Oracle9*i*

Application Developer's Guide - Advanced Queuing

Release 1 (9.0.1)

July 2001

Part No. A88890-02

ORACLE®

Oracle9*i* Application Developer's Guide - Advanced Queuing, Release 1 (9.0.1)

Part No. A88890-02

Copyright © 2001, Oracle Corporation. All rights reserved.

Primary Authors: D.K. Bradshaw, Kevin MacDowell, Den Raphaely

Contributing Authors: Neerja Bhatt, Brajesh Goyal, Shelley Higgins, Rajit Kambo, Anish Karmarkar, Krishna Kunchithapadam, Vivek Maganty, Krishnan Meiyyappan, Bhagat Nainani, Wei Wang

Contributors: Sashi Chandrasekaran, Dieter Gawlick, Mohan Kamath, Goran Olsson, Hilkka Outinen, Madhu Reddy, Mary Rhodes, Ashok Saxena, Ekrem Soylemez, Alvin To, Rahim Yaseen

Graphics Production Specialist: Valarie Moore

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

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

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

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

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

Oracle is a registered trademark, and Oracle8, Oracle8*i*, Oracle9*i*, Oracle Real Application Clusters, PL/SQL, and SQL are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments xlvii

Preface xlix

Audience xlix
Organization xlix
Related Documentation lii
Conventions liii
Documentation Accessibility lv

What's New in Advanced Queuing? lvii

Oracle9*i* New Features in Advanced Queuing lviii
Oracle8*i* New Features in Advanced Queuing lxi

1 Introduction to Oracle Advanced Queuing

What Is Advanced Queuing? 1-2
Advanced Queuing in Integrated Application Environments 1-2
Interfaces to Advanced Queuing 1-3
Queuing System Requirements 1-4
General Features of Advanced Queuing 1-5
Point-to-Point and Publish-Subscribe Messaging 1-6
Oracle Internet Directory 1-7
Oracle Enterprise Manager Integration 1-7
Message Format Transformation 1-7
SQL Access 1-8

Support for Statistics Views 1-9
Structured Payloads 1-9
Retention and Message History 1-9
Tracking and Event Journals 1-10
Queue-Level Access Control 1-10
Nonpersistent Queues 1-10
Support for Oracle9*i* Real Application Clusters 1-11
XMLType Attributes in Object Types 1-11
Internet Integration and IDAP 1-12
Nonrepudiation and the AQ$<QueueTableName> View 1-14
Enqueue Features 1-14
Correlation Identifiers 1-15
Subscription and Recipient Lists 1-15
Priority and Ordering of Messages in Enqueuing 1-16
Message Grouping 1-16
Propagation 1-16
Sender Identification 1-17
Time Specification and Scheduling 1-17
Rule-Based Subscribers 1-17
Asynchronous Notification 1-17
Dequeue Features 1-18
Recipients 1-18
Navigation of Messages in Dequeuing 1-18
Modes of Dequeuing 1-18
Optimization of Waiting for the Arrival of Messages 1-18
Retries with Delays 1-19
Optional Transaction Protection 1-19
Exception Handling 1-19
Listen Capability (Wait on Multiple Queues) 1-19
Dequeue Message Header with No Payload 1-19
Propagation Features 1-20
Automated Coordination of Enqueuing and Dequeuing 1-20
Propagation of Messages with LOBs 1-20
Propagation Scheduling 1-20
Enhanced Propagation Scheduling Capabilities 1-20

Third-Party Support 1-21
Elements of Advanced Queuing 1-21
Message 1-21
Queue 1-22
Queue Table 1-22
Agent 1-22
Recipient 1-23
Recipient and Subscription Lists 1-23
Rule 1-24
Rule-Based Subscribers 1-24
Transformation 1-24
Queue Monitor 1-24
Java Messaging Service Terminology 1-25
Demos 1-25

2 Basic Components

Data Structures 2-2
Object Name (object_name) 2-2
Type Name (type_name) 2-2
Agent Type (aq$_agent) 2-3
AQ Recipient List Type (aq$_recipient_list_t) 2-4
AQ Agent List Type (aq$_agent_list_t) 2-4
AQ Subscriber List Type (aq$_subscriber_list_t) 2-4
AQ Registration Info List Type (aq$_reg_info_list) 2-5
AQ Post Info List Type (aq$_post_info_list) 2-5
AQ Registration Info Type 2-5
AQ Notification Descriptor Type 2-7
AQ Post Info Type 2-8
Enumerated Constants in the Administrative Interface 2-8
Enumerated Constants in the Operational Interface 2-9
INIT.ORA Parameter File Considerations 2-9
AQ_TM_PROCESSES Parameter 2-10
JOB_QUEUE_PROCESSES Parameter 2-10

3 AQ Programmatic Environments

Programmatic Environments for Accessing AQ 3-2
Using PL/SQL to Access AQ 3-3
Using OCI to Access AQ 3-4
Examples 3-5
Using Visual Basic (OO4O) to Access AQ 3-5
For More Information 3-6
Using AQ Java (oracle.AQ) Classes to Access AQ 3-6
Accessing Java AQ Classes 3-7
Advanced Queuing Examples 3-7
Managing the Java AQ API 3-8
Using Oracle Java Messaging Service to Access AQ 3-8
Standard JMS Features 3-8
Oracle JMS Extensions 3-9
Accessing Standard and Oracle JMS 3-9
For More Information 3-10
Using the AQ XML Servlet to Access AQ 3-10
Comparing AQ Programmatic Environments 3-12
AQ Administrative Interfaces 3-12
AQ Operational Interfaces 3-15

4 Managing AQ

Security 4-2
Administrator Role 4-2
User Role 4-2
Access to AQ Object Types 4-3
Oracle 8.1-Style Queues 4-3
Compatibility 4-3
Security 4-3
Privileges and Access Control 4-4
OCI Applications 4-5
Security Required for Propagation 4-5
Queue Table Export-Import 4-5
Exporting Queue Table Data 4-5
Importing Queue Table Data 4-7

Creating AQ Administrators and Users 4-7
Oracle Enterprise Manager Support 4-8
Using Advanced Queuing with XA 4-9
Restrictions on Queue Management 4-9
Collection Types in Message Payloads 4-10
Synonyms on Queue Tables and Queues 4-10
Tablespace Point-in-Time Recovery 4-10
Nonpersistent Queues 4-10
Propagation Issues 4-11
Execute Privileges Required for Propagation 4-11
The Number of Job Queue Processes 4-11
Optimizing Propagation 4-12
Propagation from Object Queues 4-13
Guidelines for Debugging AQ Propagation Problems 4-13
Oracle 8.0-Style Queues 4-14
Migrating To and From 8.0 4-15
Importing and Exporting with 8.0-Style Queues 4-16
Roles in 8.0 4-16
Security with 8.0-Style Queues 4-17
Access to AQ Object Types 4-17
OCI Application Access to 8.0-Style Queues 4-17
Pluggable Tablespaces and 8.0-Style Multiconsumer Queues 4-17
Autocommit Features in the DBMS_AQADM Package 4-17

5 Performance and Scalability

Performance Overview 5-2
Advanced Queuing in the Oracle Real Application Clusters Environment 5-2
Advanced Queuing in the Multi-threaded Server Environment 5-2
Basic Tuning Tips 5-2
Running Enqueue and Dequeue Processes Concurrently—Single Queue Table 5-2
Running Enqueue and Dequeue Processes in Serial—Single Queue Table 5-3
Propagation Tuning Tips 5-3

6 Frequently Asked Questions

General Questions 6-1

JMS Questions 6-6
Internet Access Questions 6-7
Oracle Internet Directory (OID) Questions - Global Agents, Global Events, and Global Queues 6-9
Transformation Questions 6-9
Performance Questions 6-10
Installation Questions 6-11

7 Modeling and Design

Modeling Queue Entities 7-2
Basic Queuing 7-3
Basic Queuing Illustrated 7-3
AQ Client-Server Communication 7-4
Multiple-Consumer Dequeuing of the Same Message 7-6
Dequeuing of Specified Messages by Specified Recipients 7-9
AQ Implementation of Workflows 7-11
AQ Implementation of Publish/Subscribe 7-12
Message Propagation 7-15

8 A Sample Application Using AQ

A Sample Application 8-2
General Features of Advanced Queuing 8-2
System-Level Access Control 8-2
Message Format Transformation 8-4
Structured Payloads 8-10
Queue Payloads Containing XMLType Attributes 8-13
Queue-Level Access Control 8-16
Nonpersistent Queues 8-17
Retention and Message History 8-28
Publish-Subscribe Support 8-29
Support for Oracle Real Application Clusters 8-31
Support for Statistics Views 8-35
Internet Access 8-36
Enqueue Features 8-36
Subscriptions and Recipient Lists 8-37

Priority and Ordering of Messages 8-39
Time Specification: Delay 8-46
Time Specification: Expiration 8-49
Message Grouping 8-52
Message Transformation During Enqueue 8-55
Enqueue Using the AQ XML Servlet 8-56
Dequeue Features 8-58
Dequeue Methods 8-59
Multiple Recipients 8-63
Local and Remote Recipients 8-64
Message Navigation in Dequeue 8-66
Modes of Dequeuing 8-70
Optimization of Waiting for Arrival of Messages 8-75
Retry with Delay Interval 8-77
Exception Handling 8-81
Rule-Based Subscription 8-86
Listen Capability 8-90
Message Transformation During Dequeue 8-96
Dequeue Using the AQ XML Servlet 8-97
Asynchronous Notifications 8-97
Registering for Notifications Using the AQ XML Servlet 8-106
Propagation Features 8-107
Propagation 8-107
Propagation Scheduling 8-108
Propagation of Messages with LOB Attributes 8-111
Enhanced Propagation Scheduling Capabilities 8-114
Exception Handling During Propagation 8-117
Message Format Transformation During Propagation 8-118
Propagation Using HTTP 8-119

9 Administrative Interface

Use Case Model: Administrative Interface — Basic Operations 9-2
Creating a Queue Table 9-6
Purpose 9-7
Usage Notes 9-7

Syntax 9-9
Examples 9-9
PL/SQL (DBMS_AQADM Package): Creating a Queue Table 9-10
VB (OO4O): Creating a Queue Table 9-12
Java (JDBC): Creating a Queue Table 9-12
Creating a Queue Table [Set Storage Clause] 9-15
Altering a Queue Table 9-16
Purpose 9-16
Usage Notes 9-16
Syntax 9-17
Examples 9-17
PL/SQL (DBMS_AQADM Package): Altering a Queue Table 9-17
Java (JDBC): Altering a Queue Table 9-18
Dropping a Queue Table 9-19
Purpose 9-20
Usage Notes 9-20
Syntax 9-20
Examples 9-20
PL/SQL (DBMS_AQADM Package): Dropping a Queue Table 9-20
Java (JDBC): Dropping a Queue Table 9-21
Creating a Queue 9-22
Purpose 9-23
Usage Notes 9-23
Syntax 9-24
Examples 9-24
PL/SQL (DBMS_AQADM): Creating a Queue 9-24
Java (JDBC): Creating a Queue 9-26
Creating a Nonpersistent Queue 9-28
Purpose 9-28
Usage Notes 9-28
Syntax 9-29
Examples 9-29
PL/SQL (DBMS_AQADM): Creating a Nonpersistent Queue 9-29
Java (JDBC): Creating a Nonpersistent Queue 9-30
Altering a Queue 9-31

Purpose 9-32
Usage Notes 9-32
Syntax 9-32
Examples 9-32
PL/SQL (DBMS_AQADM): Altering a Queue 9-33
Java (JDBC): Altering a Queue 9-33
Dropping a Queue 9-34
Purpose 9-35
Usage Notes 9-35
Syntax 9-35
Examples 9-35
PL/SQL (DBMS_AQADM): Dropping a Queue 9-36
Java (JDBC): Dropping a Queue 9-36
Creating a Transformation 9-37
Purpose 9-37
Usage Notes 9-38
Syntax 9-38
Examples 9-38
PL/SQL (DBMS_AQADM): Creating a Transformation 9-38
Java (JDBC) 9-39
Modifying a Transformation 9-40
Purpose 9-40
Usage Notes 9-41
Applying a Transformation 9-42
Dropping a Transformation 9-43
Purpose 9-43
Usage Notes 9-43
Starting a Queue 9-44
Purpose 9-44
Usage Notes 9-45
Syntax 9-45
Examples 9-45
PL/SQL (DBMS_AQADM Package): Starting a Queue 9-45
Java (JDBC): Starting a Queue 9-46
Stopping a Queue 9-47

Purpose 9-48
Usage Notes 9-48
Syntax 9-48
Examples 9-48
PL/SQL (DBMS_AQADM): Stopping a Queue 9-48
Java (JDBC): Stopping a Queue 9-49
Granting System Privilege 9-50
Purpose 9-51
Usage Notes 9-51
Examples 9-51
PL/SQL (DBMS_AQADM): Granting System Privilege 9-52
Java (JDBC): Granting System Privilege 9-52
Revoking System Privilege 9-53
Purpose 9-53
Usage Notes 9-54
Syntax 9-54
Examples 9-54
Using PL/SQL (DBMS_AQADM): Revoking System Privilege 9-54
Granting Queue Privilege 9-55
Purpose 9-55
Usage Notes 9-56
Syntax 9-56
Examples 9-56
PL/SQL (DBMS_AQADM): Granting Queue Privilege 9-56
Java (JDBC): Granting Queue Privilege 9-56
Revoking Queue Privilege 9-58
Purpose 9-59
Usage Notes 9-59
Syntax 9-59
Examples 9-59
PL/SQL (DBMS_AQADM): Revoking Queue Privilege 9-59
Java (JDBC): Revoking Queue Privilege 9-60
Adding a Subscriber 9-61
Purpose 9-62
Usage Note 9-62

Syntax 9-62
Examples 9-63
PL/SQL (DBMS_AQADM): Adding Subscriber 9-63
PL/SQL (DBMS_AQADM): Adding a Rule-Based Subscriber 9-64
Java (JDBC): Adding a Subscriber 9-65
Altering a Subscriber 9-67
Purpose 9-68
Usage Notes 9-68
Syntax 9-68
Examples 9-68
PL/SQL (DBMS_AQADM): Altering Subscriber 9-69
Java (JDBC): Altering a Subscriber 9-70
Removing a Subscriber 9-71
Purpose 9-72
Usage Notes 9-72
Syntax 9-72
Examples 9-72
PL/SQL (DBMS_AQADM): Removing Subscriber 9-73
Java (JDBC): Removing a Subscriber 9-73
Scheduling a Queue Propagation 9-74
Purpose 9-75
Usage Notes 9-75
Syntax 9-75
Examples 9-75
PL/SQL (DBMS_AQADM): Scheduling a Queue Propagation 9-76
Java (JDBC): Scheduling a Queue Propagation 9-76
Unscheduling a Queue Propagation 9-78
Purpose 9-78
Usage Notes 9-78
Syntax 9-78
Examples 9-79
PL/SQL (DBMS_AQADM): Unscheduling a Propagation 9-79
Java (JDBC): Unscheduling a Queue propagation 9-79
Verifying a Queue Type 9-81
Purpose 9-81

Usage Notes 9-81
Syntax 9-82
Examples 9-82
PL/SQL (DBMS_AQADM): Verifying a Queue Type 9-82
Java (JDBC): Verifying a Queue type 9-83
Altering a Propagation Schedule 9-84
Purpose 9-85
Usage Notes 9-85
Syntax 9-85
Examples 9-85
PL/SQL (DBMS_AQADM): Altering a Propagation Schedule 9-86
Java (JDBC): Altering a Propagation Schedule 9-86
Enabling a Propagation Schedule 9-88
Purpose 9-88
Usage Notes 9-88
Syntax 9-88
Examples 9-89
PL/SQL (DBMS_AQADM): Enabling a Propagation 9-89
Java (JDBC): Enabling a Propagation Schedule 9-89
Disabling a Propagation Schedule 9-91
Purpose 9-91
Usage Notes 9-91
Syntax 9-91
Examples 9-92
PL/SQL (DBMS_AQADM): Disabling a Propagation 9-92
Java (JDBC): Disabling a Propagation Schedule 9-92
Creating an AQ Agent 9-94
Purpose 9-95
Usage Notes 9-95
Syntax 9-95
Examples 9-95
Altering an AQ Agent 9-97
Purpose 9-98
Usage Notes 9-98
Syntax 9-98

Examples 9-98
Dropping an AQ Agent 9-99
Purpose 9-99
Usage Notes 9-99
Syntax 9-99
Examples 9-100
Enabling Database Access 9-101
Purpose 9-101
Usage Notes 9-101
Syntax 9-102
Examples 9-102
Disabling Database Access 9-103
Purpose 9-103
Syntax 9-103
Examples 9-104
Adding an Alias to the LDAP Server 9-105
Purpose 9-105
Usage Notes 9-105
Syntax 9-106
Examples 9-106
Removing an Alias from the LDAP Server 9-107
Purpose 9-107
Usage Notes 9-107
Syntax 9-107
Examples 9-108

