into and dequeue events from the queue.

Also, most of the configuration and administration actions illustrated in this example are performed by the Oracle Streams administrator. In this step, create the Oracle Streams administrator named `strmadmin` and grant this user 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. You can choose a different name for this user.

Note:

- To ensure security, use a password other than `strmadminpw` for the Oracle Streams administrator.
- 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.
- The `ACCEPT` command must appear on a single line in the script.

```
*/
```

```
GRANT EXECUTE ON DBMS_AQ TO oe;

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter the tablespace for the Oracle Streams
administrator: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
                     QUOTA UNLIMITED ON &streams_tbs;

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);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

/*
```

Step 3 Create the SYS.AnyData Queue

Connect as the Oracle Streams administrator.

```
*/

CONNECT strmadmin/strmadminpw@oedb.net

/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `oe_queue` at `oedb.net`. This queue functions as the `SYS.AnyData` queue by holding events used in the messaging environment.

Running the `SET_UP_QUEUE` procedure performs the following actions:

- Creates a **queue table** named `oe_queue_table`. This queue table is owned by the Oracle Streams administrator (`strmadmin`) and uses the default storage of this user.
- Creates a queue named `oe_queue` owned by the Oracle Streams administrator (`strmadmin`)
- Starts the queue

```
*/

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'oe_queue_table',
    queue_name  => 'oe_queue');
END;
/

/*
```

Step 4 Grant the oe User Privileges on the Queue

```
*/

BEGIN
  SYS.DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
    privilege  => 'ALL',
    queue_name => 'strmadmin.oe_queue',
    grantee    => 'oe');
END;
/
```

```
/*
```

Step 5 Create an Agent for Explicit Enqueue

Create an agent that will be used to perform explicit enqueue operations on the `oe_queue` queue.

```
*/

BEGIN
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name  => 'explicit_enq');
END;
/

/*
```

Step 6 Associate the oe User with the explicit_enq Agent

For a user to perform queue operations, such as enqueue and dequeue, on a secure queue, the user must be configured as a secure queue user of the queue. The `oe_queue` queue is a secure queue because it was created using `SET_UP_QUEUE`. This step enables the `oe` user to perform enqueue operations on this queue.

```
*/

BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_enq',
    db_username => 'oe');
END;
/

/*
```

Step 7 Check the Spool Results

Check the `streams_setup_message.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/
```

Create the Enqueue Procedures

Complete the following steps to create one PL/SQL procedure that enqueues non-LCR events into the `SYS.AnyData` queue and one PL/SQL procedure that enqueues row LCR events into the `SYS.AnyData` queue.

1. Show Output and Spool Results
2. Create a Type to Represent Orders
3. Create a Type to Represent Customers
4. Create the Procedure to Enqueue Non-LCR Events
5. Create a Procedure to Construct and Enqueue Row LCR Events
6. Check the Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-13 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```
/************************* BEGINNING OF SCRIPT ******************************
```

Step 1 Show Output and Spool Results

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```
*/

SET ECHO ON
SPOOL streams_enqprocs_message.out

/*
```

Step 2 Create a Type to Represent Orders

Connect as `oe`.

```
*/

CONNECT oe/oe@oedb.net
```

```
/*
```

Create a type to represent orders based on the columns in the `oe.orders` table. 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 correct action to perform on the instance (either apply process dequeue or explicit dequeue). This type is used for events that are enqueued into the `SYS.AnyData` queue.

```
*/

CREATE OR REPLACE 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),
  promotion_id    NUMBER(6),
  action          VARCHAR(7));
/

/*
```

Step 3 Create a Type to Represent Customers

Create a type to represent customers based on the columns in the `oe.customers` table. 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 correct action to perform on the instance (either apply process dequeue or explicit dequeue). This type is used for events that are enqueued into the `SYS.AnyData` queue.

```
*/

CREATE OR REPLACE 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),
  cust_geo_location   MDSYS.SDO_GEOMETRY,
  action              VARCHAR(7));
/

/*
```

Step 4 Create the Procedure to Enqueue Non-LCR Events

Create a PL/SQL procedure called `enq_proc` to enqueue events into the `SYS.AnyData` queue.

> **Note:** A single enqueued message can be dequeued by an apply process and by an explicit dequeue, but this example does not illustrate this capability.

```
*/

CREATE OR REPLACE PROCEDURE oe.enq_proc (event IN SYS.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;
/

/*
```

Step 5 Create a Procedure to Construct and Enqueue Row LCR Events

Create a procedure called `enq_row_lcr` that constructs a row LCR and then enqueues the row LCR into the queue.

See Also: *PL/SQL Packages and Types Reference* for more information about LCR constructors

```
*/

CREATE OR REPLACE 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);
  -- Construct the LCR based on information passed to procedure
  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);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name         =>  'strmadmin.oe_queue',
    enqueue_options    =>  eopt,
    message_properties =>  mprop,
    payload            =>  SYS.AnyData.ConvertObject(row_lcr),
    msgid              =>  enq_msgid);
END enq_row_lcr;
/

/*
```

Step 6 Check the Spool Results

Check the `streams_enqprocs_message.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/
```

```
SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/
```

Configure an Apply Process

Complete the following steps to configure an apply process to apply the user-enqueued events in the `SYS.AnyData` queue.

1. Show Output and Spool Results
2. Create a Function to Determine the Value of the action Attribute
3. Create a Message Handler
4. Grant strmadmin EXECUTE Privilege on the Procedures
5. Create the Evaluation Context for the Rule Set
6. Create a Rule Set for the Apply Process
7. Create a Rule that Evaluates to TRUE if the Event Action Is apply
8. Create a Rule that Evaluates to TRUE for the Row LCR Events
9. Add the Rules to the Rule Set
10. Create an Apply Process
11. Grant EXECUTE Privilege on the Rule Set To oe User
12. Start the Apply Process
13. Check the Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-19 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```
/************************* BEGINNING OF SCRIPT ******************************
```

Step 1 Show Output and Spool Results

Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

```
*/

SET ECHO ON
SPOOL streams_apply_message.out

/*
```

Step 2 Create a Function to Determine the Value of the action Attribute

Connect as oe.

```
*/

CONNECT oe/oe@oedb.net

/*
```

Create a function called get_oe_action to determine the value of the action attribute in the events in the queue. This function is used in rules later in this example 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.

```
*/

CREATE OR REPLACE FUNCTION oe.get_oe_action (event IN SYS.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;
/

/*
```

Step 3 Create a Message Handler

Create a message handler called `mes_handler` that will be used as a message handler by the apply process. This procedure takes the payload in a user-enqueued event of type `oe.order_event_typ` or `oe.customer_event_typ` and inserts it as a row in the `oe.orders` table and `oe.customers` table, respectively.

```
*/

CREATE OR REPLACE PROCEDURE oe.mes_handler (event SYS.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.cust_geo_location);
  END IF;
END;
/

/*
```

Step 4 Grant strmadmin EXECUTE Privilege on the Procedures

```
*/

GRANT EXECUTE ON get_oe_action TO strmadmin;
```

```
GRANT EXECUTE ON mes_handler TO strmadmin;

/*
```

Step 5 Create the Evaluation Context for the Rule Set

Connect as the Oracle Streams administrator.

```
*/

CONNECT strmadmin/strmadminpw@oedb.net

/*
```

Create the evaluation context for the rule set. The table alias is `tab` in this example, but you can use a different table alias name if you wish.

```
*/

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;
/

/*
```

Step 6 Create a Rule Set for the Apply Process

Create the rule set for the apply process.

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       =>  'apply_oe_rs',
    evaluation_context  =>  'strmadmin.oe_eval_context');
END;
/
```

```
/*
```

Step 7 Create a Rule that Evaluates to TRUE if the Event Action Is apply

Create a rule that evaluates to `TRUE` if the `action` value of an `event` 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 Step 5 on page 24-16.

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name   => 'strmadmin.apply_action',
    condition   => ' oe.get_oe_action(tab.user_data) = ''APPLY'' ');
END;
/

/*
```

Step 8 Create a Rule that Evaluates to TRUE for the Row LCR Events

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, this evaluation context is used for the rule during evaluation instead of the rule set's evaluation context.

```
*/

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;
/

/*
```

Step 9 Add the Rules to the Rule Set

Add the rules created in Step 7 and Step 8 to the rule set created in Step 6 on page 24-16.

```
*/

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;
/

/*
```

Step 10 Create an Apply Process

Create an apply process that is associated with the `oe_queue`, that uses the `apply_oe_rs` rule set, and that uses the `mes_handler` procedure as a message handler.

```
*/

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;
/

/*
```

Step 11 Grant EXECUTE Privilege on the Rule Set To oe User

Grant `EXECUTE` privilege on the `strmadmin.apply_oe_rs` rule set. Because `oe` was specified as the apply user when the apply process was created in Step 10, `oe` needs `EXECUTE` privilege on the rule set used by the apply process.

```
*/
```

```
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;
/

/*
```

Step 12 Start the Apply Process

Set the `disable_on_error` parameter to `n` so that the apply process is not disabled if it encounters an error, and start the apply process at `oedb.net`.

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_oe',
    parameter   => 'disable_on_error',
    value       => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name  =>  'apply_oe');
END;
/

/*
```

Step 13 Check the Spool Results

Check the `streams_apply_message.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/
```

Configure Explicit Dequeue

Complete the following steps to configure explicit dequeue of messages based on message contents.

1. Show Output and Spool Results
2. Create an Agent for Explicit Dequeue
3. Associate the oe User with the explicit_dq Agent
4. Add a Subscriber to the oe_queue Queue
5. Create a Procedure to Dequeue Events Explicitly
6. Check Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-24 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```
/************************ BEGINNING OF SCRIPT ******************************
```

Step 1 Show Output and Spool Results

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```
*/

SET ECHO ON
SPOOL streams_explicit_dq.out

/*
```

Step 2 Create an Agent for Explicit Dequeue

Connect as the Oracle Streams administrator.

```
*/

CONNECT strmadmin/strmadminpw@oedb.net
```

```
/*
```

Create an agent that will be used to perform explicit dequeue operations on the oe_queue queue.

```
*/

BEGIN
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name  => 'explicit_dq');
END;
/

/*
```

Step 3 Associate the oe User with the explicit_dq Agent

For a user to perform queue operations, such as enqueue and dequeue, on a secure queue, the user must be configured as a secure queue user of the queue. The oe_queue queue is a secure queue because it was created using SET_UP_QUEUE. The oe user is able to perform dequeue operations on this queue when the agent is used to create a **subscriber** to the queue in the next step.

```
*/

BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_dq',
    db_username => 'oe');
END;
/

/*
```

Step 4 Add a Subscriber to the oe_queue Queue

Add a subscriber to the oe_queue queue. This subscriber will perform explicit dequeues of events. A subscriber rule is used to dequeue any events where the action value is not apply. If the action value is apply for an event, then the event is ignored by the subscriber. Such events are dequeued and processed by the apply process.

```
*/

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;
/

/*
```

Step 5 Create a Procedure to Dequeue Events Explicitly

Connect as `oe`.

```
*/

CONNECT oe/oe@oedb.net

/*
```

Create a PL/SQL procedure called `explicit_dq` to dequeue events explicitly using the subscriber created in Step 4 on page 24-21.