10 Administrative Interface: Views

Use Case Model: Administrative Interface — Views 10-2
Selecting All Queue Tables in Database 10-4
Selecting User Queue Tables 10-6
Selecting All Queues in Database 10-8
Selecting All Propagation Schedules 10-10
Selecting Queues for Which User Has Any Privilege 10-14
Selecting Queues for Which User Has Queue Privilege 10-16
Selecting Messages in Queue Table 10-18

Selecting Queue Tables in User Schema 10-22
Selecting Queues In User Schema 10-24
Selecting Propagation Schedules in User Schema 10-26
Selecting Queue Subscribers 10-30
Selecting Queue Subscribers and Their Rules 10-32
Selecting the Number of Messages in Different States for the Whole Database 10-34
Selecting the Number of Messages in Different States for Specific Instances 10-36
Selecting the AQ Agents Registered for Internet Access 10-38
Selecting User Transformations 10-39
Selecting User Transformation Functions 10-40
Selecting All Transformations 10-40
Selecting All Transformation Functions 10-41

11 Operational Interface: Basic Operations

Use Case Model: Operational Interface — Basic Operations 11-2
Enqueuing a Message 11-5
Purpose: 11-5
Usage Notes 11-6
Syntax 11-6
Examples 11-6
Enqueuing a Message [Specify Options] 11-7
Purpose 11-8
Usage Notes 11-8
Syntax 11-8
Examples 11-9
Enqueuing a Message [Specify Message Properties] 11-10
Purpose 11-11
Usage Notes 11-11
Syntax 11-11
Examples 11-12
Enqueuing a Message [Specify Message Properties [Specify Sender ID]] 11-13
Purpose 11-13
Usage Notes 11-13
Syntax 11-14
Examples 11-14

Enqueuing a Message [Add Payload] 11-15
Purpose 11-15
Usage Notes 11-15
Syntax 11-16
Examples 11-16
PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages 11-17
Java (JDBC): Enqueue a message (add payload) 11-19
Visual Basic (OO4O): Enqueue a message 11-22
Listening to One (Many) Queue(s) 11-24
Purpose 11-24
Usage Notes 11-24
Syntax 11-25
Examples 11-25
Listening to One (Many) Single-Consumer Queues 11-26
Usage Notes 11-26
Syntax 11-27
Examples 11-27
PL/SQL (DBMS_AQ Package): Listen to Queues 11-27
Java (JDBC): Listen to Queues 11-28
C (OCI): Listen to Single-Consumer Queue(s) 11-29
Listening to One (Many) Multi-Consumer Queue(s) 11-38
Usage Notes 11-39
Syntax 11-39
Examples 11-39
PL/SQL (DBMS_AQ Package): Listen to Queue(s) 11-40
C (OCI): Listen to Multi-Consumer Queue(s) 11-41
Dequeuing a Message 11-47
Purpose 11-47
Usage Notes 11-48
Syntax 11-49
Examples 11-49
Dequeuing a Message from a Single-Consumer Queue [SpecifyOptions] 11-50
Purpose 11-51
Usage Notes 11-51
Syntax 11-51

Examples 11-52
PL/SQL (DBMS_AQ Package): Dequeue of Object Type Messages 11-52
Java (JDBC): Dequeue a message from a single consumer queue (specify options) 11-52
Visual Basic (OO4O): Dequeue a message 11-53
Dequeuing a Message from a Multi-Consumer Queue [Specify Options] 11-55
Purpose: 11-56
Usage Notes 11-56
Syntax 11-56
Examples 11-56
Java (JDBC): Dequeue a message from a multi consumer queue (specify options) 11-57
Registering for Notification 11-58
Purpose 11-59
Usage Notes 11-59
Syntax 11-59
Examples 11-60
Registering for Notification [Specifying Subscription Name — Single-Consumer Queue] 11-61
Registering for Notification [Specifying Subscription Name — Multi-Consumer Queue] 11-62
Usage Notes 11-62
Syntax 11-63
Examples 11-63
C (OCI): Register for Notifications For Single-Consumer and Multi-Consumer Queries 11-63
Posting for Subscriber Notification 11-69
Purpose 11-70
Usage Notes 11-70
Syntax 11-70
Examples 11-71
PL/SQL (DBMS_AQ Package): Post of Object-Type Messages 11-71
Adding an Agent to the LDAP Server 11-73
Purpose 11-73
Usage notes 11-73
Syntax 11-74
Examples 11-74
Removing an Agent from the LDAP Server 11-75

Purpose 11-75
Usage notes 11-75
Syntax 11-75
Examples 11-76

12 Creating Applications Using JMS

A Sample Application Using JMS 12-2
General Features of JMS 12-2
JMS Connection and Session 12-3
JMS Destinations - Queue and Topic 12-10
System-Level Access Control in JMS 12-14
Destination-Level Access Control in JMS 12-15
Retention and Message History in JMS 12-16
Supporting Oracle Real Application Clusters in JMS 12-17
Supporting Statistics Views in JMS 12-19
Structured Payload / Message Types in JMS 12-19
Payload Used by JMS Examples 12-29
JMS Point-to-Point Model Features 12-35
Queues 12-36
Queue Sender 12-36
Queue Receiver 12-37
Queue Browser 12-39
JMS Publish-Subscribe Model Features 12-41
Topic 12-42
Durable Subscriber 12-43
Topic Publisher 12-46
Recipient Lists 12-48
TopicReceiver 12-49
Topic Browser 12-51
JMS Message Producer Features 12-53
Priority and Ordering of Messages 12-54
Time Specification - Delay 12-57
Time Specification - Expiration 12-58
Message Grouping 12-60
JMS Message Consumer Features 12-64

Receiving Messages 12-64
Message Navigation in Receive 12-67
Modes for Receiving Messages 12-70
Retry With Delay Interval 12-72
Asynchronously Receiving Message Using Message Listener 12-74
AQ Exception Handling 12-78
JMS Propagation 12-81
Remote Subscribers 12-82
Scheduling Propagation 12-86
Enhanced Propagation Scheduling Capabilities 12-88
Exception Handling During Propagation 12-90
Message Transformation with JMS 12-91
Defining Message Transformations 12-91
Sending Messages to a Destination Using a Transformation 12-93
Receiving Messages from a Destination Using a Transformation 12-94
Specifying Transformations for Topic Subscribers 12-95
Specifying Transformations for Remote Subscribers 12-96

13 JMS Administrative Interface: Basic Operations

Use Case Model: JMS Administrative Interface — Basic Operations 13-2
Use Case Model Diagram: JMS Administrative Interface — Basic Operations 13-4
Two Ways to Register a Queue/Topic Connection Factory through the Database 13-5
Registering through the Database with JDBC Connection Parameters 13-6
Purpose 13-6
Usage Notes 13-7
Syntax 13-7
Example 13-7
Registering through the Database with a JDBC URL 13-8
Purpose 13-8
Usage Notes 13-9
Syntax 13-9
Example 13-9
Two Ways to Register a Queue/Topic Connection Factory through LDAP 13-10
Registering through LDAP with JDBC Connection Parameters 13-11
Purpose 13-12

Usage Notes 13-12
Syntax 13-12
Example 13-12
Registering through LDAP with a JDBC URL 13-14
Purpose 13-14
Usage Notes 13-14
Syntax 13-15
Example 13-15
Unregister a Queue/Topic Connection Factory in LDAP through the Database 13-16
Purpose 13-16
Usage Notes 13-16
Syntax 13-16
Example 13-16
Unregister a Queue/Topic Connection Factory in LDAP through LDAP 13-18
Purpose 13-18
Usage Notes 13-18
Syntax 13-18
Example 13-19
Point-to-Point - Two Ways to Create a Queue Connection Factory 13-20
Getting a Queue Connection Factory with JDBC URL 13-21
Purpose 13-21
Usage Notes 13-21
Syntax 13-22
Example 13-22
Getting a Queue Connection Factory with JDBC Connection Parameters 13-23
Purpose 13-24
Usage Notes 13-24
Syntax 13-24
Example 13-24
Publish-Subscribe - Two Ways to Create a Topic Connection Factory 13-25
Getting a Topic Connection Factory with JDBC URL 13-26
Purpose 13-26
Usage Notes 13-26
Syntax 13-27
Example 13-27

Getting a Topic Connection Factory with JDBC Connection Parameters 13-28
Usage Note 13-29
Purpose 13-29
Syntax 13-29
Example 13-29
Getting a Queue/Topic Connection Factory in LDAP 13-30
Purpose 13-30
Example 13-30
Getting a Queue/Topic in LDAP 13-31
Purpose 13-31
Example 13-31
Creating a Queue Table 13-32
Purpose 13-32
Usage Notes 13-33
Syntax 13-33
Example 13-33
Creating a Queue Table [Specify Queue Table Property] 13-34
Purpose 13-34
Usage Notes 13-34
Syntax 13-34
Example 13-34
Getting a Queue Table 13-36
Purpose 13-36
Usage Notes 13-37
Syntax 13-37
Example 13-37
Specifying Destination Properties 13-38
Purpose 13-39
Usage Notes 13-39
Syntax 13-39
Example 13-39
Point-to-Point - Creating a Queue 13-40
Purpose 13-40
Usage Notes 13-41
Syntax 13-41

Example 13-41
Publish-Subscribe - Creating a Topic 13-42
Purpose 13-42
Usage Notes 13-43
Syntax 13-43
Example 13-43
Granting System Privileges 13-44
Purpose 13-44
Usage Notes 13-45
Syntax 13-45
Example 13-45
Revoking System Privileges 13-46
Purpose 13-46
Usage Notes 13-46
Syntax 13-47
Example 13-47
Publish-Subscribe - Granting Topic Privileges 13-48
Purpose 13-49
Usage Notes 13-49
Syntax 13-49
Example 13-49
Publish-Subscribe - Revoking Topic Privileges 13-50
Purpose 13-50
Usage Notes 13-51
Syntax 13-51
Example 13-51
Point-to-Point: Granting Queue Privileges 13-52
Purpose 13-53
Usage Notes 13-53
Syntax 13-53
Example 13-53
Point-to-Point: Revoking Queue Privileges 13-54
Purpose 13-55
Usage Notes 13-55
Syntax 13-55

Example 13-55
Starting a Destination 13-56
Purpose 13-56
Usage Notes 13-57
Syntax 13-57
Example 13-57
Stopping a Destination 13-58
Purpose 13-59
Usage Notes 13-59
Syntax 13-59
Example 13-59
Altering a Destination 13-60
Purpose 13-60
Usage Notes 13-60
Syntax 13-61
Example 13-61
Dropping a Destination 13-62
Purpose 13-62
Usage Notes 13-62
Syntax 13-62
Example 13-63
Scheduling a Propagation 13-64
Purpose 13-65
Usage Notes 13-65
Syntax 13-65
Example 13-65
Enabling a Propagation Schedule 13-66
Purpose 13-66
Usage Notes 13-66
Syntax 13-67
Example 13-67
Altering a Propagation Schedule 13-68
Purpose 13-69
Usage Notes 13-69
Syntax 13-69

Example 13-69
Disabling a Propagation Schedule 13-70
Purpose 13-70
Usage Notes 13-70
Syntax 13-71
Example 13-71
Unscheduling a Propagation 13-72
Purpose 13-72
Usage Notes 13-72
Syntax 13-73
Example 13-73

14 JMS Operational Interface: Basic Operations (Point-to-Point)

Use Case Model: Operational Interface — Basic Operations 14-2
Use Case Model Diagram: Operational Interface (Point-to-Point) 14-3
Four Ways of Creating a Queue Connection 14-3
Creating a Queue Connection with Username/Password 14-5
Purpose 14-5
Usage Notes 14-5
Syntax 14-6
Example 14-6
Creating a Queue Connection with Open JDBC Connection 14-6
Purpose 14-6
Usage Notes 14-7
Syntax 14-7
Example 14-7
Creating a Queue Connection with Default Connection Factory Parameters 14-8
Purpose 14-8
Usage Notes 14-8
Syntax 14-8
Creating a Queue Connection with an Open OracleOCIConnection Pool 14-9
Purpose 14-9
Usage notes 14-9
Syntax 14-9
Example 14-9

Creating a Queue Session 14-11
Purpose 14-11
Usage Notes 14-11
Syntax 14-12
Example 14-12
Creating a Queue Sender 14-13
Purpose 14-13
Usage Notes 14-13
Syntax 14-13
Two Ways of Sending Messages Using a Queue Sender 14-14
Sending a Message Using a Queue Sender with Default Send Options 14-15
Purpose 14-15
Usage Notes 14-16
Syntax 14-16
Example 14-16
Sending Messages Using a Queue Sender by Specifying Send Options 14-17
Purpose 14-18
Usage Notes 14-18
Syntax 14-18
Example 14-19
Two Ways of Creating a Queue Browser for JMS Message Queues 14-20
Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages 14-21
Purpose 14-21
Usage Notes 14-22
Syntax 14-22
Example 14-22
Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages, Locking Messages while Browsing 14-23
Purpose 14-23
Usage Notes 14-24
Syntax 14-24
Example 14-24
Two Ways of Creating a Queue Browser for Oracle Object Type (ADT) Messages Queues 14-25
Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages 14-26

Purpose 14-27
Usage Notes 14-27
Syntax 14-27
Example 14-27
Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages, Locking Messages While Browsing 14-28
Purpose 14-28
Usage Notes 14-28
Syntax 14-28
Example 14-28
Browsing Messages Using a Queue Browser 14-29
Purpose 14-29
Usage Notes 14-29
Syntax 14-29
Example 14-30
Two Ways of Creating a Queue Receiver 14-31
Creating a Queue Receiver for Queues of Standard JMS Type Messages 14-32
Purpose 14-32
Usage Notes 14-33
Syntax 14-33
Example 14-33
Creating a Queue Receiver for Queues of Oracle Object Type (ADT) Messages 14-34
Purpose 14-35
Usage Notes 14-35
Syntax 14-35
Example 14-35
Creating a Queue Connection with an Open OracleOCIConnection Pool 14-36
Purpose 14-36
Usage notes 14-36
Syntax 14-37
Example 14-37

15 JMS Operational Interface: Basic Operations (Publish-Subscribe)

Use Case Model: JMS Operational Interface — Basic Operations (Publish-Subscribe) 15-2

Use Case Model Diagram: Operational Interface — Basic Operations (Publish-Subscribe) 15-4
Four Ways of Creating a Topic Connection 15-4
Creating a Topic Connection with Username/Password 15-6
Purpose 15-6
Usage Notes 15-6
Syntax 15-7
Example 15-7
Creating a Topic Connection with Open JDBC Connection 15-8
Purpose 15-8
Usage Notes 15-8
Syntax 15-8
Example 15-9
Creating a Topic Connection with Default Connection Factory Parameters 15-10
Purpose 15-10
Usage Notes 15-10
Syntax 15-10
Creating a Topic Connection with an Open OracleOCIConnectionPool 15-11
Purpose 15-11
Usage notes 15-11
Syntax 15-11
Example 15-12
Creating a Topic Session 15-13
Purpose 15-13
Usage Notes 15-13
Syntax 15-14
Example 15-14
Creating a Topic Publisher 15-14
Purpose 15-14
Usage Notes 15-15
Syntax 15-15
Four Ways of Publishing Messages Using a Topic Publisher 15-16
Publishing a Message with Minimal Specification 15-17
Purpose 15-17
Usage Notes 15-18
Syntax 15-18

Example 15-18
Publishing a Message Specifying Correlation and Delay 15-20
Purpose 15-21
Usage Notes 15-21
Syntax 15-21
Example 15-21
Publishing a Message Specifying Priority and Time-To-Live 15-23
Purpose 15-24
Usage Notes 15-24
Syntax 15-24
Example 15-24
Publishing Messages Specifying a Recipient List Overriding Topic Subscribers 15-25
Purpose 15-26
Usage Notes 15-26
Syntax 15-26
Example 15-26
Two Ways of Creating a Durable Subscriber for a Topic of Standard JMS Type Messages 15-28
Creating a Durable Subscriber for a JMS Topic without Selector 15-29
Purpose 15-29
Usage Notes 15-29
Syntax 15-30
Example 15-30
Creating a Durable Subscriber for a JMS Topic with Selector 15-31
Purpose 15-32
Usage Notes 15-32
Syntax 15-33
Example 15-33
Two Ways of Creating a Durable Subscriber for a Topic of Oracle Object Type (ADT) Messages 15-34
Creating a Durable Subscriber for an ADT Topic without Selector 15-35
Purpose 15-35
Usage Notes 15-36
Syntax 15-36
Example 15-36
Creating a Durable Subscriber for an ADT Topic with Selector 15-37

Purpose 15-38
Usage Notes 15-38
Syntax 15-38
Example 15-39
Two Ways of Creating a Remote Subscriber 15-40
Creating a Remote Subscriber for Topics of JMS Messages 15-41
Purpose 15-41
Usage Notes 15-42
Syntax 15-42
Example 15-42
Creating a Remote Subscriber for Topics of Oracle Object Type (ADT) Messages 15-44
Purpose 15-45
Usage Notes 15-45
Syntax 15-46
Example 15-46
Two Ways of Unsubscribing a Durable Subscription 15-47
Unsubscribing a Durable Subscription for a Local Subscriber 15-48
Purpose 15-48
Usage Notes 15-49
Syntax 15-49
Example 15-49
Unsubscribing a Durable Subscription for a Remote Subscriber 15-50
Purpose 15-50
Usage Notes 15-50
Syntax 15-51
Example 15-51
Two Ways of Creating a Topic Receiver 15-52
Creating a Topic Receiver for a Topic of Standard JMS Type Messages 15-53
Purpose 15-54
Usage Notes 15-54
Syntax 15-54
Example 15-54
Creating a Topic Receiver for a Topic of Oracle Object Type (ADT) Messages 15-55
Purpose 15-56
Usage Notes 15-56

Syntax 15-56
Example 15-56
Two Ways of Creating a Topic Browser for JMS Message Queues 15-57
Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages 15-58
Purpose 15-58
Usage Notes 15-59
Syntax 15-59
Example 15-59
Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages, Locking Messages While Browsing 15-60
Purpose 15-60
Usage Notes 15-61
Syntax 15-61
Example 15-61
Two Ways of Creating a Topic Browser for Oracle Object Type (ADT) Message Topics . 15-62
Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages 15-63
Purpose 15-64
Usage Notes 15-64
Syntax 15-64
Example 15-64
Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages, Locking Messages While Browsing 15-66
Purpose 15-67
Usage Notes 15-67
Syntax 15-67
Example 15-67
Browsing Messages Using a Topic Browser 15-68
Purpose 15-68
Usage Notes 15-68
Syntax 15-68
Example 15-68

16 JMS Operational Interface: Basic Operations (Shared Interfaces)

Use Case Model: JMS Operational Interface — Basic Operations (Shared Interfaces) 16-2

Starting a JMS Connection 16-6
Purpose 16-6
Usage Notes 16-6
Syntax 16-7
Examples 16-7
Getting the JMS Connection from a Session 16-8
Purpose 16-8
Usage Notes 16-8
Syntax 16-8
Examples 16-9
Committing All Operations in a Transacted Session 16-9
Purpose 16-9
Usage Notes 16-9
Syntax 16-10
Examples 16-10
Rolling Back All Operations in a Transacted Session 16-10
Purpose 16-10
Usage Notes 16-11
Syntax 16-11
Examples 16-11
Getting the Underlying JDBC Connection from a JMS Session 16-12
Purpose 16-12
Usage Notes 16-12
Syntax 16-12
Examples 16-13
Getting the Underlying OracleOCIConnectionPool from a JMS Connection 16-13
Purpose 16-13
Usage Notes 16-13
Syntax 16-14
Examples 16-14
Creating a Bytes Message 16-15
Purpose 16-15
Usage Notes 16-15
Syntax 16-15
Examples 16-16

Creating a Map Message 16-16
Purpose 16-16
Usage Notes 16-16
Syntax 16-17
Examples 16-17
Creating a Stream Message 16-17
Purpose 16-17
Usage Notes 16-18
Syntax 16-18
Examples 16-18
Creating an Object Message 16-19
Purpose 16-19
Usage Notes 16-19
Syntax 16-20
Examples 16-20
Creating a Text Message 16-21
Purpose 16-21
Usage Notes 16-21
Syntax 16-22
Examples 16-22
Creating an ADT Message 16-23
Purpose 16-23
Usage Notes 16-23
Syntax 16-24
Examples 16-24
Specifying Message Correlation ID 16-25
Purpose 16-25
Usage Notes 16-25
Syntax 16-26
Examples 16-26
Specifying JMS Message Property 16-27
Usage Notes 16-28
Specifying Message Property as Boolean 16-29
Purpose 16-29
Usage Notes 16-29

Syntax 16-30
Examples 16-30
Specifying Message Property as String 16-31
Purpose 16-31
Usage Notes 16-31
Syntax 16-32
Examples 16-32
Specifying Message Property as Int 16-33
Purpose 16-33
Usage Notes 16-33
Syntax 16-34
Examples 16-34
Specifying Message Property as Double 16-35
Purpose 16-35
Usage Notes 16-35
Syntax 16-36
Examples 16-36
Specifying Message Property as Float 16-37
Purpose 16-37
Usage Notes 16-37
Syntax 16-38
Examples 16-38
Specifying Message Property as Byte 16-39
Usage Notes 16-39
Syntax 16-40
Examples 16-40
Specifying Message Property as Long 16-41
Purpose 16-41
Usage Notes 16-41
Syntax 16-42
Examples 16-42
Specifying Message Property as Short 16-43
Purpose 16-43
Usage Notes 16-43
Syntax 16-44

Examples 16-44
Specifying Message Property as Object 16-45
Purpose 16-45
Usage Notes 16-45
Syntax 16-46
Examples 16-46
Setting Default TimeToLive for All Messages Sent by a Message Producer 16-46
Purpose 16-47
Usage Notes 16-47
Syntax 16-47
Examples 16-47
Setting Default Priority for All Messages Sent by a Message Producer 16-48
Purpose 16-48
Usage Notes 16-48
Syntax 16-48
Examples 16-49
Creating an AQjms Agent 16-50
Purpose 16-50
Usage Notes 16-51
Syntax 16-51
Examples 16-51
Two Ways of Receiving a Message Synchronously Using a Message Consumer 16-52
Purpose 16-52
Usage Notes 16-52
Syntax 16-53
Examples 16-53
Receiving a Message Using a Message Consumer by Specifying Timeout 16-54
Purpose 16-54
Usage Notes 16-55
Syntax 16-55
Examples 16-55
Receiving a Message Using a Message Consumer Without Waiting 16-56
Purpose 16-56
Usage Notes 16-56
Syntax 16-56

Examples 16-57
Specifying the Navigation Mode for Receiving Messages 16-57
Purpose 16-58
Usage Notes 16-58
Syntax 16-58
Examples 16-58
Two Ways of Specifying a Message Listener to Receive a Message Asynchronously 16-60
Specifying a Message Listener at the Message Consumer 16-61
Purpose 16-61
Usage Notes 16-61
Syntax 16-62
Examples 16-62
Specifying a Message Listener at the Session 16-64
Purpose 16-64
Usage Notes 16-64
Syntax 16-65
Examples 16-65
Getting the Correlation ID of a Message 16-65
Purpose 16-65
Usage Notes 16-66
Syntax 16-66
Examples 16-66
Two Ways of Getting the Message ID of a Message 16-67
Getting the Message ID of a Message as Bytes 16-68
Purpose 16-68
Usage Notes 16-68
Syntax 16-68
Examples 16-69
Getting the Message ID of a Message as a String 16-69
Purpose 16-69
Usage Notes 16-69
Syntax 16-70
Examples 16-70
Getting the JMS Message Property 16-71
Getting the Message Property as a Boolean 16-73

Purpose 16-73
Usage Notes 16-73
Syntax 16-73
Examples 16-74
Getting the Message Property as a String 16-74
Purpose 16-74
Usage Notes 16-75
Syntax 16-75
Examples 16-75
Getting the Message Property as Int 16-76
Purpose 16-76
Usage Notes 16-76
Syntax 16-76
Examples 16-77
Getting the Message Property as Double 16-78
Purpose 16-78
Usage Notes 16-78
Syntax 16-78
Examples 16-79
Getting the Message Property as Float 16-79
Purpose 16-79
Usage Notes 16-80
Syntax 16-80
Examples 16-80
Getting the Message Property as Byte 16-81
Purpose 16-81
Usage Notes 16-81
Syntax 16-81
Examples 16-82
Getting the Message Property as Long 16-82
Purpose 16-82
Usage Notes 16-83
Syntax 16-83
Examples 16-83
Getting the Message Property as Short 16-84

Purpose 16-84
Usage Notes 16-84
Syntax 16-84
Examples 16-85
Getting the Message Property as Object 16-85
Purpose 16-85
Usage Notes 16-86
Syntax 16-86
Examples 16-86
Closing a Message Producer 16-87
Purpose 16-87
Usage Notes 16-87
Syntax 16-87
Examples 16-88
Closing a Message Consumer 16-88
Purpose 16-88
Usage Notes 16-88
Syntax 16-89
Examples 16-89
Stopping a JMS Connection 16-90
Purpose 16-90
Usage Notes 16-90
Syntax 16-90
Examples 16-91
Closing a JMS Session 16-91
Purpose 16-91
Usage Notes 16-91
Syntax 16-92
Examples 16-92
Closing a JMS Connection 16-93
Purpose 16-93
Usage Notes 16-93
Syntax 16-93
Examples 16-94
Getting the Error Code for the JMS Exception 16-94

Purpose 16-94
Usage Notes 16-94
Syntax 16-95
Examples 16-95
Getting the Error Number for the JMS Exception 16-95
Purpose 16-95
Usage Notes 16-96
Syntax 16-96
Examples 16-96
Getting the Error Message for the JMS Exception 16-97
Purpose 16-97
Usage Notes 16-97
Syntax 16-97
Examples 16-98
Getting the Exception Linked to the JMS Exception 16-98
Purpose 16-98
Usage Notes 16-98
Syntax 16-99
Examples 16-99
Printing the Stack Trace for the JMS Exception 16-99
Purpose 16-99
Usage Notes 16-100
Syntax 16-100
Examples 16-100
Setting the Exception Listener 16-101
Purpose 16-101
Usage Notes 16-101
Syntax 16-102
Examples 16-102
Getting the Exception Listener 16-103
Purpose 16-103
Usage Notes 16-103
Syntax 16-103
Examples 16-103
Setting the Ping Period for the Exception Listener 16-104

Purpose 16-104
Usage Notes 16-104
Syntax 16-105
Examples 16-105
Getting the Ping Period for the Exception Listener 16-105
Purpose 16-105
Usage Notes 16-106
Syntax 16-106
Examples 16-106

17 Internet Access to Advanced Queuing

Overview of Advanced Queuing Operations Over the Internet 17-2
The Internet Data Access Presentation (IDAP) 17-3
IDAP Message Structure 17-4
IDAP Method Invocation 17-5
AQ XML Documents 17-6
IDAP and AQ XML Schemas 17-33
The IDAP Schema 17-34
AQ XML Schema 17-35
Deploying the AQ XML Servlet 17-47
Creating the AQ XML Servlet Class 17-47
Compiling the AQ XML Servlet 17-48
User Authentication 17-49
User Authorization 17-50
Using an LDAP Server with an AQ XML Servlet 17-52
Setup for Receiving AQ XML Requests Using SMTP (Email) 17-53
Using HTTP to Access the AQ XML Servlet 17-56
User Sessions and Transactions 17-59
Using HTTP and HTTPS for Advanced Queuing Propagation 17-60
High-Level Architecture 17-60
Using SMTP to Access the AQ Servlet 17-63
Customizing the AQ Servlet 17-63
Setting the Connection Pool Size 17-64
Setting the Session Timeout 17-64
Setting the Style Sheet for All Responses from the Servlet 17-65

Callbacks Before and After AQ Operations 17-66

A Oracle Advanced Queuing by Example

Creating Queue Tables and Queues A-4
Creating a Queue Table and Queue of Object Type A-4
Creating a Queue Table and Queue of Raw Type A-5
Creating a Prioritized Message Queue Table and Queue A-5
Creating a Multiple-Consumer Queue Table and Queue A-5
Creating a Queue to Demonstrate Propagation A-6
Setting Up Java AQ Examples A-6
Creating an Java AQ Session A-7
Creating a Queue Table and Queue Using Java A-8
Creating a Queue and Start Enqueue / Dequeue Using Java A-9
Creating a Multi-Consumer Queue and Add Subscribers Using Java A-9
Enqueuing and Dequeuing Of Messages A-11
Enqueuing and Dequeuing of Object Type Messages Using PL / SQL A-11
Enqueuing and Dequeuing of Object Type Messages Using Pro*C / C++ A-12
Enqueuing and Dequeuing of Object Type Messages Using OCI A-14
Enqueuing and Dequeuing of Object Type Messages (CustomDatum interface) Using Java ... A-16
Enqueuing and Dequeuing of Object Type Messages (using SQLData interface) Using Java .. A-18
Enqueuing and Dequeuing of RAW Type Messages Using PL / SQL A-21
Enqueuing and Dequeuing of RAW Type Messages Using Pro*C / C++ A-22
Enqueuing and Dequeuing of RAW Type Messages Using OCI A-25
Enqueue of RAW Messages using Java A-26
Dequeue of Messages Using Java A-27
Dequeue of Messages in Browse Mode Using Java A-28
Enqueuing and Dequeuing of Messages by Priority Using PL / SQL A-30
Enqueue of Messages with Priority Using Java A-32
Dequeue of Messages after Preview by Criterion Using PL / SQL A-33
Enqueuing and Dequeuing of Messages with Time Delay and Expiration Using PL / SQL A-37
Enqueuing and Dequeuing of Messages by Correlation and Message ID Using Pro*C / C++ .. A-38
Enqueuing and Dequeuing of Messages by Correlation and Message ID Using OCI A-42

Enqueuing and Dequeuing of Messages to/from a Multiconsumer Queue Using PL/SQL..... A-44
Enqueuing and Dequeuing of Messages to/from a Multiconsumer Queue using OCI. A-47
Enqueuing and Dequeuing of Messages Using Message Grouping Using PL/SQL A-51
Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using PL/SQL A-53
Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using Java.. A-56
Propagation A-62
Enqueue of Messages for remote subscribers/recipients to a Multiconsumer Queue and Propagation Scheduling Using PL/SQL A-62
Managing Propagation From One Queue To Other Queues In The Same Database Using PL/SQL A-64
Manage Propagation From One Queue To Other Queues In Another Database Using PL/SQL A-64
Unscheduling Propagation Using PL/SQL A-65
Dropping AQ Objects A-66
Revoking Roles and Privileges A-67
Deploying AQ with XA A-68
AQ and Memory Usage A-72
Create_types.sql: Create Payload Types and Queues in Scott's Schema A-72
Enqueuing Messages (Free Memory After Every Call) Using OCI..... A-72
Enqueuing Messages (Reuse Memory) Using OCI..... A-76
Dequeuing Messages (Free Memory After Every Call) Using OCI A-80
Dequeuing Messages (Reuse Memory) Using OCI A-84

B Oracle JMS Interfaces, Classes and Exceptions

Oracle JMSClasses (part 1)..... B-6
Oracle JMS Classes (part 2)..... B-7
Oracle JMS Classes (part 3)..... B-8
Oracle JMS Classes (part 4) B-9
Oracle JMS Classes (part 5) B-10
Oracle JMS Classes (part 6)..... B-11
Oracle JMS Classes (part 6 continued)..... B-12
Oracle JMS Classes (part 7)..... B-13
Oracle JMS Classes (part 8)..... B-14

Oracle JMS Classes (part 9) B-15
Oracle JMS Classes (part 10) B-16
Oracle JMS Classes (part 10 continued) B-17
Interface - javax.jms.BytesMessage B-18
Interface - javax.jms.Connection B-19
Interface - javax.jms.ConnectionFactory B-20
Interface - javax.jms.ConnectionMetaData B-21
Interface - javax.jms.DeliveryMode B-22
Interface - javax.jms.Destination B-23
Interface - javax.jms.MapMessage B-24
Interface - javax.jms.Message B-25
Interface - javax.jms.MessageConsumer B-27
Interface - javax.jms.MessageListener B-28
Interface - javax.jms.MessageProducer B-29
Interface - javax.jms.ObjectMessage B-30
Interface - javax.jms.Queue B-31
Interface - javax.jms.QueueBrowser B-32
Interface - javax.jms.QueueConnection B-33
Interface - javax.jms.QueueConnectionFactory B-34
Interface - javax.jms.QueueReceiver B-35
Interface - javax.jms.QueueSender B-36
Interface - javax.jms.QueueSession B-37
Interface - javax.jms.Session B-38
Interface - javax.jms.StreamMessage B-39
Interface - javax.jms.TextMessage B-40
Interface - javax.jms.Topic B-41
Interface - javax.jms.TopicConnection B-42
Interface - javax.jms.TopicConnectionFactory B-43
Interface - javax.jms.TopicPublisher B-44
Interface - javax.jms.TopicSession B-45
Interface - javax.jms.TopicSubscriber B-46
Exception javax.jms.InvalidDestinationException B-47
Exception javax.jms.InvalidSelectorException B-48
Exception javax.jms.JMSException B-49
Exception javax.jms.MessageEOFException B-50