Note:

- This procedure commits after the dequeue of the events. The commit informs the queue that the dequeued messages have been consumed successfully by this subscriber.
- This procedure can process multiple transactions and uses two exception handlers. The first exception handler `next_trans` moves to the next transaction while the second exception handler `no_messages` exits the loop when there are no more messages.

```
*/

CREATE OR REPLACE PROCEDURE oe.explicit_dq (consumer IN VARCHAR2) AS
  deqopt       DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
  msgid        RAW(16);
  payload      SYS.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);
  pragma 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('Event 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);
    END IF;
    EXCEPTION
      WHEN next_trans THEN
      deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
      WHEN no_messages THEN
        new_messages  := FALSE;
        DBMS_OUTPUT.PUT_LINE('No more events');
     END;
  END LOOP;
END;
/

/*
```

Step 6 Check Spool Results

Check the `streams_explicit_dq.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/
```

Enqueue Events

Complete the following steps to enqueue non-LCR events and row LCR events into the queue.

1. Show Output and Spool Results
2. Enqueue Non-LCR Events to be Dequeued by the Apply Process

3. Enqueue Non-LCR Events to be Dequeued Explicitly
4. Enqueue Row LCR Events to be Dequeued by the Apply Process
5. Check Spool Results

Note:

- It is possible to dequeue user-enqueued LCRs explicitly, but this example does not illustrate this capability.
- If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-30 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```
/************************* BEGINNING OF SCRIPT ******************************
```

Step 1 Show Output and Spool Results

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```
*/

SET ECHO ON
SPOOL streams_enq_deq.out

/*
```

Step 2 Enqueue Non-LCR Events to be Dequeued by the Apply Process

Connect as `oe`.

```
*/

CONNECT oe/oe@oedb.net

/*
```

Enqueue events with `apply` for the `action` value. Based on the apply process rules, the apply process dequeues and processes these events with the `oe.mes_handler` message handler procedure created in "Create a Message Handler" on page 24-15. 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.

```
*/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(
    2500,'05-MAY-01','online',117,3,44699,161,NULL,'APPLY')));
END;
/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.customer_event_typ(
    990,'Hester','Prynne',oe.cust_address_typ('555 Beacon Street','Boston',
    'MA',02109,'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,'APPLY')));
END;
/

COMMIT;

/*
```

Step 3 Enqueue Non-LCR Events to be Dequeued Explicitly

Enqueue events with `dequeue` for the `action` value. The `oe.explicit_dq` procedure created in "Create a Procedure to Dequeue Events Explicitly" on page 24-22 dequeues these events because the `action` is not `apply`. Based on the apply process rules, the apply process ignores these events. The `COMMIT` after the enqueues makes these two enqueues part of the same transaction.

```
*/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(
    2501,'22-JAN-00','direct',117,3,22788,161,NULL,'DEQUEUE')));
END;
/

BEGIN
  oe.enq_proc(SYS.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',
    '+1 718 511 9114', '+1 718 888 4832'),'i','AMERICA',3000,
```

```
    'nick@great_gatsby.com',149,NULL,'DEQUEUE')));
END;
/

COMMIT;

/*
```

Step 4 Enqueue Row LCR Events to be Dequeued by the Apply Process

Enqueue row LCR events. The apply process applies these events directly. Enqueued LCRs should commit at transaction boundaries. In this step, 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.

Create a row LCR that inserts a row into the `oe.orders` table.

```
*/

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(
    'ORDER_ID',
    SYS.AnyData.ConvertNumber(2502),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'ORDER_DATE',
    SYS.AnyData.ConvertTimestampLTZ('04-NOV-00'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit3 := SYS.LCR$_ROW_UNIT(
    'ORDER_MODE',
    SYS.AnyData.ConvertVarchar2('online'),
```

```
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newunit4 := SYS.LCR$_ROW_UNIT(
      'CUSTOMER_ID',
      SYS.AnyData.ConvertNumber(145),
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newunit5 := SYS.LCR$_ROW_UNIT(
      'ORDER_STATUS',
      SYS.AnyData.ConvertNumber(3),
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newunit6 := SYS.LCR$_ROW_UNIT(
      'ORDER_TOTAL',
      SYS.AnyData.ConvertNumber(35199),
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newunit7 := SYS.LCR$_ROW_UNIT(
      'SALES_REP_ID',
      SYS.AnyData.ConvertNumber(160),
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newunit8 := SYS.LCR$_ROW_UNIT(
      'PROMOTION_ID',
      SYS.AnyData.ConvertNumber(1),
      DBMS_LCR.NOT_A_LOB,
      NULL,
      NULL);
    newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4,
                                 newunit5,newunit6,newunit7,newunit8);
  oe.enq_row_lcr(
    source_dbname  =>  'OEDB.NET',
    cmd_type       =>  'INSERT',
    obj_owner      =>  'OE',
    obj_name       =>  'ORDERS',
    old_vals       =>  NULL,
    new_vals       =>  newvals);
  END;
  /
  COMMIT;
```

```
/*
```

Create a row LCR that updates the row inserted into the `oe.orders` table previously.

```
*/

DECLARE
  oldunit1  SYS.LCR$_ROW_UNIT;
  oldunit2  SYS.LCR$_ROW_UNIT;
  oldvals   SYS.LCR$_ROW_LIST;
  newunit1  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_ID',
    SYS.AnyData.ConvertNumber(2502),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(35199),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  newunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(5235),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
oe.enq_row_lcr(
  source_dbname  =>  'OEDB.NET',
  cmd_type       =>  'UPDATE',
  obj_owner      =>  'OE',
  obj_name       =>  'ORDERS',
  old_vals       =>  oldvals,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

```
/*
```

Step 5 Check Spool Results

Check the `streams_enq_deq.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/
```

Dequeue Events Explicitly and Query for Applied Events

Complete the following steps to dequeue the events explicitly and query the events that were applied by the apply process. These events were enqueued in the "Enqueue Events" on page 24-24.

Step 1 Run the Procedure to Dequeue Events Explicitly

Run the procedure you created in "Create a Procedure to Dequeue Events Explicitly" on page 24-22 and specify the **consumer** of the events you want to dequeue. In this case, the consumer is the subscriber you added in "Add a Subscriber to the oe_queue Queue" on page 24-21. In this example, events that are not dequeued explicitly by this procedure are dequeued by the apply process.

```
CONNECT oe/oe@oedb.net

SET SERVEROUTPUT ON SIZE 100000

EXEC oe.explicit_dq('explicit_dq');
```

You should see the non-LCR events that were enqueued in "Enqueue Non-LCR Events to be Dequeued Explicitly" on page 24-26.

Step 2 Query for Applied Events

Query the `oe.orders` and `oe.customers` table to see the rows corresponding to the events applied by the apply process:

```
SELECT * FROM oe.orders WHERE order_id = 2500;

SELECT cust_first_name, cust_last_name, cust_email
```

```
  FROM oe.customers WHERE customer_id = 990;

SELECT * FROM oe.orders WHERE order_id = 2502;
```

You should see the non-LCR event that was enqueued in "Enqueue Non-LCR Events to be Dequeued by the Apply Process" on page 24-25 and the row LCR events that were enqueued in "Enqueue Row LCR Events to be Dequeued by the Apply Process" on page 24-27.

Enqueue and Dequeue Events Using JMS

This example enqueues non-LCR events and row LCR events into the queue using **Java Message Service** (JMS). Then, this example dequeues these events from the queue using JMS.

Note: Enqueue of JMS types and XML types does not work with Oracle Streams `Sys.Anydata` queues unless you call `DBMS_AQADM.ENABLE_JMS_TYPES(`*`queue_table_name`*`)` after `DBMS_STREAMS_ADM.SET_UP_QUEUE()`. Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

Complete the following steps:

1. Run the catxlcr.sql Script
2. Create the Types for User Events
3. Set the CLASSPATH
4. Create Java Classes that Map to the Oracle Object Types
5. Create a Java Code for Enqueuing Messages
6. Create a Java Code for Dequeuing Messages
7. Compile the Scripts
8. Run the Enqueue Program
9. Run the Dequeue Program

Step 1 Run the catxlcr.sql Script

For this example to complete successfully, the LCR schema must be loaded into the SYS schema using the catxlcr.sql script in Oracle home in the rdbms/admin/ directory. Run this script now if it has not been run already.

Note: To run catxlcr.sql, you must either have created the database using **Database Configuration Assistant** or separately installed **Java Virtual Machine**, XDB, and XML Schema.

For example, if your Oracle home directory is /usr/oracle, then enter the following to run the script:

```
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA

@/usr/oracle/rdbms/admin/catxlcr.sql
```

Step 2 Create the Types for User Events

```
CONNECT oe/oe

CREATE TYPE address AS OBJECT (street VARCHAR (30), num NUMBER)
/

CREATE TYPE person AS OBJECT (name VARCHAR (30), home ADDRESS)
/
```

Step 3 Set the CLASSPATH

The following jar and zip files should be in the CLASSPATH based on the release of JDK you are using.

Also, make sure LD_LIBRARY_PATH (Solaris operating system) or PATH (Windows) has $ORACLE_HOME/lib set.

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/rdbms/jlib/aqapi14.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/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/rdbms/jlib/aqapi12.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
```

Step 4 Create Java Classes that Map to the Oracle Object Types

First, create a file `input.typ` with the following lines:

```
SQL PERSON AS JPerson
SQL ADDRESS AS JAddress
```

Then, run Jpublisher.

```
jpub -input=input.typ -user=OE/OE
```

Completing these actions generates two Java classes named `JPerson` and `JAddress` for the `person` and `address` types, respectively.

Step 5 Create a Java Code for Enqueuing Messages

This program uses the Oracle JMS **API** to publish messages into a Oracle Streams topic.

This program does the following:

- Publishes a non-LCR based **ADT** message to the topic
- Publishes a JMS text message to a topic
- Publish an LCR based message to the topic

```
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
       {
         /* Create the TopicConnectionFactory
          * Only the JDBC OCI driver can be used to access Oracle Streams through JMS
          */
         tc_fact = AQjmsFactory.getTopicConnectionFactory(
                      args[1], args[0], Integer.parseInt(args[2]), "oci8");

         t_conn = tc_fact.createTopicConnection( "OE","OE");

         /* Create a TopicSession */
         t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

         /* Start the connection */
         t_conn.start() ;

         /* Publish non-LCR based messages */
         publishUserMessages(t_sess);

         /* Publish LCR based messages */
         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();
      }
   }

   /*
    * publishUserMessages - this method publishes an ADT message and a
    * JMS text message to a Oracle Streams topic
    */
   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
     {
       /* Get the topic */
       topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

       /* Create a publisher */
       t_pub = t_sess.createPublisher(topic);


       /* Agent to access oe_queue */
       agent = new AQjmsAgent("explicit_enq", null);


       /* Create a PERSON adt message */
       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);

       /* Set the payload in the message */
```

```
    adt_msg.setAdtPayload(pers);

    ((AQjmsMessage)adt_msg).setSenderID(agent);

    System.out.println("Publish message 1 -type  PERSON\n");

    /* Create the recipient list */
    recipList = new AQjmsAgent[1];
    recipList[0] = new AQjmsAgent("explicit_dq", null);

    /* Publish the message */
    ((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\n");

    /* Publish the message */
    ((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());
  }
}

/*
 * publishLcrMessages - this method publishes an XML LCR message to a
 * Oracle Streams topic
 */
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
    {
      /* Get the topic */
      topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

      /* Create a publisher */
      t_pub = t_sess.createPublisher(topic);

      /* Get the JDBC connection */
      db_conn = ((AQjmsSession)t_sess).getDBConnection();

      /* Agent to access oe_queue */
      agent = new AQjmsAgent("explicit_enq", null);


      /* Create a adt message */
      adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

    /* Create the LCR representation in XML */
    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
          http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd'");
      lcr_data.append("> \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");
      lcr_data.append("<tag>0ABC</tag> \n");
      lcr_data.append("<transaction_id>0.0.0</transaction_id> \n");
      lcr_data.append("<scn>0</scn> \n");
```

```
lcr_data.append("<old_values> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>Clob old</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data> \n");
lcr_data.append("<date><value>1997-11-24</value><format>SYYYY-MM-DD</format></date> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data> \n");
lcr_data.append("<timestamp><value>1999-05-31T13:20:00.000</value>
    <format>SYYYY-MM-DD'T'HH24:MI:SS.FF</format></timestamp> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C05</column_name> \n");
lcr_data.append("<data><raw>ABCDE</raw></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("</old_values> \n");
lcr_data.append("<new_values> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><number>35.23</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data><number>-100000</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data><varchar>Hel lo</varchar></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
```

```
      lcr_data.append("<column_name>C05</column_name> \n");
      lcr_data.append("<data><char>wor ld</char></data> \n");
      lcr_data.append("</new_value> \n");
      lcr_data.append("</new_values> \n");
      lcr_data.append("</ROW_LCR>");

      /* Create the XMLType containing the LCR */
      xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());

      /* Set the payload in the message */
      adt_msg.setAdtPayload(xml_lcr);

      ((AQjmsMessage)adt_msg).setSenderID(agent);

      System.out.println("Publish message 3 - XMLType containing LCR ROW\n");

      /* Create the recipient list */
      recipList = new AQjmsAgent[1];
      recipList[0] = new AQjmsAgent("explicit_dq", null);

      /* Publish the message */
      ((AQjmsTopicPublisher)t_pub).publish(topic, adt_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());
    }
  }

}
```

Step 6 Create a Java Code for Dequeuing Messages

This program uses Oracle JMS API to receive messages from a Oracle Streams topic.

This program does the following:

- Registers mappings for `person`, `address` and `XMLType` in JMS typemap
- Receives LCR messages from a Oracle Streams topic

- Receives user ADT messages from a Oracle Streams topic

```
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;

     try
     {
       if (args.length < 3 )
         System.out.println("Usage:java filename [SID] [HOST] [PORT]");
       else
       {
         /* Create the TopicConnectionFactory
          * Only the JDBC OCI driver can be used to access Oracle Streams through JMS
          */
         tc_fact = AQjmsFactory.getTopicConnectionFactory(
                      args[1], args[0], Integer.parseInt(args[2]), "oci8");

         t_conn = tc_fact.createTopicConnection( "OE","OE");

         /* Create a TopicSession */
         t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

         /* Start the connection */
         t_conn.start() ;

         receiveMessages(t_sess);

         t_sess.close() ;
         t_conn.close() ;
         System.out.println("\nEnd of StreamsDeq Demo") ;
       }
     }
```

```
    catch (Exception ex)
    {
      System.out.println("Exception-1: " + ex);
      ex.printStackTrace();
    }
  }

  /*
   * receiveMessages -This method receives messages from the SYS.AnyData queue
   */
  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
    {
      /* Get the topic */
      topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

      /* Create a TopicReceiver to receive messages for consumer "jms_recv */
      t_recv = ((AQjmsSession)t_sess).createTopicReceiver(topic,
                                                    "jms_recv", null);

      map = ((AQjmsSession)t_sess).getTypeMap();

      /* Register mappings for ADDRESS and PERSON in the JMS typemap */
      map.put("OE.PERSON", Class.forName("JPerson"));
      map.put("OE.ADDRESS", Class.forName("JAddress"));

      /* Register mapping for XMLType in the TypeMap - required for LCRs */
      map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLTypeFactory"));

      System.out.println("Receive messages ...\n");

      do
      {
```

```
        try
        {
          jms_msg = (t_recv.receive(10));

          i++;

          /* Set navigation mode to NEXT_MESSAGE */
          ((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_MESSAGE);
        }
        catch (JMSException jms_ex2)
        {
          if((jms_ex2.getLinkedException() != null) &&
             (jms_ex2.getLinkedException() instanceof SQLException))
          {
            SQLException sql_ex2 =(SQLException)(jms_ex2.getLinkedException());

            /* End of current transaction group
             * Use NEXT_TRANSACTION navigation mode
             */
            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();
    }
    catch (java.sql.SQLException sql_ex)
    {
      System.out.println("SQL Exception: " + sql_ex);
      sql_ex.printStackTrace();

      t_sess.rollback();
    }
  }
}
```

Step 7 Compile the Scripts

```
javac StreamsEnq.java StreamsDeq.java JPerson.java JAddress.java
```

Step 8 Run the Enqueue Program

```
java StreamsEnq  ORACLE_SID HOST PORT
```

For example, if your Oracle SID is `orc182`, your host is `hq_server`, and your port is `1521`, then enter the following:

```
java StreamsEnq orcl82 hq_server 1521
```

Step 9 Run the Dequeue Program

```
java StreamsDeq  ORACLE_SID HOST PORT
```

For example, if your Oracle SID is `orc182`, your host is `hq_server`, and your port is `1520`, then enter the following:

```
java StreamsDeq orcl82 hq_server 1521
```

Part IX

Troubleshooting Oracle Streams AQ

Part IX describes how to troubleshoot Oracle Streams Advanced Queuing (AQ).

This part contains the following chapter:

- Chapter 25, "Troubleshooting Oracle Streams AQ"

25

Troubleshooting Oracle Streams AQ

This chapter describes how to troubleshoot Oracle Streams Advanced Queuing (AQ).

The chapter contains these topics:

- Debugging Oracle Streams AQ Propagation Problems
- Oracle Streams AQ Error Messages

Debugging Oracle Streams AQ 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 5-14

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 dba_queue_schedules and aq$_schedules. Check that it has a 'jobno' in aq$_schedules, and that there is an entry in job$ with that jobno. Make sure that the status of the schedule is enabled: SCHEDULE_DISABLED must be set to 'N'.

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

   If propagation is not occurring, check the view for any errors (last_error_date, last_error_time, last_error_message). Also check the next_run_date and next_run_time in dba_propagation_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 by doing 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:

   ```
   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$_prop_table_`*`instno`* exists in `dba_queue_tables`. The queue `sys.aq$_prop_notify_queue_`*`instno`* must also exist in `dba_queues` and must be enabled for enqueue and dequeue.

   In case of 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-compatible queues, you can do the following:

   ```
   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**:

   ```
   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';
   ```

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 Streams AQ Error Messages

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 Streams AQ 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.

A

Scripts for Implementing BooksOnLine

This Appendix contains the following scripts:

- tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues, and Subscribers
- tkaqdocd.sql: Examples of Administrative and Operational Interfaces
- tkaqdoce.sql: Operational Examples
- tkaqdocp.sql: Examples of Operational Interfaces
- tkaqdocc.sql: Clean-Up Script

tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues, and Subscribers

```
Rem $Header: tkaqdoca.sql 26-jan-99.17:50:37 aquser1 Exp $
Rem
Rem tkaqdoca.sql
Rem
Rem  Copyright (c) Oracle 1998, 1999. All Rights Reserved.
Rem
Rem    NAME
Rem      tkaqdoca.sql - TKAQ DOCumentation Admin examples file

Rem Set up a queue admin account and individual accounts for each application
Rem
connect system/manager
set serveroutput on;
set echo on;

Rem Create a common admin account for all BooksOnLine applications
Rem
CREATE USER BOLADM IDENTIFIED BY BOLADM;
GRANT CONNECT, RESOURCE, aq_administrator_role to BOLADM;
GRANT EXECUTE ON DBMS_AQ TO BOLADM;
GRANT EXECUTE ON DBMS_AQADM TO BOLADM;
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','BOLADM',FALSE);
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('DEQUEUE_ANY','BOLADM',FALSE);

Rem Create the application schemas and grant appropriate permission
Rem to all schemas

Rem Create an account for Order Entry
CREATE USER OE IDENTIFIED BY OE;
GRANT CONNECT, RESOURCE to OE;
GRANT EXECUTE ON DBMS_AQ TO OE;
GRANT EXECUTE ON DBMS_AQADM TO OE;

Rem Create an account for WR Shipping
CREATE USER WS IDENTIFIED BY WS;
GRANT CONNECT, RESOURCE to WS;
GRANT EXECUTE ON DBMS_AQ TO WS;
GRANT EXECUTE ON DBMS_AQADM TO WS;

Rem Create an account for ER Shipping
CREATE USER ES IDENTIFIED BY ES;
GRANT CONNECT, RESOURCE to ES;
```

```
GRANT EXECUTE ON DBMS_AQ TO ES;
GRANT EXECUTE ON DBMS_AQADM TO ES;

Rem Create an account for Overseas Shipping
CREATE USER TS IDENTIFIED BY TS;
GRANT CONNECT, RESOURCE to TS;
GRANT EXECUTE ON DBMS_AQ TO TS;
GRANT EXECUTE ON DBMS_AQADM TO TS;

Rem Create an account for Customer Billing
Rem Customer Billing, for security reason, has an admin schema that
Rem hosts all the queue tables and an application schema from where
Rem the application runs.
CREATE USER CBADM IDENTIFIED BY CBADM;
GRANT CONNECT, RESOURCE to CBADM;
GRANT EXECUTE ON DBMS_AQ TO CBADM;
GRANT EXECUTE ON DBMS_AQADM TO CBADM;

CREATE USER CB IDENTIFIED BY CB;
GRANT CONNECT, RESOURCE to CB;
GRANT EXECUTE ON DBMS_AQ TO CB;
GRANT EXECUTE ON DBMS_AQADM TO CB;

Rem Create an account for Customer Service
CREATE USER CS IDENTIFIED BY CS;
GRANT CONNECT, RESOURCE to CS;
GRANT EXECUTE ON DBMS_AQ TO CS;
GRANT EXECUTE ON DBMS_AQADM TO CS;


Rem All object types are created in the administrator schema.
Rem All application schemas that host any propagation source
Rem queues are given the ENQUEUE_ANY system level privilege
Rem allowing the application schemas to enqueue to the destination
Rem queue.
Rem
connect BOLADM/BOLADM;

Rem Create objects

CREATE OR REPLACE TYPE customer_typ AS object (
        custno          number,
        name            varchar2(100),
        street          varchar2(100),
        city            varchar2(30),
```

```
        state           varchar2(2),
        zip             number,
        country         varchar2(100));
/

CREATE OR REPLACE TYPE book_typ AS object (
        title           varchar2(100),
        authors         varchar2(100),
        ISBN            number,
        price           number);
/

CREATE OR REPLACE TYPE orderitem_typ AS object (
        quantity        number,
        item            book_typ,
        subtotal        number);
/

CREATE OR REPLACE TYPE orderitemlist_vartyp AS varray (20) of orderitem_typ;
/

CREATE OR REPLACE TYPE order_typ AS object (
        orderno         number,
        status          varchar2(30),
        ordertype       varchar2(30),
        orderregion     varchar2(30),
        customer        customer_typ,
        paymentmethod   varchar2(30),
        items           orderitemlist_vartyp,
        total           number);
/

GRANT EXECUTE ON ORDER_TYP TO OE;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO OE;
GRANT EXECUTE ON ORDERITEM_TYP TO OE;
GRANT EXECUTE ON BOOK_TYP TO OE;
GRANT EXECUTE ON CUSTOMER_TYP TO OE;
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','OE',FALSE);

GRANT EXECUTE ON ORDER_TYP TO WS;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO WS;
GRANT EXECUTE ON ORDERITEM_TYP TO WS;
GRANT EXECUTE ON BOOK_TYP TO WS;
GRANT EXECUTE ON CUSTOMER_TYP TO WS;
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','WS',FALSE);
```

```
GRANT EXECUTE ON ORDER_TYP TO ES;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO ES;
GRANT EXECUTE ON ORDERITEM_TYP TO ES;
GRANT EXECUTE ON BOOK_TYP TO ES;
GRANT EXECUTE ON CUSTOMER_TYP TO ES;
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','ES',FALSE);

GRANT EXECUTE ON ORDER_TYP TO TS;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO TS;
GRANT EXECUTE ON ORDERITEM_TYP TO TS;
GRANT EXECUTE ON BOOK_TYP TO TS;
GRANT EXECUTE ON CUSTOMER_TYP TO TS;
execute DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY','TS',FALSE);

GRANT EXECUTE ON ORDER_TYP TO CBADM;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO CBADM;
GRANT EXECUTE ON ORDERITEM_TYP TO CBADM;
GRANT EXECUTE ON BOOK_TYP TO CBADM;
GRANT EXECUTE ON CUSTOMER_TYP TO CBADM;

GRANT EXECUTE ON ORDER_TYP TO CB;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO CB;
GRANT EXECUTE ON ORDERITEM_TYP TO CB;
GRANT EXECUTE ON BOOK_TYP TO CB;
GRANT EXECUTE ON CUSTOMER_TYP TO CB;

GRANT EXECUTE ON ORDER_TYP TO CS;
GRANT EXECUTE ON ORDERITEMLIST_VARTYP TO CS;
GRANT EXECUTE ON ORDERITEM_TYP TO CS;
GRANT EXECUTE ON BOOK_TYP TO CS;
GRANT EXECUTE ON CUSTOMER_TYP TO CS;

Rem Create queue tables, queues for OE
Rem
connect OE/OE;
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'OE_orders_sqtab',
        comment => 'Order Entry Single Consumer Orders queue table',
        queue_payload_type => 'BOLADM.order_typ',
        message_grouping => DBMS_AQADM.TRANSACTIONAL,
        compatible => '8.1',
        primary_instance => 1,
        secondary_instance => 2);
```

```
end;
/

Rem Create a priority queue table for OE
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'OE_orders_pr_mqtab',
        sort_list =>'priority,enq_time',
        comment => 'Order Entry Priority MultiConsumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1',
        primary_instance => 2,
        secondary_instance => 1);
end;
/

begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'OE_neworders_que',
        queue_table             => 'OE_orders_sqtab');
end;
/

begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'OE_bookedorders_que',
        queue_table             => 'OE_orders_pr_mqtab');
end;
/

Rem Orders in OE_bookedorders_que are being propagated to WS_bookedorders_que,
Rem ES_bookedorders_que and TS_bookedorders_que according to the region
Rem the books are shipped to.  At the time an order is placed, the customer
Rem can request Fed-ex shipping (priority 1), priority air shipping (priority
Rem 2) and ground shipping (priority 3).  A priority queue is created in
Rem each region, the shipping applications dequeue from these priority
Rem queues according to the orders' shipping priorities, process the orders
Rem and enqueue the processed orders into
Rem the shipped_orders queues or the back_orders queues.  Both the shipped_
Rem orders queues and the back_orders queues are FIFO queues. However,
Rem orders put into the back_orders_queues are enqueued with delay time
Rem set to 1 day, so that each order in the back_order_queues is processed
Rem only once a day until the shipment is filled.
```

```
Rem Create queue tables, queues for WS Shipping
connect WS/WS;

Rem Create a priority queue table for WS shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'WS_orders_pr_mqtab',
        sort_list =>'priority,enq_time',
        comment => 'West Shipping Priority MultiConsumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Create a FIFO queue tables for WS shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'WS_orders_mqtab',
        comment => 'West Shipping Multi Consumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'WS_bookedorders_que',
        queue_table             => 'WS_orders_pr_mqtab');
end;
/

Rem Shipped orders and backorders are stored in the FIFO queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'WS_shippedorders_que',
        queue_table             => 'WS_orders_mqtab');
end;
/


begin
DBMS_AQADM.CREATE_QUEUE (
```

```
        queue_name              => 'WS_backorders_que',
        queue_table             => 'WS_orders_mqtab');
end;
/

Rem
Rem  In order to test history, set retention to 1 DAY for the queues
Rem  in WS

begin
DBMS_AQADM.ALTER_QUEUE(
         queue_name => 'WS_bookedorders_que',
         retention_time => 86400);
end;
/

begin
DBMS_AQADM.ALTER_QUEUE(
         queue_name => 'WS_shippedorders_que',
         retention_time => 86400);
end;
/

begin
DBMS_AQADM.ALTER_QUEUE(
         queue_name => 'WS_backorders_que',
         retention_time => 86400);
end;
/


Rem Create queue tables, queues for ES Shipping
connect ES/ES;

Rem Create a priority queue table for ES shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'ES_orders_mqtab',
        comment => 'East Shipping Multi Consumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/
```

```
Rem Create a FIFO queue tables for ES shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'ES_orders_pr_mqtab',
        sort_list =>'priority,enq_time',
        comment => 'East Shipping Priority Multi Consumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'ES_bookedorders_que',
        queue_table             => 'ES_orders_pr_mqtab');
end;
/

Rem Shipped orders and backorders are stored in the FIFO queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'ES_shippedorders_que',
        queue_table             => 'ES_orders_mqtab');
end;
/

begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'ES_backorders_que',
        queue_table             => 'ES_orders_mqtab');
end;
/


Rem Create queue tables, queues for Overseas Shipping
connect TS/TS;

Rem Create a priority queue table for TS shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'TS_orders_pr_mqtab',
        sort_list =>'priority,enq_time',
        comment => 'Overseas Shipping Priority MultiConsumer Orders queue
```

```
table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Create a FIFO queue tables for TS shipping
begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'TS_orders_mqtab',
        comment => 'Overseas Shipping Multi Consumer Orders queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');
end;
/

Rem Booked orders are stored in the priority queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name             => 'TS_bookedorders_que',
        queue_table            => 'TS_orders_pr_mqtab');
end;
/

Rem Shipped orders and backorders are stored in the FIFO queue table
begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name             => 'TS_shippedorders_que',
        queue_table            => 'TS_orders_mqtab');
end;
/

begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name             => 'TS_backorders_que',
        queue_table            => 'TS_orders_mqtab');
end;
/


Rem Create queue tables, queues for Customer Billing
connect CBADM/CBADM;
begin
```

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'CBADM_orders_sqtab',
        comment => 'Customer Billing Single Consumer Orders queue table',
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');

DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'CBADM_orders_mqtab',
        comment => 'Customer Billing Multi Consumer Service queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');

DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CBADM_shippedorders_que',
        queue_table             => 'CBADM_orders_sqtab');

end;
/

Rem Grant dequeue privilege on the shopped orders queue to the Customer Billing
Rem application.  The CB application retrieves shipped orders (not billed yet)
Rem from the shopped orders queue.
execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE', 'CBADM_shippedorders_que',
'CB', FALSE);

begin
DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CBADM_billedorders_que',
        queue_table             => 'CBADM_orders_mqtab');
end;
/

Rem Grant enqueue privilege on the billed orders queue to Customer Billing
Rem application.  The CB application is allowed to put billed orders into
Rem this queue.
execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('ENQUEUE', 'CBADM_billedorders_que',
'CB', FALSE);


Rem Customer support tracks the state of the customer request in the system
Rem
Rem At any point, customer request can be in one of the following states
Rem  A. BOOKED  B. SHIPPED  C. BACKED   D. BILLED
Rem  Given the order number the customer support returns the state
```

```
Rem  the order is in. This state is maintained in the order_status_table

connect CS/CS;

CREATE TABLE Order_Status_Table(customer_order       boladm.order_typ,
                                status               varchar2(30));

Rem Create queue tables, queues for Customer Service

begin
DBMS_AQADM.CREATE_QUEUE_TABLE(
        queue_table => 'CS_order_status_qt',
        comment => 'Customer Status multi consumer queue table',
        multiple_consumers => TRUE,
        queue_payload_type => 'BOLADM.order_typ',
        compatible => '8.1');

DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CS_bookedorders_que',
        queue_table             => 'CS_order_status_qt');

DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CS_backorders_que',
        queue_table             => 'CS_order_status_qt');

DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CS_shippedorders_que',
        queue_table             => 'CS_order_status_qt');

DBMS_AQADM.CREATE_QUEUE (
        queue_name              => 'CS_billedorders_que',
        queue_table             => 'CS_order_status_qt');

end;
/

Rem Create the Subscribers for OE queues
Rem Add the Subscribers for the OE booked_orders queue

connect OE/OE;

Rem Add a rule-based subscriber for West Shipping
Rem West Shipping handles Western region US orders
Rem Rush Western region orders are handled by East Shipping
declare
```

```
  subscriber     aq$_agent;
begin
  subscriber := aq$_agent('West_Shipping', 'WS.WS_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'OE.OE_bookedorders_que',
                            subscriber => subscriber,
                            rule       =>
                              'tab.user_data.orderregion = ''WESTERN'' AND
                               tab.user_data.ordertype != ''RUSH''');
end;
/

Rem Add a rule-based subscriber for East Shipping
Rem East shipping handles all Eastern region orders
Rem East shipping also handles all US rush orders
declare
  subscriber     aq$_agent;
begin
  subscriber := aq$_agent('East_Shipping', 'ES.ES_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'OE.OE_bookedorders_que',
                            subscriber => subscriber,
                            rule       =>
                              'tab.user_data.orderregion = ''EASTERN'' OR
                              (tab.user_data.ordertype = ''RUSH'' AND
                               tab.user_data.customer.country = ''USA'') ');
end;
/

Rem Add a rule-based subscriber for Overseas Shipping
Rem Intl Shipping handles all non-US orders
declare
  subscriber     aq$_agent;
begin
  subscriber := aq$_agent('Overseas_Shipping', 'TS.TS_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'OE.OE_bookedorders_que',
                            subscriber => subscriber,
                            rule       => 'tab.user_data.orderregion =
''INTERNATIONAL''');
end;
/