Exception javax.jms.MessageFormatException B-51
Exception javax.jms.MessageNotReadableException B-52
Exception javax.jms.MesageNotWriteableException B-53
Interface - oracle.jms.AdtMessage B-54
Interface - oracle.jms.AQjmsQueueReceiver B-55
Interface - oracle.jms.AQjmsQueueSender B-56
Interface - oracle.jms.AQjmsTopicPublisher B-57
Interface - oracle.jms.TopicReceiver B-58
Interface - oracle.jms.AQjmsTopicSubscriber B-59
Interface - oracle.jms.AQjmsTopicReceiver B-60
Class - oracle.jms.AQjmsAdtMessage B-61
Class - oracle.jms.AQjmsAgent B-62
Class - oracle.jms.AQjmsBytesMessage B-63
Class - oracle.jms.AQjmsConnection B-64
Interface - oracle.jms.AQjmsConnectionMetadata B-65
Class - oracle.jms.AQjmsConstants B-66
Interface - oracle.jms.AQjmsConsumer B-67
Class - oracle.jms.AQjmsDestination B-68
Class - oracle.jms.AQjmsDestinationProperty B-70
Class - oracle.jms.AQjmsFactory B-71
Class - oracle.jms.AQjmsMapMessage B-72
Class - oracle.jms.AQjmsMessage B-73
Class - oracle.jms.AQjmsObjectMessage B-74
Class - oracle.jms.AQjmsOracleDebug B-75
Class - oracle.jms.AQjmsProducer B-76
Class - oracle.jms.AQjmsQueueBrowser B-77
Class - AQjmsQueueConnectionFactory B-78
Class - oracle.jms.AQjmsSession B-79
Class - oracle.jms.AQjmsStreamMessage B-82
Class - oracle.jms.AQjmsTextMessage B-83
Class - oracle.jms.AQjmsTopicConnectionFactory B-84
Exception oracle.jms.AQjmsException B-85
Exception oracle.jms.AQjmsInvalidDestinationException B-86
Exception oracle.jms.AQjmsInvalidSelectorException B-87
Exception oracle.jms.AQjmsMessageEOFException B-88

Exception oracle.jms.AQjmsMessageFormatException B-89
Exception oracle.jms.AQjmsMessageNotReadableException B-90
Exception oracle.jms.AQjmsMesssageNotWriteableException B-91
Interface - oracle.AQ.AQQueueTable B-92
Class - oracle.AQ.AQQueueTableProperty B-93
Interface - oracle.jms.TopicBrowser B-95
Class - oracle.jms.AQjmsTopicBrowser B-96

C Scripts for Implementing 'BooksOnLine'

tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues & Subscribers C-2
tkaqdocd.sql: Examples of Administrative and Operational Interfaces C-16
tkaqdoce.sql: Operational Examples C-21
tkaqdocp.sql: Examples of Operational Interfaces C-22
tkaqdocc.sql: Clean-Up Script C-37

D JMS and AQ XML Servlet Error Messages

JMS Error Messages D-2
AQ XML Servlet Error Messages D-16

E Interpreting Unified Modeling Language Diagrams

Use Case Diagrams E-1
State Diagrams E-7
Links in Online Versions of this Document E-8

Index

Send Us Your Comments

Oracle9*i* Application Developer's Guide - Advanced Queuing, Release 1 (9.0.1)

Part No. A88890-02

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

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

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  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 (optionally) electronic mail address.

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 Advanced Queuing. This information applies to versions of the Oracle Server that run on all platforms, unless otherwise specified.

This preface discusses the following:

- Audience
- Organization
- Related Documentation
- Conventions
- Documentation Accessibility

Audience

The *Oracle9i Application Developer's Guide - Advanced Queuing* is intended for programmers developing new applications that use Oracle Advanced Queuing, as well as those who have already implemented this technology and now wish to take advantage of new features.

Organization

The *Oracle9i Application Developer's Guide - Advanced Queuing* contains the following chapters and appendices:

Chapter 1, "Introduction to Oracle Advanced Queuing"

This chapter describes the requirements for optimal messaging systems. Although Oracle AQ is a relatively new technology, and not all these goals have been realized, you can get an overview of the design and a clear idea of the intended direction.

Chapter 2, "Basic Components"

This chapter describes features already present in Oracle AQ under three headings: General Features, Enqueue Features, and Dequeue Features.

Chapter 3, "AQ Programmatic Environments"

This chapter describes the elements you need to work with and issues you will want to take into consideration in preparing your AQ application environment.

Chapter 4, "Managing AQ"

This chapter discusses issues related to managing Advanced Queuing such as migrating queue tables (import-export), security, enterprise manager support, protocols, sample dba actions as preparation for working with AQ, and current restrictions.

Chapter 5, "Performance and Scalability"

This chapter discusses performance and scalability issues.

Chapter 6, "Frequently Asked Questions"

Frequently asked questions are answered here.

Chapter 7, "Modeling and Design"

This chapter covers the fundamentals of Advanced Queueing modeling anddesign.

Chapter 8, "A Sample Application Using AQ"

This chapter considers the features of Oracle Advanced Queuing in the context of a sample application.

Chapter 9, "Administrative Interface"

This chapter describes the administrative interface to Oracle Advanced Queuing.

Chapter 10, "Administrative Interface: Views"

In this chapter we describe the administrative interface with respect to views in terms of a hybrid of use cases and state diagrams.

Chapter 11, "Operational Interface: Basic Operations"

In this chapter we describe the operational interface to Oracle Advanced Queuing in terms of use cases.

Chapter 12, "Creating Applications Using JMS"

In this chapter we consider the features of the Oracle JMS interface to AQ in the context of a sample application based on that scenario.

Chapter 13, "JMS Administrative Interface: Basic Operations"

In this chapter we describe the administrative interface to Oracle Advanced Queuing in terms of use cases.

Chapter 14, "JMS Operational Interface: Basic Operations (Point-to-Point)"

In this chapter we describe point-to-point operations.

Chapter 15, "JMS Operational Interface: Basic Operations (Publish-Subscribe)"

In this chapter we describe publish-subscribe operations.

Chapter 16, "JMS Operational Interface: Basic Operations (Shared Interfaces)"

In this chapter we describe shared interface operations.

Chapter 17, "Internet Access to Advanced Queuing"

In this chapter we describe how to perform AQ operations over the Internet by using the Internet Data Access Presentation (IDAP) and transmitting the message over the Internet using transport protocols such as HTTP or SMTP.

Appendix A, "Oracle Advanced Queuing by Example"

This appendix provides examples of operations using different programmatic environments.

Appendix B, "Oracle JMS Interfaces, Classes and Exceptions"

This appendix provides a list of Oracle JMS interfaces, classes, and exceptions.

Appendix C, "Scripts for Implementing 'BooksOnLine'"

This appendix contains scripts used in the BooksOnLine example.

Appendix D, "JMS and AQ XML Servlet Error Messages"

This appendix listserror messages.

Appendix E, "Interpreting Unified Modeling Language Diagrams"

This appendix provides a brief explanation of use case diagrams and UML notation.

Related Documentation

Use the *PL/SQL User's Guide and Reference* to learn PL/SQL and to get a complete description of this high-level programming language, which is Oracle Corporation's procedural extension to SQL.

The Oracle Call Interface (OCI) is described in:

- *Oracle Call Interface Programmer's Guide*

You can use the OCI to build third-generation language (3GL) applications that access the Oracle Server.

Oracle Corporation also provides the Pro* series of precompilers, which allow you to embed SQL and PL/SQL in your application programs. If you write 3GL application programs in Ada, C, C++, COBOL, or FORTRAN that incorporate embedded SQL, refer to the corresponding precompiler manual. For example, if you program in C or C++, refer to the *Pro*C/C++ Precompiler Programmer's Guide*.

For SQL information, see the *Oracle9i SQL Reference* and *Oracle9i Database Administrator's Guide*. For basic Oracle concepts, see *Oracle9i Database Concepts*.

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

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

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

`http://www.oraclebookshop.com/`
Other customers can contact their Oracle representative to purchase printed documentation.

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

`http://technet.oracle.com/membership/index.htm`

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

`http://technet.oracle.com/docs/index.htm`

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions 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.	The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid. When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders.	*Oracle9i Database Concepts* You can specify the *parallel_clause*. Run `U`*`old_release`*`.SQL` where *old_release* refers to the release you installed prior to upgrading.
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, user names, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Specify the `ROLLBACK_SEGMENTS` parameter. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables 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, user names and roles, program units, and parameter values.	Enter `sqlplus` to open SQL*Plus. 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.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	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.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
. . .	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS subquery;`
	■ That you can repeat a portion of the code	`SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Convention	Meaning	Example
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown.	`acctbal NUMBER(11,2);` `acct  CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*

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.	`SELECT last_name, employee_id FROM employees;` `sqlplus hr/hr`

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. 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/`

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.

What's New in Advanced Queuing?

This section describes the new Advanced Queuing features of Oracle9*i* and previous releases.

The following sections describe the new features in Oracle Advanced Queuing:

- Oracle9i New Features in Advanced Queuing
- Oracle8i New Features in Advanced Queuing

Oracle9*i* New Features in Advanced Queuing

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

- **Internet Integration**

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

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

  The HTTP and SMTP support in AQ is implemented by using the 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 AQ Java 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. Oracle9*i* supports HTTP and SMTP; other proprietary protocols can also be supported using the callout mechanism through transformations.

- **Advanced Queuing Security over the Internet**

  AQ functionality allows only authorized Internet users to perform AQ operations on AQ 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 AQ queues may not reside in the same schema as the connected database user. Advanced Queuing uses proxy authentication so that only authorized Internet users can perform AQ operations on AQ queues.

- **LDAP Integration**

OID Integration: To leverage LDAP as the single point for managing generic information, Advanced Queuing is integrated with the Oracle Internet Directory (OID) server. This addresses the following requirements:

- Global topics (queues): AQ queue information can be stored in an OID server. OID 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 Messaging 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: OID 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. 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 can 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 have to be transformed. There are various existing solutions to handle these transformations. 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 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.

- **AQ Management**

  You can use new and enhanced Oracle Enterprise Manager to manage Advanced Queuing, 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 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 per 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 will be 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 will be 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: 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. AQ Queue information can be stored and looked up from OID.
- 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 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 Advanced Queuing

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

- **Queue-level access control**
- **Nonpersistent queues**
- **Support for Oracle Parallel Server**
- **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 Messaging Service (JMS) API**
- **Separate storage of history management information**

1

Introduction to Oracle Advanced Queuing

In this chapter, Oracle Advanced Queuing (AQ) and the requirements for complex information handling in an integrated environment are discussed under the following topics:

- What Is Advanced Queuing?
- General Features of Advanced Queuing
- Enqueue Features
- Dequeue Features
- Propagation Features
- Elements of Advanced Queuing
- Java Messaging Service Terminology
- Demos

What Is Advanced Queuing?

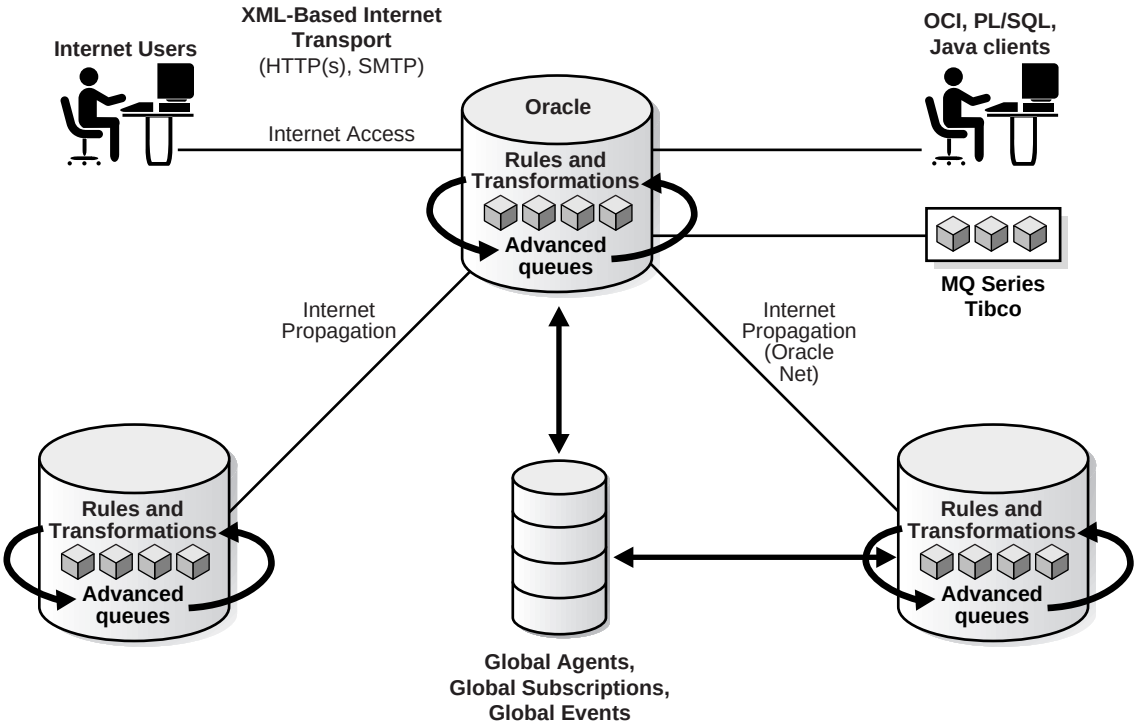
When Web-based business applications communicate with each other, producer applications enqueue messages and consumer applications dequeue messages. Advanced Queuing provides database-integrated message queuing functionality. Advanced Queuing leverages the functions of the Oracle database so that messages can be stored persistently, propagated between queues on different machines and databases, and transmitted using Oracle Net Services (formerly Net8), HTTP(S), and SMTP.

Since Oracle Advanced Queuing is implemented in database tables, all the operational benefits of high availability, scalability, and reliability are applicable to queue data. Standard database features such as recovery, restart, and security are supported in Advanced Queuing, and queue tables can be imported and exported. See Chapter 4, "Managing AQ" for more information. You can also use database development and management tools such as Oracle Enterprise Manager to monitor queues. See "Oracle Enterprise Manager Support" in Chapter 4.

Advanced Queuing in Integrated Application Environments

Advanced Queuing provides the message management functionality and asynchronous 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 (formerly Net8), messages are exchanged between a client and the Oracle database server or between two Oracle databases. Oracle Net Services also propagates messages from one Oracle queue to another. Or, as shown in Figure 1–1, you can perform Advanced Queuing operations over the Internet using transport protocols such as HTTP, HTTPS, or SMTP. In this case, the client, a user or Internet application, produces structured XML messages. During propagation over the Internet, Oracle servers communicate using structured XML also. See Chapter 17, "Internet Access to Advanced Queuing" for more information on Internet integration with Advanced Queuing.

Figure 1–1 An Integrated Application Environment Using Advanced Queuing



Interfaces to Advanced Queuing

You can access Advanced Queuing functionality through the following interfaces:

- PL/SQL using `DBMS_AQ`, `DBMS_AQADM`, and `DBMS_AQELM`. See the *Oracle9i Supplied PL/SQL Packages and Types Reference.*
- Visual Basic using Oracle Objects for OLE. See the Online Help for Oracle Objects for OLE.
- Java using the `oracle.AQ` Java package. See the *Oracle9i Supplied Java Packages Reference.*
- Java Messaging Service (JMS) using the `oracle.jms` Java package. See the *Oracle9i Supplied Java Packages Reference.*
- Internet access using HTTP, HTTPS, and SMTP

Queuing System Requirements

Advanced Queuing meets queuing system requirements for performance, scalability, and persistence. See Chapter 5, "Performance and Scalability" for more information.

Performance

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

- Number of messages enqueued / dequeued per 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. Advanced Queuing exhibits high performance as the number of programs using the application increases, as the number of messages increases, and as 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, machine, and application failures. Advanced Queuing 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, the application must be able to store the messages until they can be processed.
- Eternal 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 may be of higher priority than messages arriving earlier; messages arriving earlier may have to wait for messages arriving later before actions are executed; the same message may have to be accessed by different processes; and so on. Priorities also change. Messages in a specific queue can become more important, and so need to 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 a crucial for business and legal reasons. With the persistence features of Advanced Queuing, you can analyze periods of greatest demand or evaluate the lag between receiving and completing an order.

General Features of Advanced Queuing

The following general features are discussed:

- Point-to-Point and Publish-Subscribe Messaging
- Oracle Internet Directory
- Oracle Enterprise Manager Integration
- Message Format Transformation
- SQL Access
- Support for Statistics Views
- Structured Payloads
- Retention and Message History
- Tracking and Event Journals
- Queue-Level Access Control
- Nonpersistent Queues
- Support for Oracle9i Real Application Clusters
- XMLType Attributes in Object Types

- Internet Integration and IDAP

See Chapter 8, "A Sample Application Using AQ" for a hypothetical scenario in which the messaging system for a hypothetical online bookseller, BooksOnLine, is described. Many features discussed here are exemplified in the BooksOnLine example.

Point-to-Point and Publish-Subscribe Messaging

A combination of features allows publish-subscribe messaging between applications. These features include rule-based subscribers, message propagation, the listen feature, and notification capabilities.