Rem Add the Customer Service order queues as a subscribers to the
Rem corresponding queues  in OrderEntry, Shipping and Billing

declare
  subscriber     aq$_agent;
```

```
begin
  /* Subscribe to the booked orders queue */
  subscriber := aq$_agent('BOOKED_ORDER', 'CS.CS_bookedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'OE.OE_bookedorders_que',
                            subscriber => subscriber);
end;
/


connect WS/WS;

declare
  subscriber     aq$_agent;
begin
  /* Subscribe to the WS backorders queue */
  subscriber := aq$_agent('BACK_ORDER', 'CS.CS_backorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'WS.WS_backorders_que',
                            subscriber => subscriber);
end;
/

declare
  subscriber     aq$_agent;
begin
  /* Subscribe to the WS shipped orders queue */
  subscriber := aq$_agent('SHIPPED_ORDER', 'CS.CS_shippedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'WS.WS_shippedorders_que',
                            subscriber => subscriber);
end;
/

connect CBADM/CBADM;
declare
  subscriber     aq$_agent;
begin
  /* Subscribe to the BILLING billed orders queue */
  subscriber := aq$_agent('BILLED_ORDER', 'CS.CS_billedorders_que', null);
  DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'CBADM.CBADM_billedorders_que',
                            subscriber => subscriber);

end;
/


Rem
```

```
Rem BOLADM will Start all the queues
Rem
connect BOLADM/BOLADM
execute DBMS_AQADM.START_QUEUE(queue_name => 'OE.OE_neworders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'OE.OE_bookedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'WS.WS_bookedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'WS.WS_shippedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'WS.WS_backorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'ES.ES_bookedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'ES.ES_shippedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'ES.ES_backorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'TS.TS_bookedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'TS.TS_shippedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'TS.TS_backorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CBADM.CBADM_shippedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CBADM.CBADM_billedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CS.CS_bookedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CS.CS_backorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CS.CS_shippedorders_que');
execute DBMS_AQADM.START_QUEUE(queue_name => 'CS.CS_billedorders_que');

connect system/manager

Rem
Rem Start job_queue_processes to handle AQ propagation
Rem

alter system set job_queue_processes=4;
```

tkaqdocd.sql: Examples of Administrative and Operational Interfaces

```
Rem
Rem $Header: tkaqdocd.sql 26-jan-99.17:51:23 aquser1 Exp $
Rem
Rem tkaqdocd.sql
Rem
Rem  Copyright (c) Oracle 1998, 1999. All Rights Reserved.
Rem
Rem    NAME
Rem      tkaqdocd.sql - <one-line expansion of the name>
Rem
```

```
Rem    DESCRIPTION
Rem      <short description of component this file declares/defines>
Rem
Rem    NOTES
Rem      <other useful comments, qualifications, and so on>
Rem
Rem


Rem
Rem  Schedule propagation for the shipping, billing, order entry queues
Rem

connect OE/OE;

execute DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'OE.OE_bookedorders_que');

connect WS/WS;
execute DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'WS.WS_backorders_que');
execute DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'WS.WS_shippedorders_
que');

connect CBADM/CBADM;
execute DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'CBADM.CBADM_billedorders_
que');


Rem
Rem   Customer service application
Rem
Rem   This application monitors the status queue for messages and updates
Rem   the Order_Status table.


connect CS/CS

Rem
Rem  Dequeus messages from the 'queue' for 'consumer'

CREATE OR REPLACE PROCEDURE DEQUEUE_MESSAGE(
                         queue      IN   VARCHAR2,
                         consumer   IN   VARCHAR2,
                         message    OUT  BOLADM.order_typ)
IS
```

```
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_msgid                raw(16);
BEGIN
  dopt.dequeue_mode := dbms_aq.REMOVE;
  dopt.navigation := dbms_aq.FIRST_MESSAGE;
  dopt.consumer_name := consumer;

  dbms_aq.dequeue(
                queue_name => queue,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => message,
                msgid => deq_msgid);
  commit;
END;
/



Rem
Rem  Updates the status of the order in the status table
Rem

CREATE OR REPLACE PROCEDURE update_status(
                                new_status    IN VARCHAR2,
                                order_msg    IN BOLADM.ORDER_TYP)
IS
 old_status    VARCHAR2(30);
 dummy         NUMBER;
BEGIN

  BEGIN
    /* query old status from the table */
    SELECT st.status INTO old_status from order_status_table st
       where st.customer_order.orderno = order_msg.orderno;

  /* Status can be 'BOOKED_ORDER', 'SHIPPED_ORDER', 'BACK_ORDER'
   *         and   'BILLED_ORDER'
   */

   IF new_status = 'SHIPPED_ORDER' THEN
      IF old_status = 'BILLED_ORDER' THEN
        return;             /* message about a previous state */
      END IF;
```

```
    ELSIF new_status = 'BACK_ORDER' THEN
       IF old_status = 'SHIPPED_ORDER' OR old_status = 'BILLED_ORDER' THEN
         return;             /* message about a previous state */
       END IF;
    END IF;

    /* update the order status  */
      UPDATE order_status_table st
         SET st.customer_order = order_msg, st.status = new_status
         where st.customer_order.orderno = order_msg.orderno;

    COMMIT;

   EXCEPTION
   WHEN OTHERS  THEN     /* change to no data found */
     /* first update for the order */
     INSERT INTO order_status_table(customer_order, status)
     VALUES (order_msg, new_status);
     COMMIT;

   END;
 END;
 /


 Rem
 Rem  Monitors the customer service queues for 'time' seconds
 Rem

  CREATE OR REPLACE PROCEDURE MONITOR_STATUS_QUEUE(time  IN  NUMBER)
 IS
   agent_w_message   aq$_agent;
   agent_list        dbms_aq.aq$_agent_list_t;
   wait_time         INTEGER := 120;
   no_message        EXCEPTION;
   pragma EXCEPTION_INIT(no_message, -25254);
   order_msg         boladm.order_typ;
   new_status        VARCHAR2(30);
   monitor           BOOLEAN := TRUE;
   begin_time        number;
   end_time          number;
 BEGIN

  begin_time :=  dbms_utility.get_time;
  WHILE (monitor)
```

```
 LOOP
 BEGIN
  agent_list(1) := aq$_agent('BILLED_ORDER', 'CS_billedorders_que', NULL);
  agent_list(2) := aq$_agent('SHIPPED_ORDER', 'CS_shippedorders_que', NULL);
  agent_list(3) := aq$_agent('BACK_ORDER', 'CS_backorders_que', NULL);
  agent_list(4) := aq$_agent('Booked_ORDER', 'CS_bookedorders_que', NULL);

   /* wait for order status messages */
   dbms_aq.listen(agent_list, wait_time, agent_w_message);

   dbms_output.put_line('Agent' || agent_w_message.name || ' Address '|| agent_
w_message.address);
   /* dequeue the message from the queue */
   dequeue_message(agent_w_message.address, agent_w_message.name, order_msg);

   /* update the status of the order depending on the type of the message
    * the name of the agent contains the new state
    */
   update_status(agent_w_message.name, order_msg);

  /* exit if we have been working long enough */
   end_time := dbms_utility.get_time;
   IF  (end_time - begin_time > time)   THEN
     EXIT;
   END IF;

  EXCEPTION
  WHEN  no_message  THEN
    dbms_output.put_line('No messages in the past 2 minutes');
       end_time := dbms_utility.get_time;
    /* exit if we have done enough work */
    IF  (end_time - begin_time > time)   THEN
      EXIT;
    END IF;
  END;

  END LOOP;
END;
/




Rem
Rem   History queries
```

```
Rem

Rem
Rem   Average processing time for messages in western shipping:
Rem   Difference between the ship- time and book-time for the order
Rem
Rem   NOTE: we assume that order ID is the correlation identifier
Rem         Only processed messages are considered.

Connect WS/WS


SELECT  SUM(SO.enq_time - BO.enq_time) / count (*) AVG_PRCS_TIME
 FROM WS.AQ$WS_orders_pr_mqtab BO , WS.AQ$WS_orders_mqtab SO
 WHERE SO.msg_state = 'PROCESSED' and BO.msg_state = 'PROCESSED'
 AND SO.corr_id = BO.corr_id and SO.queue = 'WS_shippedorders_que';


Rem
Rem  Average backed up time (again only processed messages are considered
Rem

SELECT SUM(BACK.deq_time - BACK.enq_time)/count (*) AVG_BACK_TIME
  FROM WS.AQ$WS_orders_mqtab BACK
  WHERE BACK.msg_state = 'PROCESSED' and BACK.queue = 'WS_backorders_que';
```

tkaqdoce.sql: Operational Examples

```
Rem
Rem $Header: tkaqdoce.sql 26-jan-99.17:51:28 aquser1 Exp $
Rem
Rem tkaqdocl.sql
Rem
Rem  Copyright (c) Oracle 1998, 1999. All Rights Reserved.
Rem


set echo on

Rem ==================================================================
Rem       Demonstrate enqueuing a backorder with delay time set
Rem       to 1 day.  This guarantees that each backorder is
```

```
Rem        processed only once a day until the order is filled.
Rem ==================================================================

Rem Create a package that enqueue with delay set to one day
connect BOLADM/BOLADM
create or replace procedure requeue_unfilled_order(sale_region varchar2,
                                                   backorder order_typ)
as
  back_order_queue_name    varchar2(62);
  enqopt                   dbms_aq.enqueue_options_t;
  msgprop                  dbms_aq.message_properties_t;
  enq_msgid                raw(16);
begin
  -- Choose a backorder queue based the the region
  IF sale_region = 'WEST' THEN
    back_order_queue_name := 'WS.WS_backorders_que';
  ELSIF sale_region = 'EAST' THEN
    back_order_queue_name := 'ES.ES_backorders_que';
  ELSE
    back_order_queue_name := 'TS.TS_backorders_que';
  END IF;

  -- Enqueue the order with delay time set to 1 day
  msgprop.delay := 60*60*24;
  dbms_aq.enqueue(back_order_queue_name, enqopt, msgprop,
                  backorder, enq_msgid);
end;
```

tkaqdocp.sql: Examples of Operational Interfaces

```
Rem
Rem $Header: tkaqdocp.sql 26-jan-99.17:50:54 aquser1 Exp $
Rem
Rem tkaqdocp.sql
Rem
Rem  Copyright (c) Oracle 1998, 1999. All Rights Reserved.
Rem
Rem    NAME
Rem      tkaqdocp.sql - <one-line expansion of the name>
Rem

set echo on;
```

```
Rem =================================================================
Rem         Illustrating Support for Real Application Clusters
Rem =================================================================

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem check instance affinity of OE queue tables from AQ administrative view

select queue_table, primary_instance, secondary_instance, owner_instance
from user_queue_tables;

Rem alter instance affinity of OE queue tables

begin
DBMS_AQADM.ALTER_QUEUE_TABLE(
        queue_table => 'OE.OE_orders_sqtab',
        primary_instance => 2,
        secondary_instance => 1);
end;
/

begin
DBMS_AQADM.ALTER_QUEUE_TABLE(
        queue_table => 'OE.OE_orders_pr_mqtab',
        primary_instance => 1,
        secondary_instance => 2);
end;
/

Rem check instance affinity of OE queue tables from AQ administrative view

select queue_table, primary_instance, secondary_instance, owner_instance
from user_queue_tables;