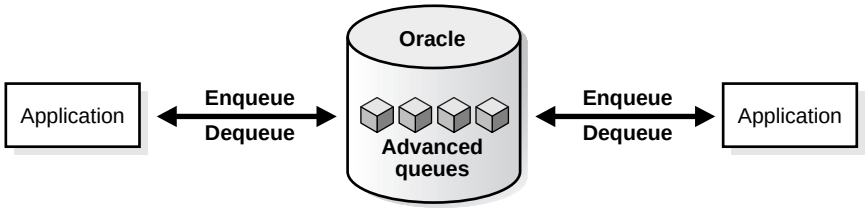
Advanced Queuing sends and receives messages in the following ways:

- Point-to-Point
- Publish-Subscribe

Point-to-Point

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 1–2 shows that each application has its own message queue, known as a **single consumer queue.**

Figure 1–2 Point-to-Point Messaging



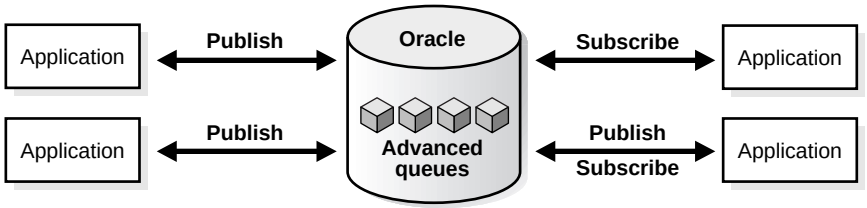
Publish-Subscribe

A publish-subscribe message can be consumed by multiple receivers, as shown in Figure 1–3. Publish-subscribe messaging has a wide dissemination mode—broadcast—and a more narrowly aimed mode—multicast, also called point-to-multipoint.

Broadcasting is the equivalent of 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 the same as 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 1–3 Publish-Subscribe Model



Oracle Internet Directory

Oracle Internet Directory is a native LDAPv3 directory service built on the Oracle database that centralizes a wide variety of information, including email 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. See the *Oracle Internet Directory Administrator's Guide* for more information.

Oracle Enterprise Manager Integration

You can use Enterprise Manager to create and manage queues, queue tables, and propagation schedules, to view all queue propagation schedules, queue errors, and the message queue, and other Advanced Queuing functions. See "Oracle Enterprise Manager Support" in Chapter 4.

Message Format Transformation

The message format transformation feature supports applications that 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.

A transformation can be specified as follows:

- During enqueue, to transform the message to the correct type before inserting it into the queue.

You can convert a message to the payload type of the queue at enqueue time. Thus, the type of the message to be enqueued need not match the payload type of the queue.

- During dequeue, to receive the message in the desired format

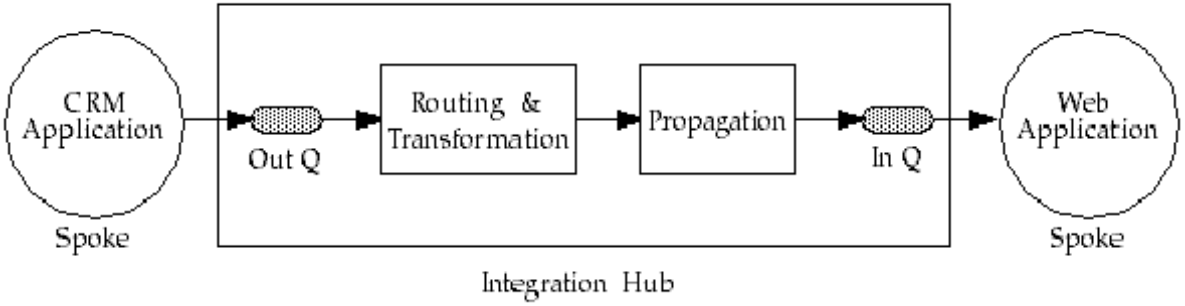
  A message can be transformed to the desired format before returning it to the dequeuer.

- By a remote subscriber, who can choose to receive a message in a format different from the format of the source queue

  Before propagating the message to the remote subscriber, the message is transformed according to the transformation that the remote subscriber specified when subscribing to the queue.

As Figure 1–4 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 1–4 Transformations in Application Integration



See "Message Format Transformation" on page 8-4 for more information.

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.

Support for Statistics Views

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

Structured Payloads

You can use object types to structure and manage message payloads. RDBMSs in general have a richer typing system than messaging systems. Since Oracle is an object-relational DBMS, it supports both traditional relational types as well as 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: Advanced Queuing 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 execute queries on the content of the message enables message warehousing.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Structured Payloads" on page 8-2.

Retention and Message History

The systems administrator specifies the retention duration to retain messages after consumption. Advanced Queuing 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.

To see this feature applied in the context of the `BooksOnLine` scenario, refer to Retention and Message History on page 8-28.

Tracking and Event Journals

If messages are retained, they can be related to each other. For example, if a message m2 is produced as a result of the consumption of message m1, 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. Advanced Queuing 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. Advanced Queuing 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 Advanced Queuing tracking features, you can use SQL SELECT and JOIN statements to get order information from AQ$QUETABLENAME 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
- Message content that can be used for tracking—USER_DATA

Queue-Level Access Control

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

To see this feature applied in the context of the BooksOnLine scenario, refer to Queue-Level Access Control on page 8-16.

Nonpersistent Queues

Advanced Queuing 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). Advanced Queuing supports persistent and nonpersistent messages with a common API.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Nonpersistent Queues" on page 8-17.

Support for Oracle9*i* Real Application Clusters

An application can specify the instance affinity for a queue table. When Advanced Queuing 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, the queue tables is 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 Advanced Queuing propagation jobs running in different instances. If compatibility is set to Oracle8*i*, release 8.1.5 or higher, an instance affinity (primary and secondary) can be specified for a queue table. When Advanced Queuing 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, the secondary instance or some available instance takes over the ownership for the queue table.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Support for Oracle Real Application Clusters" on page 8-31.

XMLType Attributes in Object Types

You can create queues that use Oracle object types containing attributes of 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 CLOBs
- 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 selectors

Internet Integration and IDAP

You can perform AQ operations over the Internet by using the Internet Data Access Presentation (IDAP), which defines message structure using XML. An IDAP-structured message is transmitted over the Internet using transport protocols such as HTTP or SMTP. See "Propagation over the Internet: HTTP and SMTP" on page 1-12 and Chapter 17, "Internet Access to Advanced Queuing" for more information.

Propagation over the Internet: HTTP and SMTP

Figure 1–5 shows the architecture for performing AQ operations over HTTP. The major components are:

- The AQ client program
- The Web server / ServletRunner hosting the AQ servlet
- The Oracle database server

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

Figure 1–5 Architecture for Performing AQ Operations Using HTTP

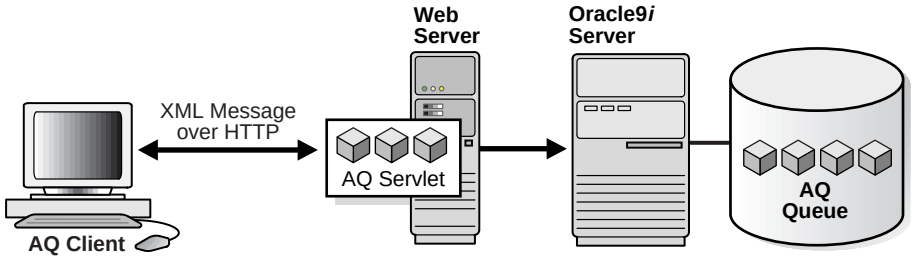
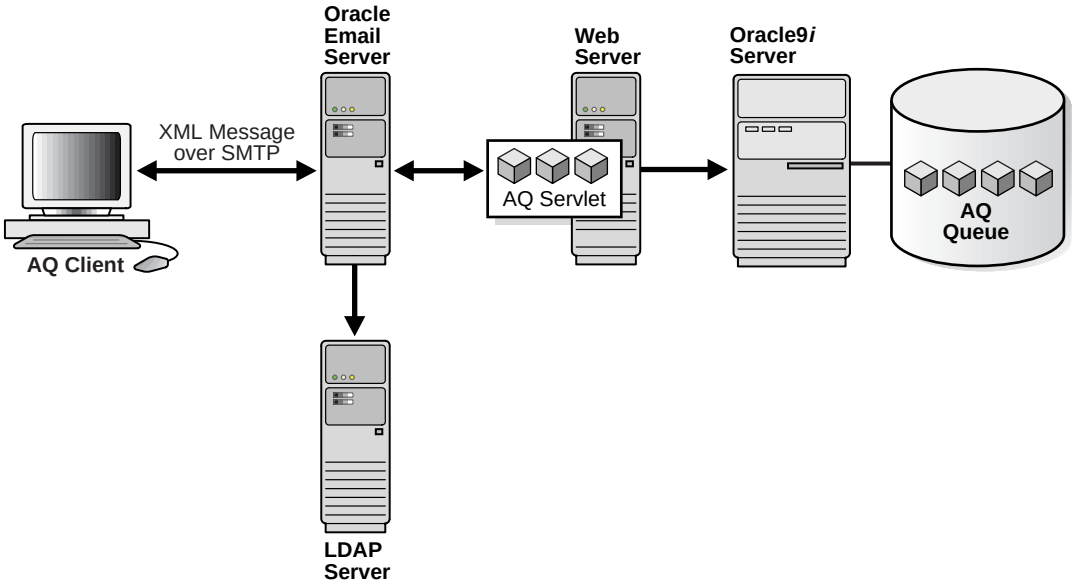


Figure 1–6 shows additional components in the architecture for sending AQ messages over SMTP:

- Email server
- LDAP server (Oracle Internet Directory)

The email server verifies client signatures using certificates stored in LDAP and then routes the request to the AQ servlet.

Figure 1–6 Architecture for Performing AQ Operations Using SMTP



The Internet Data Access Presentation (IDAP)

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

- All protocol tags are scoped to the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes.
- The receiver processes IDAP messages that have correct namespaces; for requests with incorrect namespaces, the receiver returns an invalid request error.
- The receiver processes IDAP messages without namespaces as though they had the correct namespaces, if the context is valid.
- The IDAP namespace has the value `http://ns.oracle.com/AQ/schemas/envelope`

- An XML document forming the request of an IDAP invocation does not require the use of an XML DTD or a schema.

For more information about IDAP, see Chapter 17, "Internet Access to Advanced Queuing"

Nonrepudiation and the AQ$<QueueTableName> View

Advanced Queuing maintains the entire history of information about a message along with the message itself. You can look up history information by using the `AQ$<QueueTableName>` view. This information serves as the proof of sending and receiving of messages and can be used for nonrepudiation of the sender and nonrepudiation of the receiver. See Chapter 10, "Administrative Interface: Views" for more information about the `AQ$<QueueTableName>` view.

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

- 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:

- 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.

Enqueue Features

The following features apply to enqueuing messages.

Correlation Identifiers

Users 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 may 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, 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 will be 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 may 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 may 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 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, 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, the name and address are not interpreted by the system and the message can be dequeued by a special consumer. To see this

feature applied in the context of the BooksOnLine scenario, refer to "Elements of Advanced Queuing" on page 1-21.

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 allows you to position a message in relation to other messages. Further, if several consumers act on the same queue, a consumer will get the first message that is available for immediate consumption. A message that is in the process of being consumed by another consumer will be skipped.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Priority and Ordering of Messages" on page 8-39.

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 have to 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.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Message Grouping" on page 8-52.

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 AQ to another, irrespective of whether the queues are local or remote. Propagation is done using database links and Oracle Net Services (formerly Net8).

To see this feature applied in the context of the BooksOnLine scenario, refer to "Propagation" on page 8-107.

Sender Identification

Applications can mark the messages they send with a custom identification. Oracle 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 messages 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 has to 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.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Rule-Based Subscription" on page 8-86.

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 does not have to be connected to the database to receive notifications regarding its subscriptions.

Clients can use the OCI function, `OCISubcriptionRegister`, or the PL/SQL procedure `DBMS_AQ.REGISTER` to register interest in messages in a queue (see "Registering for Notification" in Chapter 11, "Operational Interface: Basic Operations").

To see this feature applied in the context of the BooksOnLine scenario, refer to "Asynchronous Notifications" on page 8-77.

Dequeue Features

The following features apply to dequeuing messages.

Recipients

A message can be retrieved by multiple recipients without the need for multiple copies of the same message. To see this feature applied in the context of the BooksOnLine scenario, refer to "Multiple Recipients" on page 8-63.

Designated recipients can be located locally or at remote sites. To see this feature applied in the context of the BooksOnLine scenario, refer to "Local and Remote Recipients" on page 8-64.

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.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Message Navigation in Dequeue" on page 8-66.

Modes of Dequeuing

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

To see this feature applied in the context of the BooksOnLine scenario, refer to "Modes of Dequeuing" on page 8-70.

Optimization of Waiting for the Arrival of Messages

A DEQUEUE can be issued 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.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Optimization of Waiting for Arrival of Messages" on page 8-75.

Retries with Delays

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

To see this feature applied in the context of the BooksOnLine scenario, refer to "Retry with Delay Interval" on page 8-77.

Optional Transaction Protection

ENQUEUE and DEQUEUE requests are normally part of a transaction that contains the requests, thereby providing the desired transactional behavior. 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 as soon as the ENQUEUE or DEQUEUE statement is issued 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, the message will be moved to a user-specified exception queue.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Exception Handling" on page 8-81.

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, a dequeue must be used to retrieve the message.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Listen Capability" on page 8-90.

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. See "Internet Integration and IDAP" on page 1-12 for information on propagation over the Internet.

Automated Coordination of Enqueuing and Dequeuing

Recipients can be local or remote. Because Oracle does not support distributed object types, remote enqueuing or dequeuing using a standard database link does not work. However, you can use 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 AQ so that all messages enqueued in DROPBOX will be automatically propagated to another queue in database Y, regardless of whether database Y is local or remote. AQ will automatically check if the type of the remote queue in database Y is structurally equivalent to the type of the local queue in database X and propagate the message.

Recipients of propagated messages can be applications or queues. If the recipient is a queue, the actual recipients are determined by the subscription list associated with the recipient queue. If the queues are remote, messages are propagated using the specified database link. Only AQ-to-AQ message propagation is supported.

Propagation of Messages with LOBs

Propagation handles payloads with LOB attributes. To see this feature applied in the context of the BooksOnLine scenario, refer to "Propagation of Messages with LOB Attributes" on page 8-111.

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.

To see this feature applied in the context of the BooksOnLine scenario, refer to "Enhanced Propagation Scheduling Capabilities" on page 8-114.

Third-Party Support

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, the address of the recipient is not interpreted by AQ and so the message is not propagated by the 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.

Elements of Advanced Queuing

By integrating transaction processing with queuing technology, persistent messaging in the form of **Advanced Queuing** is possible. This section defines a number of Advanced Queuing terms.

Message

A message is the smallest unit of information inserted into and retrieved from a queue. A message consists of the following:

- Control information (metadata)
- Payload (data)

The control information represents message properties used by AQ to manage messages. The payload data is the information stored in the queue and is transparent to Oracle AQ. A message can reside in only one queue. A message is created by the enqueue call and consumed by the dequeue call.

Queue

A queue is a repository for messages. There are two types of queues: user queues, also known as normal queues, and exception queues. The user queue is for normal message processing. Messages are transferred to an exception queue if they cannot be retrieved and processed for some reason. Queues can be created, altered, started, stopped, and dropped by using the Oracle AQ administrative interfaces (see Chapter 9, "Administrative Interface" for more information.)

User queues can be persistent (the default) or nonpersistent queues. Persistent queues store messages in database tables. These queues provide all the reliability and availability features of database tables. 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.

Queue Table

Queues are stored in queue tables. Each queue table is a database table and contains one or more queues. Each queue table contains a default exception queue. Figure 7–1, "Basic Queues" on page 7-2 shows the relationship between messages, queues, and queue tables.

Agent

An agent is a queue user. This can be an end user or an application. There are two types of agents:

- Producers who place messages in a queue (enqueuing)
- Consumers who retrieve messages (dequeuing)

Any number of producers and consumers may be accessing the queue at a given time. Agents insert messages into a queue and retrieve messages from the queue by using the Oracle AQ operational interfaces (see Chapter 11, "Operational Interface: Basic Operations")

An agent is identified by its `name`, `address` and `protocol` (see "Agent Type (aq$_agent)" on page 2-3 in Chapter 2, "Basic Components" for formal description of this data structure).

- The name of the agent may be the name of the application or a name assigned by the application. A queue may itself be an agent—enqueuing or dequeuing from another queue.

- The address field is a character field of up to 1024 bytes that is interpreted in the context of the protocol. For instance, the default value for the protocol is 0, signifying a database link addressing. In this case, the address for this protocol is of the form

  ```
  queue_name@dblink
  ```

  where `queue_name` is of the form `[schema.]queue` and `dblink` may either be a fully qualified database link name or the database link name without the domain name.

Recipient

The recipient of a message may be specified by its name only, in which case the recipient must dequeue the message from the queue in which the message was enqueued. The recipient may be specified by 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 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`, 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, the name and address are not interpreted by the system and the message can be dequeued by a special consumer (see "Third-Party Support" on page 1-21).

Recipient and Subscription Lists

Multiple consumers can consume a single message:

- The enqueuer can explicitly specify the consumers who may retrieve the message as recipients of the message. A recipient is an agent identified by a name, address, and protocol.
- A queue administrator can specify a default list of recipients who can retrieve messages from a queue. The recipients specified in the default list are known as subscribers. If a message is enqueued without specifying the recipients, the message is sent to all the subscribers.

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.

Rule

A rule is used to define one or more subscribers' interest in subscribing to messages that conform to that rule. The messages that meet the rule criterion are delivered to the interested subscribers. A rule is specified as a boolean expression (one that evaluates to true or false) using syntax similar to the `WHERE` clause of a SQL query. The boolean expression can include conditions on the following:

- Message properties (currently priority and correlation identifier)
- User data properties (object payloads only)
- Functions (as specified in the `WHERE` clause of a SQL query)

Rule-Based Subscribers

A rule-based subscriber is a subscriber with associated rules in the default recipient list. If an associated rule evaluates to `TRUE` for a message, the message is sent to the rule-based subscriber even if the message has no specified recipients.

Transformation

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. 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 desired format. If specified with a remote subscriber, the message will be transformed before propagating it to the destination queue.

Queue Monitor

The queue monitor (QMNn) is a background process that monitors messages in queues. It provides the mechanism for message delay, expiration, and retry delay. The QMNn also performs garbage collection for the queue table and its indexes and index-organized tables (IOTs). For example, the QMNn determines when all subscribers of multiconsumer queues have received a message and subsequently removes the message from the queue table and supporting indexes and IOTs.

You can start a maximum of 10 multiple queue monitors at the same time. You start the queue monitors by setting the dynamic `init.ora` parameter `aq_tm_processes`. The queue monitor wakes up every minute, or whenever there is work to do, for instance, if a message is marked expired or ready to be processed.

Java Messaging Service Terminology

When using the `oracle.jms` Java package, keep in mind the following:

- The JMS equivalent of enqueue is **Send**.
- The destination of messages is a **Queue**, without any qualification.
- The container of messages is a **Topic**, with the idea being that each application can **Publish** on or **Subscribe** to a given topic.
- **Topic** in JMS maps to a **Multi-Consumer Queue** in the other AQ interfaces.
- The Java package `oracle.jms` extends the public JMS standard to allow for defined recipient lists.

Demos

The following demos can be found in the `$ORACLE_HOME/demo` directory:

Table 1–1

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 queue browser
`aqjmsdemo06.java`	Schedule propagation between queues in the database
`aqjmsdmo.sql`	Set up AQ JMS demos
`aqjmsREADME.txt`	Describe the 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 AQ java API demos
`newaqdemo00.sql`	Create users, message types, tables, etc.

Table 1–1

Demo and Locations	Topic
`newaqdemo01.sql`	Set up queue_tables, queues, subscribers and set up
`newaqdemo02.sql`	Enqueue messages
`newaqdemo03.sql`	Install dequeue procedures
`newaqdemo04.sql`	Perform blocking dequeue
`newaqdemo05.sql`	Perform listen for multiple agents
`newaqdemo06.sql`	Clean up users, queue_tables, queues, subscribers (cleanup script)
`ociaqdemo00.c`	Enqueue messages
`ociaqdemo01.c`	Perform blocking dequeue
`ociaqdemo02.c`	Perform listen for multiple agents

2

Basic Components

The following basic components are discussed in this chapter:

- Data Structures
- Enumerated Constants in the Administrative Interface
- Enumerated Constants in the Operational Interface
- INIT.ORA Parameter File Considerations

Data Structures

The following chapters discuss the Advanced Queuing administrative and operational interfaces in which data structures are used:

- Chapter 9, "Administrative Interface"
- Chapter 11, "Operational Interface: Basic Operations"

Object Name (object_name)

Purpose

To name 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 the *Oracle9i 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

To define queue types.

Syntax

```
type_name := VARCHAR2
type_name := <object_type> | "RAW"
```

Usage

Table 2–1 lists usage information for `type_name`.

Table 2–1 Type Name (type_name)

Parameter	Description
<object_types>	For details on creating object types please refer to *Oracle9i Database Concepts*. The maximum number of attributes in the object type is limited to 900.
"RAW"	To store payload of type RAW, 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 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.

Agent Type (aq$_agent)

Purpose

To identify 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.

Table 2–2 lists usage information for aq$_agent.

Table 2–2 Agent (aq$_agent)

Parameter	Description
`name` (`VARCHAR2(30)`)	Name of a producer or consumer of a message.The name must follow the reserved character guidelines in the Oracle9i SQL Reference.
`address` (`VARCHAR2(1024)`)	Protocol specific address of the recipient. If the protocol is 0 (default), 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

To identify the list of agents that will 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

To identify 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

To identify 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 Info List Type (aq$_reg_info_list)

Purpose

To identify the list of registrations to a queue.

Syntax

```
TYPE aq$_reg_info_list AS VARRAY(1024) OF sys.aq$_reg_info
```

AQ Post Info List Type (aq$_post_info_list)

Purpose

To identify 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 Info Type

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

Table 2–3 AQ Registration Info Type Attributes

Attribute	Description
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 queues.
namespace	Specifies the namespace of the subscription. To receive notifications from AQ queues the namespace must be DBMS_AQ.NAMESPACE_AQ. To receive notifications from other applications via DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.
callback	Specifies the action to be performed on message notification. For email notifications, the form is mailto://xyz@company.com For AQ PL/SQL Callback, use plsql://<schema>.<procedure>?PR=0 for raw message payload OR plsql://<schema>.<procedure>?PR=1 for ADT message payload converted to XML
context	Specifies the context that is to be passed to the callback function. Default: NULL

Table 2–4 shows the actions performed when different notification mechanisms/presentations are specified for nonpersistent queues.

Table 2–4 Nonpersistent Queues

Queue Payload Type	Presentation Specified					
	RAW			XML		
	Notification Mechanism			Notification Mechanism		
	OCI	Email	PL/SQL Callback	OCI	Email	PL/SQL Callback
RAW	The callback receives the RAW data in the payload.	Not supported	The PL/SQL callback receives the RAW data in the payload.	The callback receives the XML data in the payload.	The XML data is formatted as an IDAP message and emailed to the registered email address.	The PL/SQL callback receives the XML data in the payload.
ADT	Not supported.	Not supported.	Not supported.	The callback receives the XML data in the payload.	The XML data is formatted as an IDAP message and emailed to the registered email address.	The PL/SQL callback receives the XML data in the payload.

AQ Notification Descriptor Type

The `aq$_descriptor` data structure specifies the AQ Descriptor received by the 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

Table 2–5 AQ Notification Descriptor Type

Attribute	Description
queue_name	Name of the queue in which the message was enqueued which resulted in the notification.

Table 2–5 AQ Notification Descriptor Type

Attribute	Description
consumer_name	Name of the consumer for the multi-consumer queue
msg_id	Id of the message.
msg_prop	Message properties.

AQ Post Info Type

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

Table 2–6 AQ Post Info Type Attributes

Attribute	Description
name	name of the anonymous subscription to which you want to post to.
namespace	To receive notifications from other applications via DBMS_AQ.POST or OCISubscriptionPost(), the namespace must be DBMS_AQ.NAMESPACE_ANONYMOUS.
payload	The payload to be posted to the anonymous subscription Default: NULL

Enumerated Constants in the 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 2–7 lists the enumerated constants.

Table 2–7 Enumerated Constants in the 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 Operational Interface

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

```
DBMS_AQ.BROWSE
```

Table 2–8 lists the enumerated constants.

Table 2–8 Enumerated Constants in the 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

A parameter called AQ_TM_PROCESSES should be specified in the init.ora PARAMETER *file* if you want to perform time monitoring on queue messages. Use this for messages that have delay and expiration properties specified. This parameter can be set in a range from 0 to 10. Setting it to any other number will result in an error. If this parameter is set to 1, one queue monitor process (QMN) will be created as a background process. If the parameter is not specified, or is set to 0, the queue monitor process is not created.

Table 2–9 lists parameter information.

Table 2–9 AQ_TM_PROCESSES Parameter

Parameter	Options
Parameter Name	aq_tm_processes
Parameter Type	integer
Parameter Class	Dynamic
Allowable Values	0 to 10
Syntax	aq_tm_processes = <0 to 10>
Name of process	ora_qmn<n>_<oracle sid>
Example	aq_tm_processes = 1

JOB_QUEUE_PROCESSES Parameter

Propagation is handled by job queue (SNP) 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 1. The database administrator can set it to higher values if there are many queues from which the messages have to be propagated, or if there are many destinations to which the messages have to be propagated, or if there are other jobs in the job queue.

> **See Also:** *Oracle9i SQL Reference* for more information on JOB_QUEUE_PROCESSES.

The Java Advanced Queuing API supports both the administrative and operational features of Advanced Queuing. In developing Java programs for messaging applications, you will use JDBC to open a connection to the database and then use

`oracle.AQ`, the Java AQ API for message queuing. This means that you will no longer need to use PL/SQL interfaces.

3

AQ Programmatic Environments

This chapter describes the elements you need to work with and issues you will want to take into consideration in preparing your AQ application environment. The following topics are discussed:

- Programmatic Environments for Accessing AQ
- Using PL/SQL to Access AQ
- Using OCI to Access AQ
- Using Visual Basic (OO4O) to Access AQ
- Using AQ Java (oracle.AQ) Classes to Access AQ
- Using Oracle Java Messaging Service to Access AQ
- Using the AQ XML Servlet to Access AQ
- Comparing AQ Programmatic Environments

Programmatic Environments for Accessing AQ

The following programmatic environments are used to access the Advanced Queuing functions of Oracle:

- *Native AQ Interface*
  - *PL/SQL (DBMS_AQADM and DBMS_AQ packages):* supports administrative and operational functions
  - *C (OCI)*: supports operational functions
  - *Visual Basic (OO4O)*: supports operational functions
  - *Java (oracle.AQ package using JDBC)*: supports administrative and operational functions
- *JMS Interface to AQ*
  - Java (javax.jms and oracle.jms packages using JDBC): supports the standard JMS administrative and operational functions and Oracle JMS Extensions
- *XML Interface to AQ*
  - The AQ XML servlet supports operational functions using an XML message format.

For a comparison of the available functions for these programmatic environments, see the following tables:

- "Comparing AQ Programmatic Environments: Administrative Interfaces"
- "Comparing AQ Programmatic Environments: Operational Interfaces"

The AQ programmatic environments and their syntax references are listed in Table 3–1, "AQ Programmatic Environments".

Table 3–1 AQ Programmatic Environments

Language	Precompiler or Interface Program	Syntax Reference	In This Chapter See...
PL/SQL	DBMS_AQADM and DBMS_AQ Package	*Oracle9i Supplied PL/SQL Packages and Types Reference*	"Using PL/SQL to Access AQ" on page 3-3
C	Oracle Call Interface (OCI)	*Oracle Call Interface Programmer's Guide*	"Using OCI to Access AQ" on page 3-4)

Table 3–1 AQ Programmatic Environments

Language	Precompiler or Interface Program	Syntax Reference	In This Chapter See...
Visual Basic	Oracle Objects For OLE (OO4O)	Oracle Objects for OLE (OO4O) is a Windows-based product included with Oracle Client for Windows NT. There are no manuals for this product, only online help. Online help is available through the Application Development submenu of the Oracle installation.	"Using AQ Java (oracle.AQ) Classes to Access AQ" on page 3-6
Java (AQ)	oracle.AQ package via JDBC Application Programmatic Interface (API)	*Oracle9i Supplied Java Packages Reference*	"Using AQ Java (oracle.AQ) Classes to Access AQ" on page 3-6
Java (JMS)	oracle.JMS package via JDBC Application Programmatic Interface (API)	*Oracle9i Supplied Java Packages Reference*	"Using AQ Java (oracle.AQ) Classes to Access AQ" on page 3-6 and "Using Oracle Java Messaging Service to Access AQ" on page 3-8
AQ XML Servlet	oracle.AQ.xml.AQxml Servlet via HTTP or SMTP	*Oracle9i Supplied Java Packages Reference*	"Using the AQ XML Servlet to Access AQ" on page 3-10

Using PL/SQL to Access AQ

PL/SQL packages, DBMS_AQADM and DBMS_AQ support access to Oracle Advanced Queuing administrative and operational functions using the native AQ interface. These functions include the following:

- Create: queue, queue table, nonpersistent queue, multi-consumer queue/topic, RAW message, message with structured data
- Get: queue table, queue, multi-consumer queue/topic
- Alter: queue table, queue/topic
- Drop: queue/topic
- Start or stop: queue/topic
- Grant and revoke privileges

- Add, remove, alter subscriber
- Add, remove, alter AQ internet agents
- Grant or revoke privileges of database users to AQ internet agents
- Enable, disable, and alter propagation schedule
- Enqueue messages to single consumer queue (point-to-point model)
- Publish messages to multi-consumer queue / topic (publish-subscribe model)
- Subscribing for messages in multi-consumer 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 LDAP server
- Add or remove aliases to AQ objects in a LDAP server

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for detailed documentation, including parameters, parameter types, return values, examples, DBMS_AQADM and DBMS_AQ syntax.

Available PL / SQL `DBMS_AQADM` and `DBMS_AQ` functions are listed in detail in the following tables:

- Table 3–2, "Comparing AQ Programmatic Environments: Administrative Interfaces"
- Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"

Using OCI to Access AQ

Oracle Call Interface (OCI) provides an interface to Oracle Advanced Queuing functions using the native 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:** *Oracle Call Interface Programmer's Guide*: "OCI and Advanced Queuing" and "Publish-Subscribe Notification" sections, for syntax details.

For queues with user-defined payload type, OTT must be used to generate the OCI mapping for the Oracle type. The OCI client is responsible for freeing the memory of the AQ descriptors and the message payload.

Examples

OCI Interface

See Appendix A, "Oracle Advanced Queuing by Example", under "Enqueuing and Dequeuing Of Messages" on page A-11, for OCI Advanced Queuing interface examples.

Managing OCI Descriptor Memory

See Appendix A, "Oracle Advanced Queuing by Example", "AQ and Memory Usage" on page A-72, for examples illustrating memory management of OCI descriptors.

Using Visual Basic (OO4O) to Access AQ

Visual Basic (OO4O) supports access to Oracle Advanced Queuing operational functions using the native AQ interface.

These functions include the following:

- Create: connection, RAW message, message with structured data
- Enqueue messages to single consumer queue (point-to-point model)
- Publish messages to multi-consumer queue / topic (publish-subscribe model)
- Browse messages in a queue

- Receive messages from queue/topic
- Register to received messages asynchronously

Available Visual Basic (OO4O) functions are listed in detail in the following tables:

- Table 3–2, "Comparing AQ Programmatic Environments: Administrative Interfaces"
- Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"

For More Information

For more information about OO4O, refer to the following web site:

- http://technet.oracle.com

  Select Products > Internet Tools > Programmer. Scroll down to: Oracle Objects for OLE. At the bottom of the page is a list of useful articles for using the interfaces.

- http://www.oracle.com/products

  Search for articles on OO4O or Oracle Objects for OLE.

Using AQ Java (oracle.AQ) Classes to Access AQ

Java AQ API supports both the administrative and operational features of Oracle AQ (Advanced Queueing). In developing Java programs for messaging applications, you will use JDBC to open a connection to the database and then the oracle.AQ, the Java AQ API for message queuing. This means that you will no longer need to use PL/SQL interfaces.

Oracle9i Supplied Java Packages Reference describes the common interfaces and classes based on current PL/SQL interfaces.

- Common interfaces are prefixed with **"AQ"**. These interfaces will have different implementations in Oracle8*i* and Oracle Lite.
- In this document we describe the common interfaces and their corresponding Oracle8*i* implementations, that are in turn prefixed with "**AQOracle"**.

Accessing Java AQ Classes

The Java AQ classes are located in `$ORACLE_HOME/rdbms/jlib/aqapi.jar`. These classes can be used with any Oracle8*i* JDBC driver.

- **Using OCI8 or Thin JDBC Driver:** If your application uses the OCI8 or thin JDBC driver:
  - For JDK 1.2 you must include `$ORACLE_HOME/rdbms/jlib/aqapi.jar` in the `CLASSPATH`
  - For JDK 1.1 you must include `$ORACLE_HOME/rdbms/jlib/aqapi11.jar` in the `CLASSPATH.`
- **Using Oracle Server Driver in JServer:** If the application is using the Oracle Server driver and accessing the java AQ API from java stored procedures, the Java files are generally automatically pre-loaded in a Java-enabled database. If the Java files are not loaded, you must first load the `jmscommon.jar` and `aqapi.jar` files into the database using the "loadjava" utility.

Advanced Queuing Examples

Appendix A, "Oracle Advanced Queuing by Example" contains the following examples:

- Enqueue and Dequeue of Object Type Messages (CustomDatum interface) Using Java
- Enqueue and Dequeue of Object Type Messages (using SQLData interface) Using Java
- Create a Queue Table and Queue Using Java
- Create a Queue and Start Enqueue/Dequeue Using Java
- Create a Multi-Consumer Queue and Add Subscribers Using Java
- Enqueue of RAW Messages using Java
- Dequeue of Messages Using Java
- Dequeue of Messages in Browse Mode Using Java
- Enqueue of Messages with Priority Using Java
- Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using Java

Managing the Java AQ API

The various implementations of the Java AQ API are managed via an `AQDriverManager`. Both OLite and Oracle9*i* will have an `AQDriver` that is registered with the `AQDriverManager`. The driver manager is used to create an `AQSession` that can be used to perform messaging tasks.