Rem =================================================================
Rem                    Illustrating Propagation Scheduling
Rem =================================================================

Rem Login into OE account

set echo on;
connect OE/OE;
set serveroutput on;
```

```
Rem
Rem Schedule Propagation from bookedorders_que to shipping
Rem

execute DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'OE.OE_bookedorders_que');

Rem Login into boladm account
set echo on;
connect boladm/boladm;
set serveroutput on;

Rem create a procedure to enqueue an order
create or replace procedure order_enq(book_title   in varchar2,
                                      book_qty     in number,
                                      order_num    in number,
                                      shipping_priority in number,
                                      cust_state   in varchar2,
                                      cust_country in varchar2,
                                      cust_region  in varchar2,
                                      cust_ord_typ in varchar2) as

OE_enq_order_data        BOLADM.order_typ;
OE_enq_cust_data         BOLADM.customer_typ;
OE_enq_book_data         BOLADM.book_typ;
OE_enq_item_data         BOLADM.orderitem_typ;
OE_enq_item_list         BOLADM.orderitemlist_vartyp;
enqopt                   dbms_aq.enqueue_options_t;
msgprop                  dbms_aq.message_properties_t;
enq_msgid                raw(16);

begin

        msgprop.correlation := cust_ord_typ;
        OE_enq_cust_data := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
                                cust_state, NULL, cust_country);
        OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
        OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL);
        OE_enq_item_list := BOLADM.orderitemlist_vartyp(
                                BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL));
        OE_enq_order_data := BOLADM.order_typ(order_num, NULL,
                                cust_ord_typ, cust_region,
                                OE_enq_cust_data, NULL,
```

```
                                OE_enq_item_list, NULL);

        -- Put the shipping priority into message property before
        -- enqueuing the message
        msgprop.priority := shipping_priority;
        dbms_aq.enqueue('OE.OE_bookedorders_que', enqopt, msgprop,
                        OE_enq_order_data, enq_msgid);
end;
/

show errors;

GRANT EXECUTE ON ORDER_ENQ TO OE;

Rem now create a procedure to dequeue booked orders for shipment processing
create or replace procedure shipping_bookedorder_deq(
                                        consumer in varchar2,
                                        deqmode in binary_integer) as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
qname                    varchar2(30);
no_messages              exception;
pragma exception_init    (no_messages, -25228);
new_orders               BOOLEAN := TRUE;

begin

        dopt.consumer_name := consumer;
        dopt.wait := DBMS_AQ.NO_WAIT;
        dopt.dequeue_mode := deqmode;
        dopt.navigation := dbms_aq.FIRST_MESSAGE;

        IF (consumer = 'West_Shipping') THEN
                qname := 'WS.WS_bookedorders_que';
        ELSIF (consumer = 'East_Shipping') THEN
                qname := 'ES.ES_bookedorders_que';
        ELSE
                qname := 'TS.TS_bookedorders_que';
        END IF;
```

```
        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;
            deq_cust_data := deq_order_data.customer;

            dbms_output.put_line(' **** next booked order **** ');
            dbms_output.put_line('order_num: ' || deq_order_data.orderno ||
                        ' book_title: ' || deq_book_data.title ||
                        ' quantity: ' || deq_item_data.quantity);
            dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
                        ' ship_country: ' || deq_cust_data.country ||
                        ' ship_order_type: ' || deq_order_data.ordertype);
            dopt.navigation := dbms_aq.NEXT_MESSAGE;
          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE BOOKED ORDERS ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/
show errors;

Rem now create a procedure to dequeue rush orders for shipment
create or replace procedure get_rushtitles(consumer in varchar2) as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
qname                    varchar2(30);
no_messages              exception;
```

```
pragma exception_init    (no_messages, -25228);
new_orders               BOOLEAN := TRUE;

begin

        dopt.consumer_name := consumer;
        dopt.wait := 1;
        dopt.correlation := 'RUSH';

        IF (consumer = 'West_Shipping') THEN
                qname := 'WS.WS_bookedorders_que';
        ELSIF (consumer = 'East_Shipping') THEN
                qname := 'ES.ES_bookedorders_que';
        ELSE
                qname := 'TS.TS_bookedorders_que';
        END IF;

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;

            dbms_output.put_line(' rushorder book_title: ' ||
                                    deq_book_data.title ||
                        ' quantity: ' || deq_item_data.quantity);
          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE RUSH TITLES ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/
show errors;

Rem now create a procedure to dequeue orders for handling North American
Rem orders
```

```
create or replace procedure get_northamerican_orders as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
deq_order_nodata         BOLADM.order_typ;
qname                    varchar2(30);
no_messages              exception;
pragma exception_init    (no_messages, -25228);
new_orders               BOOLEAN := TRUE;

begin

        dopt.consumer_name := 'Overseas_Shipping';
        dopt.wait := DBMS_AQ.NO_WAIT;
        dopt.navigation := dbms_aq.FIRST_MESSAGE;
        dopt.dequeue_mode := DBMS_AQ.LOCKED;

        qname := 'TS.TS_bookedorders_que';

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name => qname,
                dequeue_options => dopt,
                message_properties => mprop,
                payload => deq_order_data,
                msgid => deq_msgid);

            deq_item_data := deq_order_data.items(1);
            deq_book_data := deq_item_data.item;
            deq_cust_data := deq_order_data.customer;

            IF (deq_cust_data.country = 'Canada' OR
                deq_cust_data.country = 'Mexico' ) THEN

                dopt.dequeue_mode := dbms_aq.REMOVE_NODATA;
                dopt.msgid := deq_msgid;
                dbms_aq.dequeue(
                        queue_name => qname,
                        dequeue_options => dopt,
```

```
                        message_properties => mprop,
                        payload => deq_order_nodata,
                        msgid => deq_msgid);

                dbms_output.put_line(' **** next booked order **** ');
                dbms_output.put_line('order_no: ' || deq_order_data.orderno ||
                        ' book_title: ' || deq_book_data.title ||
                        ' quantity: ' || deq_item_data.quantity);
                dbms_output.put_line('ship_state: ' || deq_cust_data.state ||
                        ' ship_country: ' || deq_cust_data.country ||
                        ' ship_order_type: ' || deq_order_data.ordertype);

            END IF;

            commit;
            dopt.dequeue_mode := DBMS_AQ.LOCKED;
            dopt.msgid := NULL;
            dopt.navigation := dbms_aq.NEXT_MESSAGE;
          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE BOOKED ORDERS ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/
show errors;

GRANT EXECUTE ON SHIPPING_BOOKEDORDER_DEQ TO WS;
GRANT EXECUTE ON SHIPPING_BOOKEDORDER_DEQ TO ES;
GRANT EXECUTE ON SHIPPING_BOOKEDORDER_DEQ TO TS;
GRANT EXECUTE ON SHIPPING_BOOKEDORDER_DEQ TO CS;

GRANT EXECUTE ON GET_RUSHTITLES TO ES;

GRANT EXECUTE ON GET_NORTHAMERICAN_ORDERS TO TS;

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem
Rem Enqueue some orders into OE_bookedorders_que
Rem
```

```
execute BOLADM.order_enq('My First   Book', 1, 1001, 3,'CA', 'USA', 'WESTERN',
'NORMAL');
execute BOLADM.order_enq('My Second  Book', 2, 1002, 3,'NY', 'USA', 'EASTERN',
'NORMAL');
execute BOLADM.order_enq('My Third   Book', 3, 1003, 3, '',   'Canada',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Fourth  Book', 4, 1004, 2, 'NV', 'USA', 'WESTERN',
'RUSH');
execute BOLADM.order_enq('My Fifth   Book', 5, 1005, 2, 'MA', 'USA', 'EASTERN',
'RUSH');
execute BOLADM.order_enq('My Sixth   Book', 6, 1006, 3,''  , 'UK',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Seventh Book', 7, 1007, 1,'',   'Canada',
'INTERNATIONAL', 'RUSH');
execute BOLADM.order_enq('My Eighth  Book', 8, 1008, 3,'',   'Mexico',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Ninth   Book', 9, 1009, 1, 'CA', 'USA', 'WESTERN',
'RUSH');
execute BOLADM.order_enq('My Tenth   Book', 8, 1010, 3, ''  , 'UK',
'INTERNATIONAL', 'NORMAL');
execute BOLADM.order_enq('My Last    Book', 7, 1011, 3, ''  , 'Mexico',
'INTERNATIONAL', 'NORMAL');
commit;
/

Rem
Rem Wait for Propagation to Complete
Rem

execute dbms_lock.sleep(100);

Rem ==================================================================
Rem                  Illustrating Dequeue Modes/Methods
Rem ==================================================================


connect WS/WS;
set serveroutput on;

Rem Dequeue all booked orders for West_Shipping
execute BOLADM.shipping_bookedorder_deq('West_Shipping', DBMS_AQ.REMOVE);
commit;
/
```

```
connect ES/ES;
set serveroutput on;

Rem Browse all booked orders for East_Shipping
execute BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.BROWSE);

Rem Dequeue all rush order titles for East_Shipping
execute BOLADM.get_rushtitles('East_Shipping');
commit;
/

Rem Dequeue all remaining booked orders (normal order) for East_Shipping
execute BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.REMOVE);
commit;
/

connect TS/TS;
set serveroutput on;

Rem Dequeue all international North American orders for Overseas_Shipping
execute BOLADM.get_northamerican_orders;
commit;
/

Rem Dequeue rest of the booked orders for Overseas_Shipping
execute BOLADM.shipping_bookedorder_deq('Overseas_Shipping', DBMS_AQ.REMOVE);
commit;
/


Rem ==================================================================
Rem           Illustrating Enhanced Propagation Capabilities
Rem ==================================================================

connect OE/OE;
set serveroutput on;

Rem
Rem Get propagation schedule information & statistics
Rem

Rem get averages
select avg_time, avg_number, avg_size from user_queue_schedules;

Rem get totals
```

```
select total_time, total_number, total_bytes from user_queue_schedules;

Rem get status information of schedule (present only when active)
select process_name, session_id, instance, schedule_disabled
        from user_queue_schedules;

Rem get information about last and next execution
select last_run_date, last_run_time, next_run_date, next_run_time
        from user_queue_schedules;

Rem get last error information if any
select failures, last_error_msg, last_error_date, last_error_time
        from user_queue_schedules;

Rem disable propagation schedule for booked orders

execute DBMS_AQADM.DISABLE_PROPAGATION _SCHEDULE(queue_name => 'OE_bookedorders_
que');
execute dbms_lock.sleep(30);
select schedule_disabled from user_queue_schedules;

Rem alter propagation schedule for booked orders to execute every
Rem 15 mins (900 seconds) for a window duration of 300 seconds

begin
DBMS_AQADM.ALTER_PROPAGATION _SCHEDULE(
        queue_name => 'OE_bookedorders_que',
        duration => 300,
        next_time => 'SYSDATE + 900/86400',
        latency => 25);
end;
/

execute dbms_lock.sleep(30);
select next_time, latency, propagation_window from user_queue_schedules;

Rem enable propagation schedule for booked orders

execute DBMS_AQADM.ENABLE_PROPAGATION _SCHEDULE(queue_name => 'OE_bookedorders_
que');
execute dbms_lock.sleep(30);
select schedule_disabled from user_queue_schedules;

Rem unschedule propagation for booked orders
```

```
execute DBMS_AQADM.UNSCHEDULE_PROPAGATION(queue_name => 'OE.OE_bookedorders_
que');

set echo on;

Rem =================================================================
Rem                       Illustrating Message Grouping
Rem =================================================================

Rem Login into boladm account
set echo on;
connect boladm/boladm;
set serveroutput on;

Rem now create a procedure to handle order entry
create or replace procedure new_order_enq(book_title   in varchar2,
                                          book_qty     in number,
                                          order_num    in number,
                                          cust_state   in varchar2) as

OE_enq_order_data        BOLADM.order_typ;
OE_enq_cust_data         BOLADM.customer_typ;
OE_enq_book_data         BOLADM.book_typ;
OE_enq_item_data         BOLADM.orderitem_typ;
OE_enq_item_list         BOLADM.orderitemlist_vartyp;
enqopt                   dbms_aq.enqueue_options_t;
msgprop                  dbms_aq.message_properties_t;
enq_msgid                raw(16);

begin

        OE_enq_cust_data := BOLADM.customer_typ(NULL, NULL, NULL, NULL,
                                cust_state, NULL, NULL);
        OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
        OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL);
        OE_enq_item_list := BOLADM.orderitemlist_vartyp(
                                BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL));
        OE_enq_order_data := BOLADM.order_typ(order_num, NULL,
                                NULL, NULL,
                                OE_enq_cust_data, NULL,
                                OE_enq_item_list, NULL);
        dbms_aq.enqueue('OE.OE_neworders_que', enqopt, msgprop,
                        OE_enq_order_data, enq_msgid);
```

```
end;
/
show errors;

Rem now create a procedure to handle order enqueue
create or replace procedure same_order_enq(book_title   in varchar2,
                                           book_qty     in number) as

OE_enq_order_data        BOLADM.order_typ;
OE_enq_book_data         BOLADM.book_typ;
OE_enq_item_data         BOLADM.orderitem_typ;
OE_enq_item_list         BOLADM.orderitemlist_vartyp;
enqopt                   dbms_aq.enqueue_options_t;
msgprop                  dbms_aq.message_properties_t;
enq_msgid                raw(16);

begin

        OE_enq_book_data := BOLADM.book_typ(book_title, NULL, NULL, NULL);
        OE_enq_item_data := BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL);
        OE_enq_item_list := BOLADM.orderitemlist_vartyp(
                                BOLADM.orderitem_typ(book_qty,
                                OE_enq_book_data, NULL));
        OE_enq_order_data := BOLADM.order_typ(NULL, NULL,
                                NULL, NULL,
                                NULL, NULL,
                                OE_enq_item_list, NULL);
        dbms_aq.enqueue('OE.OE_neworders_que', enqopt, msgprop,
                        OE_enq_order_data, enq_msgid);
end;
/
show errors;

GRANT EXECUTE ON NEW_ORDER_ENQ TO OE;
GRANT EXECUTE ON SAME_ORDER_ENQ TO OE;

Rem now create a procedure to get new orders by dequeuing
create or replace procedure get_new_orders as

deq_cust_data            BOLADM.customer_typ;
deq_book_data            BOLADM.book_typ;
deq_item_data            BOLADM.orderitem_typ;
deq_msgid                RAW(16);
dopt                     dbms_aq.dequeue_options_t;
```

```
mprop                    dbms_aq.message_properties_t;
deq_order_data           BOLADM.order_typ;
qname                    varchar2(30);
no_messages              exception;
end_of_group             exception;
pragma exception_init    (no_messages, -25228);
pragma exception_init    (end_of_group, -25235);
new_orders               BOOLEAN := TRUE;

begin

        dopt.wait := 1;
        dopt.navigation := DBMS_AQ.FIRST_MESSAGE;
        qname := 'OE.OE_neworders_que';
        WHILE (new_orders) LOOP
          BEGIN
            LOOP
                BEGIN
                    dbms_aq.dequeue(
                        queue_name => qname,
                        dequeue_options => dopt,
                        message_properties => mprop,
                        payload => deq_order_data,
                        msgid => deq_msgid);

                    deq_item_data := deq_order_data.items(1);
                    deq_book_data := deq_item_data.item;
                    deq_cust_data := deq_order_data.customer;

                    IF (deq_cust_data IS NOT NULL) THEN
                      dbms_output.put_line(' **** NEXT ORDER **** ');
                      dbms_output.put_line('order_num: ' ||
                                deq_order_data.orderno);
                      dbms_output.put_line('ship_state: ' ||
                                deq_cust_data.state);
                    END IF;
                    dbms_output.put_line(' ---- next book ---- ');
                    dbms_output.put_line(' book_title: ' ||
                                deq_book_data.title ||
                                ' quantity: ' || deq_item_data.quantity);
                EXCEPTION
                    WHEN end_of_group THEN
                      dbms_output.put_line ('*** END OF ORDER ***');
                      commit;
                      dopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
```

```
                END;
            END LOOP;
          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE NEW ORDERS ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;

end;
/

show errors;

GRANT EXECUTE ON GET_NEW_ORDERS TO OE;

Rem Login into OE account
connect OE/OE;
set serveroutput on;

Rem
Rem Enqueue some orders using message grouping into OE_neworders_que
Rem

Rem First Order
execute BOLADM.new_order_enq('My First   Book', 1, 1001, 'CA');
execute BOLADM.same_order_enq('My Second  Book', 2);
commit;
/

Rem Second Order
execute BOLADM.new_order_enq('My Third   Book', 1, 1002, 'WA');
commit;
/

Rem Third Order
execute BOLADM.new_order_enq('My Fourth  Book', 1, 1003, 'NV');
execute BOLADM.same_order_enq('My Fifth   Book', 3);
execute BOLADM.same_order_enq('My Sixth   Book', 2);
commit;
/

Rem Fourth Order
execute BOLADM.new_order_enq('My Seventh Book', 1, 1004, 'MA');
execute BOLADM.same_order_enq('My Eighth  Book', 3);
```

```
execute BOLADM.same_order_enq('My Ninth   Book', 2);
commit;
/

Rem
Rem Dequeue the neworders
Rem

execute BOLADM.get_new_orders;
```

tkaqdocc.sql: Clean-Up Script

```
Rem
Rem $Header: tkaqdocc.sql 26-jan-99.17:51:05 aquser1 Exp $
Rem
Rem tkaqdocc.sql
Rem
Rem  Copyright (c) Oracle 1998, 1999. All Rights Reserved.
Rem
Rem    NAME
Rem      tkaqdocc.sql - <one-line expansion of the name>
Rem

set echo on;
connect system/manager
set serveroutput on;

drop user WS cascade;
drop user ES cascade;
drop user TS cascade;
drop user CB cascade;
drop user CBADM cascade;
drop user CS cascade;
drop user OE cascade;
drop user boladm cascade;
```

Glossary

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.

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 queues

Buffered queues store an enqueued **message** in the **SGA** instead of a **queue table**. Messages are spilled over to a queue table if they are not dequeued within a system-controlled period of time or if a specified memory threshold is exceeded. See **nonpersistent**.

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.

connection factory

A connection factory 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**.

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 Streams AQ 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 10*g*, or in a database server such as Oracle Database 10*g*.

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 Streams AQ 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.

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. See **buffered queues**.

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 Streams AQ based on the **Java Message Service** (JMS) standard. OJMS supports the standard JMS interfaces and has extensions to support the Oracle Streams AQ administrative operations and other Oracle Streams AQ 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 Streams AQ, 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

See **Simple Object Access Protocol**.

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.

Index

A

access control. *See* system-level access control, 5-4, 7-3
ADD_SUBSCRIBER procedure, 24-21
adding a subscriber, 8-26
administration
  Messaging Gateway, 18-4
administration user
  creating, 19-6
administrative interface, 5-4
AdtMessage, 11-27
Advanced Queuing
  operations over the Internet, 17-2
agent
  identifying, 3-3, 3-4
agent. *See* AQ agent, 8-38
agent user
  creating, 19-6
altering
  destination, 12-27
AnyData datatype
  message propagation, 23-7
  queues, 23-2, 23-9
    dequeuing, 23-12
    enqueuing, 23-9
    propagating to typed queues, 23-7
    user-defined types, 23-8
  wrapper for messages, 23-3, 23-9
apply process
  message handlers
    creating, 24-15
AQ agent
  altering, 8-38
  creating, 8-37
  dropping, 8-38
  registering, 17-49
AQ XML
  schema, 17-30
  servlet, 17-45, 17-52
AQ XML servlet
  registering for notifications, 7-110
AQ_TM_PROCESSES, 3-9
AQXmlPublish method, 17-6
AQXmlReceive method, 17-17
AQXmlSend method, 17-6
array
  dequeue message array, 10-34
  enqueue message array, 10-12
asynchronous notification, 1-23, 7-101
asynchronously receiving message, 11-54

B

batch dequeue, 10-34
batch enqueue, 10-12
bytes message, 11-25

C

commit response, 17-28
commit transaction, 17-22
compositing, 1-13
configuring
  connection information, 19-7
connection factory
  queue/topic, LDAP, 12-14
  topic, with JDBC URL, 12-12

unregistering in LDAP through database, 12-8
unregistering in LDAP through LDAP, 12-9
connection information
configuring, 19-7
CONVERT_ANYDATA_TO_LCR_DDL function, 23-17
CONVERT_ANYDATA_TO_LCR_ROW function, 23-17
correlation identifier, 1-21
creating
administration user, 19-6
agent user, 19-6
point-to-point queue, 12-18
queue, 8-13
queue table, 12-16
queue tables and queues, examples, 2-3
creating publish/subscribe topic, 12-18
creation of prioritized message queue table and queue, 2-4
creation of queue table and queue of object type, 2-3
creation of queue table and queue of RAW type, 2-4

D

database
tuning, 6-2
database access
enabling, 8-39
database objects
loading, 19-2
database session, 17-50
DBA_ATTRIBUTE_TRANSFORMATIONS, 9-20
DBA_QUEUE_TABLES, 9-2
DBA_QUEUES, 9-4
DBA_TRANSFORMATIONS, 9-19
DBMS_AQADM package, 5-3
DBMS_MGWADM package, 18-4
DBMS_TRANSFORM
applying a transformation, 7-10
create_transformation, 7-11
creating a single PL/SQL function, 7-9
DBMS_TRANSFORM package, 23-16, 23-18
delay, 3-9
time specification, 11-44
delay interval
retry with, 7-80
time specification, 7-48
dequeue
client request for, 17-17
message array, 10-34
dequeue mode, 3-9
dequeue of messages after preview, 2-32
DEQUEUE procedure, 23-12
example, 24-22
dequeue request
server response, 17-25
dequeuing, 10-28
features, 7-60
message navigation, 7-68
methods, 7-61
modes, 1-25, 7-73
multiple-consumer dequeuing of one message, 1-6
navigation of messages, 1-25
same message, multiple-consumer, 1-6
using HTTP, 7-111
destination
altering, 12-27
dropping, 12-28
starting, 12-25
stopping, 12-26
destination-level access control, 11-17
disabling
propagation schedule, 8-36
dropping
destination, 12-28
queue table, 8-8
dropping AQ objects, 2-59
durable subscriber, 11-36

E

enqueue
client request for, 17-6
message array, 10-12
server response, 17-23
enqueue and dequeue of messages
by Correlation and Message Id Using

Pro*C/C++, 2-37
by priority, 2-14, 2-16, 2-18
examples, 2-9
of object type, 2-11
of RAW type, 2-14, 2-16, 2-18
of RAW type using Pro*C/C++, 2-22, 2-24
to/from multiconsumer queues, 2-43, 2-46
ENQUEUE procedure, 23-11, 24-11
enqueuing, 10-2, 10-5
features, 7-38
specify message properties, 10-5
specify options, 10-3
enqueuing, priority and ordering of messages, 1-22
Enterprise Manager, 1-29
enumerated constants
administrative interface, 3-8
operational interface, 3-8
evaluation contexts
creating, 24-16
events
captured
propagating, 23-17
dequeue
programmatic environments, 23-3
enqueue
programmatic environments, 23-3
user-enqueued
propagating, 23-14
exception handling, 1-25, 7-84, 11-56
Exception Handling During Propagation, 11-67
exception handling during propagation, 11-65, 11-66
expiration, 3-9
time specification, 7-50
exporting
incremental, 5-7
queue table data, 5-6

F

fanning-out of messages, 1-13
frequently asked questions
Internet access questions, 17-66
Oracle Internet Directory, 17-67
transformation questions, 2-87
funneling-in of messages. *See* compositing, 1-13

G

gateway agent
managing, 20-2
getting
queue table, 12-17
global agents, 17-67
global events, 17-67
global queues, 17-67
granting
system privilege, 8-23
system privileges, 12-20
grouping
message, 11-45

H

HTTP, 1-18, 17-2, 17-5, 17-49, 17-55
accessing AQ XML servlet, 17-53
AQ operations over, 17-2
headers, 17-4
propagation, 17-57
response, 17-5
HTTPS
propagation, 17-57

I

IDAP
message, 17-6
schema, 17-32
transmitted over Internet, 17-1
IDAP. *See Internet Data Access Presentation*, 1-19, 17-2
INIT.ORA parameter, 3-9
Internet
access, 7-37
Advanced Queuing operations, 17-2
Advanced Queuing operations over, 17-1
AQ operations over, 1-4, 1-16
Internet Data Access Presentation (IDAP), 1-19, 17-2

J

JDBC
- connection parameters, registering through LDAP, 12-5
- connection parameters, registering through the database, 12-2
- connection parameters, topic connection factory, 12-13

JDBC URL
- registering through LDAP, 12-7
- registering through the database, 12-4

JMS
- Oracle Streams
  - example, 24-31

JMS Extension, 4-5
JMS Type queues/topics, 1-18
JMS types, 1-18
JOB_QUEUE_PROCESSES parameter, 3-10

L

LDAP
- queue/topic connection factory, 12-14
- queue/topic, getting, 12-15
- registering, 12-7
- unregistering, 12-8, 12-9

LDAP server
- with an AQ XML Servlet, 17-52

listen capability, 7-94
listener.ora file
- modifying, 19-3

loading
- database objects, 19-2

logical change records (LCRs)
- constructing
  - example, 24-11

M

managing
- gateway agent, 20-2

map message, 11-26
message
- fanning-out, 1-13
- grouping, 7-53
- history, 7-28
- navigation in dequeue, 7-68
- ordering, 7-40
- priority and ordering, 7-40, 11-43
- propagation, 1-13
- recipient, 1-8

message array
- dequeue, 10-34
- enqueue, 10-12

message enqueuing, 10-2
message grouping, 1-22, 11-45
message handlers
- creating, 24-15

message history and retention, 11-18
message navigation in receive, 11-48
message payloads, 1-17
message_grouping, 3-8
MessageProducer features, 11-43, 11-66
messaging
- propagation, 23-7

Messaging Gateway, 18-4
- administration, 18-4
- prerequisites for installing, 19-2

Messaging Gateway agent. *See* gateway agent
MGW_ADMINISTRATOR_ROLE role, 20-1, 21-5
mgw.ora file, 19-5
modifying
- listener.ora file, 19-3
- tnsnames.ora file, 19-4

multiple recipients, 1-24

N

navigation, 3-9
nonpersistent queue, 1-32, 2-85
- creating, 8-16

notification, 17-28
- asynchronous, 7-101

O

object message, 11-27
object types, 5-4
object_name, 3-2
optimization

arrival wait, 7-79
optimization of waiting for messages, 1-25
Oracle Extension, 4-5
Oracle Internet Directory, 17-67
Oracle object (ADT) type queues, 1-17
Oracle Real Application Clusters, 1-32, 7-32, 11-19
Oracle Streams
AnyData queues, 23-9
JMS, 23-3
example, 24-31
messaging, 23-9
OCI, 23-3

P

payload, 1-17
structured, 7-12
performance, 6-2
priority and ordering of messages, 11-43
privileges, 5-4
revoking, 2-60
privileges. *See specific privilege, such as* system privilege, topic privilege, 12-21
programmatic environments, 4-2
propagation, 1-23, 3-10, 7-111, 11-57, 17-57
exception handling, 11-65, 11-66
exception handling during, 11-65
failures, 5-15
features, 7-111
issues, 5-13
LOB attributes, 7-116
message, 1-13, 5-5
messages with LOB attributes, 7-116
processing, 18-6
schedule, 11-62
schedule, altering, 12-31
schedule, disabling, 12-32
scheduling, 1-27, 7-112, 7-118, 12-29
scheduling, enabling, 12-30
unscheduling, 12-33
using HTTP, 7-123
propagation schedule, 11-64
altering, 8-35
disabling, 8-36
enabling, 8-35
selecting, 9-5
selecting all, 9-5
Propagation, Exception Handling During, 11-67
propagations
transformations
SYS.AnyData to typed queue, 23-14, 23-17
publish/subscribe, 7-29
topic, 11-34

Q

QMN. *See* queue monitor (QMN), 1-28
queue
altering, 8-17
creating, 8-13
creating, example, 2-3
dropping, 8-18
nonpersistent, 1-32, 2-85, 8-16
point-to-point, 11-30
point-to-point, creating, 12-18
selecting all, 9-4
selecting in user schema, 9-13
selecting, in user schema, 9-13
selecting, user has any privilege, 9-7
selecting, user has queue privilege, 9-5
staring, 8-19
starting, 8-19
stopping, 8-20
subscriber rules, 9-17
subscriber, selecting, 9-16
subscribers, 1-7
subscribers, selecting, 9-16
queue monitor, 1-28
queue privilege
granting, 8-25
granting, point-to-point, 12-23
revoking, 8-25
revoking, point-to-point, 12-24
queue propagation
scheduling, 8-32
unscheduling, 8-33
queue subscribers
selecting, rules, 9-17
queue table
altering, 8-7

creating, 8-2, 12-16
creating prioritized message, 8-15
creating, example, 8-5, 8-6, 8-14
creating, example, XMLType attributes, 8-6
dropping, 8-8
getting, 12-17
messages, selecting, 9-9
selecting all, 9-2, 9-9
selecting messages, 9-9
selecting user tables, 9-3
queue table data
exporting, 5-6
queue tables
creating, example, 2-3
selecting all in user schema, 9-12
queue type
verifying, 8-33
queue_type, 3-8
queue-level access control, 1-32, 7-5
queues
AnyData, 23-2, 23-9
dequeuing, 24-22
enqueuing, 24-11
user-defined types, 23-8
propagation, 23-7
queue/topic
connection factory in LDAP, 12-14
connection factory, unregistering in LDAP through the database, 12-8
connection factory, unregistering in LDAP through the LDAP, 12-9
LDAP, 12-15

R

RAW queues, 1-17
Real Application Clusters. *See* Oracle Real Application Clusters, 1-32
receiving messages, 11-46
recipient, 1-8
list, 7-38, 11-38
local and remote, 1-24, 7-67
multiple, 7-66
REF
non-support of payload attribute, 5-12
register request
server response, 17-28
registering
AQ Agent, 17-49
JDBC connection parameters through LDAP, 12-5
JDBC URL through LDAP, 12-7
through the database, JDBC connection parameters, 12-2
through the database, JDBC URL, 12-4
registration
client request for, 17-6
to a queue, 3-5
registration for notification vs. listener, 2-85
retention, 3-8
retention and message history, 1-31, 7-28, 11-18
retries with delays, 1-25
retry
delay interval, 7-80
revoking roles and privileges, 2-60
role
revoking, 2-60
user, 5-3
rollback a transaction, 17-23
rollback response, 17-28
rule
selecting subscriber, 9-17
rule-based subscriber, 1-23
rule-based subscription, 7-90
rules
evaluation contexts
creating, 24-16

S

scheduling
propagation, 1-27, 11-62, 12-29
schema
AQ XML, 17-30
IDAP, 17-32
SOAP, 17-30
security, 5-3, 5-4
sender identification, 1-23
sequence_deviation, 3-9
servlet

AQ XML, 17-45, 17-52
SOAP, 17-30
body, 17-4
envelope, 17-3
headers, 17-3
message structure, 17-3
method invocation, 17-4
SYS.AnyData queues, 23-7
SOAP schema, 17-30
SQL access, 1-30
starting
destination, 12-25
state parameter, 3-9
statistics views, 7-37
statistics views support, 11-19
stopping
destination, 12-26
stream message, 11-25
structured payload, 1-30, 7-12
structured payload/message types, 11-20
subscriber, 3-4
adding, 8-26
altering, 8-29
durable, 11-36
removing, 8-30
rule-based, 1-23
selecting, 9-16
subscription, 7-38
anonymous, 3-5
rule-based, 7-90
subscription and recipient list, 1-21
subscription and recipient lists, 1-21
SYS.AnyData. *See Also* AnyData datatype
system privilege
granting, 8-23
revoking, 8-24
system privileges
granting, 12-20
system-level access control, 7-3, 11-17

T

text message, 11-26
time specification, 1-23
delay, 7-48, 11-44
expiration, 7-50, 11-44
tnsnames.ora file
modifying, 19-4
topic
connection factory, JDBC connection parameters, 12-13
connection factory, with JDBC URL, 12-12
publish/subscribe, creating, 12-18
topic privilege
granting, publish/subscribe, 12-21
revoking, publish/subscribe, 12-22
TopicPublisher, 11-38
tracking and event journals, 1-31
transaction protection, 1-25
transformations
propagations, 23-14, 23-17
tuning. *See* database tuning, 6-2
type_name, 3-2
types
object, 5-4

U

unloading
Messaging Gateway, 19-10
unregistering
queue/topic connection factory in LDAP, 12-8, 12-9
unscheduling
propagation, 12-33
user authentication, 17-48
user authorization, 17-49
user role, 5-3
USER_ATTRIBUTE_TRANSFORMATIONS, 9-21
USER_TRANSFORMATIONS, 9-21
user-defined datatypes
AnyData queues, 23-8
users
administration, 19-6
agent, 19-6

V

views
statistics, 7-37

visibility, 3-9

W

wait, 3-9
waiting
    for message arrival, 7-79

X

XA
    using with AQ, 5-10
XML, 17-1
    components, 1-18
    schema, 17-30
    servlet, 17-45, 17-52
    servlet, HTTP, 17-53