The Oracle8*i* AQ driver is registered using the Class.forName ("oracle.AQ.AQOracleDriver") command.

When the `AQDriverManager.createAQSession`() method is invoked, it calls the appropriate `AQDriver` (amongst the registered drivers) depending on the parameter passed to the `createAQSession`() call.

The Oracle9*i* `AQDriver` expects a valid JDBC connection to be passed in as a parameter to create an AQSession. Users must have the execute privilege on the `DBMS_AQIN` package to use the AQ Java interfaces. Users can also acquire these rights through the `AQ_USER_ROLE` or the `AQ_ADMINSTRATOR_ROLE`. Users will also need the appropriate system and queue privileges for 8.1-style queue tables.

Using Oracle Java Messaging Service to Access AQ

Java Messaging Service (JMS): 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 Messaging Service: Oracle Java Messaging Service provides a Java API for Oracle Advanced Queuing based on the JMS standard. Oracle JMS supports the standard JMS interfaces and has extensions to support the AQ administrative operations and other AQ features that are not a part of the standard.

Standard JMS Features

Standard JMS features include:

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

Oracle JMS Extensions

Oracle JMS extensions include the following:

- Administrative API - to create queue tables, queues and topics
- Point-to-multipoint communication - using recipient lists for topics
- Message propagation between destinations. Allows the application to define remote subscribers.
- Supports transacted sessions that enable you to perform JMS as well as SQL operations in one atomic transaction.
- Message retention after messages have been dequeued
- Message delay - messages can be made visible after a certain delay
- Exception handling - messages are moved to exception queues if they cannot be processed successfully
- In addition to the standard JMS message types, Oracle supports `AdtMessages`. These are stored in the database as Oracle objects and hence 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 - 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 AQ for that subscriber).

Accessing Standard and Oracle JMS

Oracle JMS uses JDBC to connect to the database, hence it applications can run as follows:

- Outside the database using the OCI8 or thin JDBC driver
- Inside Oracle8*i* JServer using the Oracle Server driver

The standard JMS interfaces are in the **javax.jms** package.

The Oracle JMS interfaces are in the **oracle.jms** package.

- Using OCI8 or Thin JDBC Driver: To use JMS with clients running outside the database, you must include the appropriate JDBC driver, JNDI jar files and the following AQ jar files in your CLASSPATH:
  - For JDK 1.1 include the following:

```
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqapi11.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jdbc/lib/classes111.jar
```

- For JDK 1.2 include the following:

```
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqapi.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/jdbc/lib/classes12.jar
```

- Using Oracle Server Driver in JServer: If your application is running inside the JServer, 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, you may have to load `jmscommon.jar` followed by `aqapi.jar` using the **loadjava** utility.

Privileges

Users must have EXECUTE privilege on `DBMS_AQIN` and `DBMS_AQJMS` packages in order to use the Oracle JMS interfaces. Users can also acquire these rights through the `AQ_USER_ROLE` or the `AQ_ADMINSTRATOR_ROLE`.

Users will also need the appropriate system and queue or topic privileges to send or receive messages.

For More Information

Oracle JMS interfaces are described in detail in the *Oracle9i Supplied Java Packages Reference*,.

Using the AQ XML Servlet to Access AQ

You can use the AQ XML servlet to access Oracle9i AQ using open protocols like HTTP and SMTP and using an XML message format called Internet Data Access Presentation (IDAP).

Using the AQ servlet, a client can perform the following actions:

- Send messages to single-consumer queues

- Publish messages to multi-consumer 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 Javasoft's Servlet 2.0 or Servlet 2.2 interfaces.

- To deploy the AQ Servlet with a Web server that implements Javasoft's Servlet2.0 interfaces, you must define a class that extends the `oracle.AQ.xml.AQxmlServle20` class.
- To deploy the AQ Servlet with a Web server that implements Javasoft's Servlet2.2 interfaces, you must define a class that extends the `oracle.AQ.xml.AQxmlServlet` class.

The servlet can be compiled using JDK 1.1.x or JDK 1.2.x libraries.

- For JDK 1.1.x the CLASSPATH must contain:

```
$ORACLE_HOME/jdbc/lib/classes111.jar
$ORACLE_HOME/jlib/jta.jar
$ORACLE_HOME/jdbc/lib/nls_charset11.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/lib/lclasses11.zip
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/lib/xschema.jar
$ORACLE_HOME/rdbms/jlib/aqapi11.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqxml.jar
$ORACLE_HOME/rdbms/jlib/xsu111.jar
$ORACLE_HOME/lib/servlet.jar
```

- For JDK 1.2.x the CLASSPATH must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.jar
$ORACLE_HOME/jlib/jta.jar
$ORACLE_HOME/jdbc/lib/nls_charset12.jar
$ORACLE_HOME/jlib/jndi.jar
$ORACLE_HOME/lib/lclasses12.zip
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/lib/xschema.jar
$ORACLE_HOME/rdbms/jlib/aqapi.jar
```

```
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqxml.jar
$ORACLE_HOME/rdbms/jlib/xsu12.jar
$ORACLE_HOME/lib/servlet.jar
```

Since the servlet uses JDBC OCI drivers to connect to the Oracle9*i* database server, the 9i Oracle Client libraries must be installed on the machine that hosts the servlet. The `LD_LIBRARY_PATH` must contain `$ORACLE_HOME/lib`.

See Chapter 17, "Internet Access to Advanced Queuing" for more information on Internet access to Advanced Queuing.

Comparing AQ Programmatic Environments

Available functions for the AQ programmatic environments are listed by use case in the following tables:

- Table 3–2, "Comparing AQ Programmatic Environments: Administrative Interfaces"
- Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"

Each use case is described in detail, including examples, in Chapters 9 through 16.

AQ Administrative Interfaces

Table 3–2 lists the equivalent AQ administrative functions for the three programmatic environments, PL/SQL, Java (native AQ), and Java (JMS):

Table 3–2 Comparing AQ Programmatic Environments: Administrative Interfaces

Use Case	PL/SQL	Java (native AQ)	Java (JMS)
Create a Connection Factory	N/A	N/A	AQjmsFactory.getQueueConnectionFactory AQjmsFactory.getTopicConnectionFactory
Register a Connection Factory in a LDAP server	N/A	N/A	AQjmsFactory.registerConnectionFactory
Create a Queue Table	DBMS_AQADM.create_queue_table	Create AQQueueTableProperty, then AQSession.createQueueTable	AQjmsSession.createQueueTable

Table 3–2 Comparing AQ Programmatic Environments: Administrative Interfaces (Cont.)

Get a Queue Table	Use <schema>.<queue_table_name>	AQSession.getQueueTable	AQjmsSession.getQueueTable
Alter a Queue Table	DBMS_AQADM.alter_queue_table	AQQueueTable.alter	AQQueueTable.alter
Drop a Queue Table	DBMS_AQADM.drop_queue_table	AQQueueTable.drop	AQQueueTable.drop
Create a Queue	DBMS_AQADM.create_queue	AQSession.createQueue	AQjmsSession.createQueue
Get a Queue	Use <schema>.<queue_name>	AQSession.getQueue	AQjmsSession.getQueue
Create a Nonpersistent Queue	DBMS_AQADM.create_np_queue	Not supported	Not supported
Create a Multiconsumer Queue/Topic	DBMS_AQADM.create_queue in a queue table with multiple consumers enabled	AQSession.createQueue in a queue table with multiple consumers enabled	AQjmsSession.createTopic in a queue table with multiple consumers enabled
Get a Multiconsumer Queue/Topic	Use <schema>.<queue_name>	AQSession.getQueue	AQjmsSession.getTopic
Alter a Queue/Topic	DBMS_AQADM.alter_queue	AQQueue.alterQueue	AQjmsDestination.alter
Start a Queue/Topic	DBMS_AQADM.start_queue	AQQueue.start AQQueue.startEnqueue AQQueue.startDequeue	AQjmsDestination.start
Stop a Queue/Topic	DBMS_AQADM.stop_queue	AQQueue.stop AQQueue.stopEnqueue AQQueue.stopDequeue	AQjmsDestination.stop
Drop a Queue/Topic	DBMS_AQADM.drop_queue	AQQueue.drop AQQueueTable.dropQueue	AQjmsDestination.drop
Grant System Privileges	DBMS_AQADM.grant_system_privilege	Not supported	AQjmsSession.grantSystemPrivilege
Revoke System Privileges	DBMS_AQADM.revoke_system_privilege	Not supported	AQjmsSession.revokeSystemPrivilege

Table 3–2 Comparing AQ Programmatic Environments: Administrative Interfaces (Cont.)

Grant a Queue / Topic Privilege	DBMS_AQADM.grant_queue_privilege	AQQueue.grantQueuePrivilege	AQjmsDestination.grantQueuePrivilege AQjmsDestination.grantTopicPrivilege
Revoke a Queue / Topic Privilege	DBMS_AQADM.revoke_queue_privilege	AQQueue.revokeQueuePrivilege	AQjmsDestination.revokeQueuePrivilege AQjmsDestination.revokeTopicPrivilege
Verify a Queue Type	DBMS_AQADM.verify_queue_types	Not supported	Not supported
Add a Subscriber	DBMS_AQADM.add_subscriber	AQQueue.addSubscriber	See Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"
Alter a Subscriber	DBMS_AQADM.alter_subscriber	AQQueue.alterSubscriber	See Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"
Remove a Subscriber	DBMS_AQADM.remove_subscriber	AQQueue.removeSubscriber	See Table 3–3, "Comparing AQ Programmatic Environments: Operational Interfaces"
Schedule Propagation	DBMS_AQADM.schedule_propagation	AQQueue.schedulePropagation	AQjmsDestination.schedulePropagation
Enable a Propagation Schedule	DBMS_AQADM.enable_propagation_schedule	AQQueue.enablePropagationSchedule	AQjmsDestination.enablePropagationSchedule
Alter a Propagation Schedule	DBMS_AQADM.alter_propagation_schedule	AQQueue.alterPropagationSchedule	AQjmsDestination.alterPropagationSchedule
Disable a Propagation Schedule	DBMS_AQADM.disable_propagation_schedule	AQQueue.disablePropagationSchedule	AQjmsDestination.disablePropagationSchedule
Unschedule a Propagation	DBMS_AQADM.unschedule_propagation	AQQueue.unschedulePropagation	AQjmsDestination.unschedulePropagation
Create an AQ Internet Agent	DBMS_AQADM.create_aq_agent	not supported	not supported
Alter an AQ Internet Agent	DBMS_AQADM.alter_aq_agent	not supported	not supported

Table 3–2 Comparing AQ Programmatic Environments: Administrative Interfaces (Cont.)

Drop an AQ Internet Agent	DBMS_AQADM.drop_aq_agent	not supported	not supported
Grant Database User privileges to an AQ Internet Agent	DBMS_AQADM.enable_db_agent	not supported	not supported
Revoke Database User privileges from an AQ Internet Agent	DBMS_AQADM.disable_db_agent	not supported	not supported
Add alias for queue, agent, ConnectionFactory in a LDAP server	DBMS_AQADM.add_alias_to_ldap	not supported	not supported
Delete alias for queue, agent, ConnectionFactory in a LDAP server	DBMS_AQADM.del_alias_from_ldap	not supported	not supported

AQ Operational Interfaces

Table 3–3 lists equivalent AQ operational functions for the programmatic environments, PL/SQL, Java (native AQ), OCI, Visual Basic (OO4O), and Java (JMS):

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces

Use Cases	PL/SQL	Java (native AQ)	OCI	AQ XML Servlet	JMS
Create Connection, Session, Message					
Create a Connection	N/A	Create JDBC connection	OCIServerAttach	Open an HTTP connection after authenticating with the Web server	AQjmsQueueConnectionFactory.createQueueConnection AQjmsTopicConnectionFactory.createTopicConnection
Create a Session	N/A	AQDriverManager.createAQSession	OCISessionBegin	An HTTP servlet session is automatically started with the first IDAP request	QueueConnection.createQueueSession TopicConnecion.createTopicSession

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Create a RAW Message	Use SQL RAW type for message	AQQueue.createMessage Set AQRawPayload in message	Use OCIRaw for Message	Supply the hex representation of the message payload in the XML message. E.g.: `<raw>023f4523</raw>`	Not supported
Create a Message with Structured Data	Use SQL ADT type for message	AQQueue.createMessage Set AQObjectPayload in message	Use SQL ADT type for message	For ADT 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	N / A	QueueSession.createSender TopicSession.createPublisher
Enqueue Messages to a Single Consumer Queue: Point-to-Point Model					
Enqueue a Message to a single-consumer queue	DBMS_AQ.enqueue	AQQueue.enqueue	OCIAQEnq	`<AQXmlSend>`	QueueSender.send

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Enqueue a Message to a queue - specify visibility options	DBMS_AQ.enqueue Specify visibility in ENQUEUE_OPTIONS	AQQueue.enqueue Specify visibility in AQEnqueueOption	OCIAQEnq Specify OCI_ATTR_VISIBILITY in OCIAQEnqOptions	`<AQXmlSend>` Specify `<visibility>` in `<producer_options>`	Not supported
Enqueue a Message to a single-consumer queue - specify message properties - priority, expiration	DBMS_AQ.enqueue Specify priority, expiration in MESSAGE_PROPERTIES	AQQueue.enqueue Specify priority, expiration in AQMessageProperty	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 MessageProducer.setTimeToLive & MessageProducer.setPriority followed by QueueSender.send
Enqueue a Message to a single-consumer Queue - specify message properties - correlationID, delay, exception queue	DBMS_AQ.enqueue Specify correlation, delay, exception_queue in MESSAGE_PROPERTIES	AQQueue.enqueue Specify correlation, delay, exception queue in AQMessageProperty	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 - specify Message Properties (user-defined)	Not supported Properties should be part of payload	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>`, etc. in `<user_properties>`	Message.setIntProperty Message.setStringProperty Message.setBooleanProperty etc. followed by QueueSender.send

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Enqueue a Message to a single-consumer Queue - specify Message Transformation	DBMS_AQ.enqueue Specify transformation in ENQUEUE_OPTIONS	AQQueue.enqueue Specify transformation in AQDequeueOption	OCIAQEnq Specify OCI_ATTR_TRANSFORMATION in OCIAQEnqOptions	`<AQXmlSend>` Specify `<transformation>` in `<producer_options>`	AQjmsQueueSender.setTransformation followed by QueueSender.send
Publish Messages to a Multi-Consumer Queue/Topic - Publish-Subscriber Model					
Publish a Message to a Multi-consumer queue / Topic (using default subscription list)	DBMS_AQ.enqueue Set recipient_list to NULL in MESSAGE_PROPERTIES	AQQueue.enqueue Set recipient_list to NULL in AQMessageProperty	OCIAQEnq Set OCI_ATTR_RECIPIENT_LIST to NULL in OCIAQMsgProperties	`<AQXmlPublish>`	TopicPublisher.publish
Publish a Message to a Multi-consumer queue / Topic (using specific recipient list) See footnote-1	DBMS_AQ.enqueue Specify recipient list in MESSAGE_PROPERTIES	AQQueue.enqueue Specify recipient_list in AQMessageProperty	OCIAQEnq Specify OCI_ATTR_RECIPIENT_LIST in OCIAQMsgProperties	`<AQXmlPublish>` Specify `<recipient_list>` in `<message_header>`	AQjmsTopicPublisher.pubish Specify recipients as an array of AQjmsAgent
Publish a Message to a multi-consumer Queue / Topic - specify message properties - priority, expiration	DBMS_AQ.enqueue Specify priority, expiration in MESSAGE_PROPERTIES	AQQueue.enqueue Specify priority, expiration in AQMessageProperty	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 & MessageProducer.setPriority followed by TopicPublisher.publish

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Publish a Message to a multi-consumer queue/topic - specify send options - correlationID, delay, exception queue	DBMS_AQ.enqueue Specify correlation, delay, exception_queue in MESSAGE_PROPERTIES	AQQueue.enqueue Specify correlation, delay, exception queue in AQMessageProperty	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- specify Message Properties (user-defined)	Not supported Properties should be part of payload	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>`, etc. in `<user_properties>`	Message.setIntProperty Message.setStringProperty Message.setBooleanProperty etc. followed by TopicPublisher.publish
Publish a Message to a topic- specify Message Transformation	DBMS_AQ.enqueue Specify transformation in ENQUEUE_OPTIONS	AQQueue.enqueue Specify transformation in AQDequeueOption	OCIAQEnq Specify OCI_ATTR_TRANSFORMATION in OCIAQEnqOptions	`<AQXmlPublish>` Specify `<transformation>` in `<producer_options>`	AQjmsTopicPublisher.setTransformation followed by TopicPublisher.publish
Subscribing for Messages in a Multi Consumer Queue/Topic - Publish Subscribe Model					
Add a Subscriber	See administrative interfaces	See administrative interfaces	Not supported	Not supported	TopicSession.createDurableSubscriber AQjmsSession.createDurableSubscriber
Alter a Subscriber	See administrative interfaces	See administrative interfaces	Not supported	Not supported	TopicSession.createDurableSubscriber AQjmsSession.createDurableSubscriber using the new selector
Remove a Subscriber	See administrative interfaces	See administrative interfaces	Not supported	Not supported	AQjmsSession.unsubscriber

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Browse Messages In a Queue					
Browse messages in a Queue/Topic	DBMS_AQ.dequeue Set dequeue_mode to BROWSE in DEQUEUE_OPTIONS	AQQueue.dequeue Set dequeue_mode to BROWSE in AQDequeueOption	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 - locking messages while browsing	DBMS_AQ.dequeue Set dequeue_mode to LOCKED in DEQUEUE_OPTIONS	AQQueue.dequeue Set dequeue_mode to LOCKED in AQDequeueOption	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
Receive Messages From a Queue/Topic					
Start a connection for receiving messages	N/A	N/A	N/A	N/A	Connection.start
Create a Message Consumer	N/A	N/A	N/A	N/A	QueueSession.createQueueReceiver TopicSession.createDurableSubscriber AQjmsSession.createTopicReceiver
Dequeue a message from a queue/topic - specify visibility	DBMS_AQ.dequeue Specify visibility in DEQUEUE_OPTIONS	AQQueue.dequeue Specify visibility in AQDequeueOption	OCIAQDeq Specify OCI_ATTR_VISIBILITY in OCIAQDeqOptions	`<AQXmlReceive>` Specify `<visibility>` in `<consumer_options>`	Not supported

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Dequeue a message from a queue/topic - specify transformation	DBMS_AQ.dequeue Specify transformation in DEQUEUE_OPTIONS	DBMS_AQ.dequeue Specify transformation in AQDequeueOption	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 - specify navigation mode	DBMS_AQ.dequeue Specify navigation in DEQUEUE_OPTIONS	DBMS_AQ.dequeue Specify navigation in AQDequeueOption	OCIAQDeq Specify OCI_ATTR_NAVIGATION in OCIAQDeqOptions	`<AQXmlReceive>` Specify `<navigation>` in `<consumer_options>`	AQjmsQueueReceiver.setNavigationMode AQjmsTopicSubscriber.setNavigationMode AQjmsTopicReceiver.setNavigationMode
Dequeue a message from a single consumer queue	DBMS_AQ.dequeue Set dequeue_mode to REMOVE in DEQUEUE_OPTIONS	AQQueue.dequeue Set dequeue_mode to REMOVE in AQDequeueOption	OCIAQDeq Set OCI_ATTR_DEQ_MODE to REMOVE in OCIAQDeqOptions	`<AQXmlReceive>`	QueueReceiver.receive or QueueReceiver.receiveNoWait or AQjmsQueueReceiver.receiveNoData
Dequeue a message from a multi-consumer Queue/Topic (using subscription name)	DBMS_AQ.dequeue Set dequeue_mode to REMOVE and Set consumer_name to subscription name in DEQUEUE_OPTIONS	AQQueue.dequeue Set dequeue_mode to REMOVE and Set consumer_name to subscription name in AQDequeueOption	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

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

Dequeue a message from a multi-consumer Queue/Topic (using recipient name)	DBMS_AQ.dequeue Set dequeue_mode to REMOVE and Set consumer_name to recipient name in DEQUEUE_OPTIONS	AQQueue.dequeue Set dequeue_mode to REMOVE and Set consumer_name to recipient name inAQDequeueOption	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
Register to Receive Messages Asynchronously from a Queue/Topic					
Receive messages Asynchronously from a single-consumer queue	Define a PL/SQL callback procedure Register it using DBMS_AQ.register	Not supported	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 multi-consumer queue/Topic	Define a PL/SQL callback procedure Register it using DBMS_AQ.register	Not supported	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 TopicReceiver.setMessageListener

Table 3–3 Comparing AQ Programmatic Environments: Operational Interfaces (Cont.)

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	Not supported	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) multi-consumer queues/Topics	DBMS_AQ.listen Specify agent_name for all agents in agent_list	Not supported	OCIAQListen Specify agent_name for all agents in agent_list	Not supported	Create multiple TopicSubscribers or TopicReceivers on a TopicSession, then TopicSession.setMessageListener

4

Managing AQ

This chapter discusses the following topics related to managing Advanced Queuing:

- Security
- Oracle 8.1-Style Queues
- Queue Table Export-Import
- Oracle Enterprise Manager Support
- Oracle Enterprise Manager Support
- Using Advanced Queuing with XA
- Restrictions on Queue Management
- Propagation Issues
- Oracle 8.0-Style Queues

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 will be 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.

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
- Execute all procedures in DBMS_AQELM

User Role

You should avoid granting AQ_USER_ROLE in Oracle9*i* and 8.1 since this role will not provide sufficient privileges for enqueuing or dequeuing on Oracle9*i* or 8.1-compatible 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, provided that you wish 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 AQ Object Types

All internal AQ objects are now accessible to PUBLIC.

Oracle 8.1-Style Queues

Compatibility

For 8.1-style queues, the `compatible` parameter of `init.ora` and the `compatible` parameter of the queue table should be set to 8.1 to use the following features:

- Queue-level access control
- Nonpersistent queues (automatically created when queue table compatible = 8.1)
- Support for Oracle Parallel Server environments
- Rule-based subscribers for publish-subscribe
- Asynchronous notification
- Sender identification
- Separate storage of history management information

Security

AQ administrators of an Oracle9*i* database can create 8.1-style queues. All 8.1 security features are enabled for 8.1-style queues. Note that AQ 8.1 security features work only with 8.1-style queues. When you create queues, the default value of the `compatible` parameter in `DBMS_AQADM.CREATE_QUEUE_TABLE` is `8.1`.

Table 4–1 lists the AQ security features and privilege equivalences supported with 8.1-style queues.

Table 4–1 Security with 8.1-Style Queues

Privilege	8.1.x-Style Queues in a 8.1.x Database or Higher
AQ_USER_ROLE	Not supported. Equivalent privileges: ■ execute right on dbms_aq ■ enqueue any queue system privilege ■ dequeue any queue system privilege ■ execute right on dbms_transform
AQ_ADMINISTRATOR_ROLE	Supported.
Execute right on DBMS_AQ	Execute right on DBMS_AQ should be granted to all AQ users. To enqueue / dequeue on 8.1-compatible queues, the user needs the following privileges: ■ execute right on DBMS_AQ ■ enqueue / dequeue privileges on target queues, or ENQUEUE ANY QUEUE/DEQUEUE ANY QUEUE system privileges

Privileges and Access Control

You can grant or revoke privileges at the object level on 8.1- style queues. You can also grant or revoke various system-level privileges. The following table lists all common AQ operations and the privileges need to perform these operations for an Oracle9*i* or 8.1-compatible queue:

Table 4–2 Operations and Required Privileges

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.

Table 4–2 Operations and Required Privileges

Operation(s)	Privileges Required
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 AQ administrator using DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE.

OCI Applications

For an OCI application to access an 8.1-style queue, the session user has to be granted either the object privilege of the queue he intends to access or the ENQUEUE ANY QUEUE and/or DEQUEUE ANY QUEUE system privileges. The EXECUTE right of DBMS_AQ will not be checked against the session user's rights if the queue he intends to access is anOracle9*i* or 8.1-compatible queue.

Security Required for Propagation

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 have to 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 rights to enqueue to the destination queue. However, if the login user in the database link also owns the queue tables at the destination, no explicit AQ privileges need to be granted.

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.

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 done at queue-table granularity.

Exporting Queue Tables with Multiple Recipients

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

- A dequeue index-organized table (IOT)
- A time-management index-organized table
- A subscriber table (for 8.1-compatible queue tables)
- A history index-organized table (for 8.1-compatible 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 4-6.

Because the metadata tables contain rowids of some rows in the queue table, the import process will generate a note about the rowids being obsoleted when importing the metadata tables. This message can be ignored, since the queuing system will automatically correct 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), you should correct the problem and repeat the import.

Export Modes

Exporting operates in full database mode, user mode, and table mode, as follows. Incremental exports on queue tables are not supported.

- Full database mode—Queue tables, all related tables, system-level grants, and primary and secondary object grants are exported automatically.
- User mode—Queue tables, all related tables, and primary object grants are exported automatically.
- Table mode—This mode is not recommended. If you need to export a queue table in table mode, you must export all related objects that belong to that queue table. For example, when exporting an 8.1-compatible multiconsumer queue table `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.

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 queue tables)
- A history IOT (for 8.1-compatible queue tables)

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

Import IGNORE Parameter

You should not import queue data into a queue table that already contains data. The `IGNORE` parameter of the import utility should 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 will be loaded from the dump file into the existing table. At the same time, the old queue table definition and the old queue definition will be dropped and recreated. Hence, queue table and queue definitions prior to the import will be lost, and duplicate rows will appear in the queue table.

Creating AQ Administrators and Users

Creating a User as an AQ Administrator

To set a user up as an AQ administrator, do the following:

```
CONNECT system/manager
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT AQ_ADMINISTRATOR_ROLE TO aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
```

Additionally, you can grant execute privilege on the AQ packages as follows:

```
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT EXECUTE ON DBMS_AQ TO aqadm;
```

This allows the user to execute the procedures in the AQ packages from within a user procedure.

Creating Users AQUSER1 and AQUSER2

If you want to create AQ users who create and access queues within their own schemas, follow the steps outlined in "Creating a User as an AQ Administrator" except do not grant the `AQ_ADMINISTRATOR_ROLE`.

```
CONNECT system/manager
CREATE USER aquser1 IDENTIFIED BY aquser1;
GRANT CONNECT, RESOURCE TO aquser1;
```

Additionally, you can grant execute privilege on the AQ packages as follows:

```
GRANT EXECUTE ON DBMS_AQADM to aquser1;
GRANT EXECUTE ON DBMS_AQ TO aquser1;
```

If you wish to create an AQ user who does not create queues but uses a queue in another schema, first follow the steps outlined in the previous section. In addition, you must grant object level privileges. However, note that this applies only to queues defined using 8.1 compatible queue tables.

```
CONNECT system/manager
CREATE USER aquser2 IDENTIFIED BY aquser2;
GRANT CONNECT, RESOURCE TO aquser2;
```

Additionally, you can grant execute on the AQ packages as follows:

```
GRANT EXECUTE ON DBMS_AQADM to aquser2;
GRANT EXECUTE ON DBMS_AQ TO aquser2;
```

For `aquser2` to access the queue, `aquser1_q1` in `aquser1` schema, `aquser1` must execute the following statements:

```
CONNECT aquser1/aquser1
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
  'ENQUEUE','aquser1_q1','aquser2',FALSE);
```

Oracle Enterprise Manager Support

Oracle Enterprise Manager supports GUIs for most of the administrative functions listed in the administrative interfaces section. These include:

- Queues as part of the schema manager to view properties
- Create, start, stop, and drop queues
- Schedule and unschedule propagation
- Add and remove subscribers
- View propagation schedules for all queues in the database
- View errors for all queues in the database
- View the message queue
- Grant and revoke privileges

Using Advanced Queuing with XA

You must specify "Objects=T" in the xa_open string if you want to use the AQ OCI interface. This forces XA to initialize the client-side cache in Objects mode. You do not need to do this if you plan to use AQ through PL/SQL wrappers from OCI or Pro*C. The LOB memory management concepts from the Pro* documentation are not relevant for AQ raw messages because AQ provides a simple RAW buffer abstraction (although they are stored as LOBs).

When using the AQ navigation option, you must reset the dequeue position by using the `FIRST_MESSAGE` 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, you will get an error message stating that the navigation is used out of sequence (ORA-25237).

Restrictions on Queue Management

See the following topics for restrictions on queue management:

- Collection Types in Message Payloads
- Synonyms on Queue Tables and Queues
- Tablespace Point-in-Time Recovery
- Nonpersistent Queues

Note: Queue names and queue table names are converted to upper case. Mixed case (upper and lower case together) is not supported for queue names and queue table names.

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.

For example, the following operations are allowed:

```
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

All AQ PL/SQL calls do not resolve synonyms on queues and queue tables. Although you can create a synonyms, you should not apply the synonym to the AQ interface.

Tablespace Point-in-Time Recovery

AQ currently does not support tablespace point-in-time recovery. Creating a queue table in a tablespace will disable that particular tablespace for point-in-time recovery.

Nonpersistent Queues

Currently you can create nonpersistent queues of `RAW` and `ADT` 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 `OCISubcriptionRegister`.

Propagation Issues

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.

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 snapshot processes will 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 user name 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 nonpropagation-related jobs, 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, 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. Since 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 have to be propagated and the number of destinations (rather than queues) to which messages have to 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 behavior. 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 nonpropagation-related jobs 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 less than half the number of job queue processes, 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, multiple active schedules are assigned to an acquired job queue process.
- If the system is overloaded (all schedules are busy propagating), depending on availability, additional job queue processes will be acquired up to one less 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 will be 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 also has robust support for handling failures. It may not be able to propagate messages from a queue due to various types of failures. Some of the common reasons include failure of the database link, non-availability of the remote database, non-existence of the remote queue, remote queue not started and security violation while trying to enqueue messages into the remote queue. Under

all these circumstances the appropriate error messages will be reported in the DBA_QUEUE_SCHEDULES view. When an error occurs in a schedule, propagation of messages in that schedule is attempted periodically using an exponential backoff algorithm for a maximum of 16 times, after which the schedule is disabled. If the problem causing the error is fixed and the schedule is enabled, the error fields that indicate the last error date, time, and message will still continue to show the error information. These fields are reset only when messages are successfully propagated in that schedule. During the later stages of the exponential backoff, many hours or even days can elapse between propagation attempts. This happens when an error has been neglected for a long time. Under such circumstances it may be better to unschedule the propagation and schedule it again.

Propagation from Object Queues

Note that AQ does not support propagation from object queues that have BFILE or REF attributes in the payload.

Guidelines for Debugging AQ 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 will supply the actual name of the entity (without the brackets).

To begin debugging, do the following:

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

   Debugging information will be 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.

2. Check the database link to database 2.

   You can do this by doing `select count(*)` from @.

3. 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$` or `dbms_jobs` with that jobno.

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

5. Check for messages in the source queue with `select count(*) from` where `q_name = '<queue_name>';`
6. Check for messages in the destination queue with the same kind of `select`.
7. Check to see who is using job queue processes.

   Is it possible that the propagation job is being starved of processing time by other jobs?
8. Check to see that `sys.aq$_prop_table_` exists in `dba_queue_tables` and that queue `aq$_prop_notify_` exists in `dba_queues` (used for communication between job queue processes).
9. Check that the consumer attempting to dequeue a message from the destination queue is a recipient of the propagated messages.

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

```
select consumer_name, deq_txn_id, deq_time, deq_user_id,
  propagated_msgid from aq$
  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 t, table(t.history) h
where t.q_name = '<queue_name>';
```

   or

```
select consumer, transaction_id, deq_time, deq_user,
propagated_msgid from
the(select cast(history as sys.aq$_dequeue_history_t)
from  where q_name = '<queue_name>');
```

Oracle 8.0-Style Queues

If you use 8.0-style queues and 8.1 or higher database compatibility, the following features are not available:

- Support for Oracle Parallel Server environments
- Asynchronous notification

To use these features, you should migrate to 8.1-style or higher queues.

For more information, see:

- "Security Required for Propagation" on page 4-5
- *Oracle9i Database Migration*

Migrating To and From 8.0

To upgrade a 8.0-style queue table to an 8.1-style queue table or to downgrade a 8.1-style queue table to an 8.0-style queue table, use `DBMS_AQADM.MIGRATE_QUEUE_TABLE`. Table 4–3 lists the parameters for `DBMS_AQADM.MIGRATE_QUEUE_TABLE`.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE(
    queue_table        IN       VARCHAR2,
    compatible         IN       VARCHAR2)
```

Table 4–3 DBMS_AQADM_MIGRATE_QUEUE_TABLE Parameters

Parameter	Description
`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: Upgrading an 8.0 Queue Table to an 8.1-Compatible Queue Table

Note: You may need to set up the following data structures for certain examples to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table            => 'qtable1',
   multiple_consumers     => TRUE,
   queue_payload_type     => 'aq.message_typ',
   compatible             =>'8.0');
```

```
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 will generate a note about the rowids being obsoleted when importing the metadata tables. This message can be ignored, since the queuing system will automatically correct 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), you should correct the problem and repeat the import or export.

Roles in 8.0

Access to AQ operations in Oracle 8.0 is granted to users through roles that provide execution privileges on the AQ procedures. The fact that there is no control at the database object level when using Oracle 8.0 means that, in Oracle 8.0, a user with the `AQ_USER_ROLE` can enqueue and dequeue to any queue in the system. For finer-grained access control, use 8.1-style queue tables in an 8.1- compatible or higher database.

AQ administrators of an Oracle9*i* or 8.1 database can create queues with 8.0 compatibility; 8.0-style 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, the queue table must be converted to 8.1 style by running `DBMS_AQADM.MIGRATE_QUEUE_TABLE` on the queue table.

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information on `DBMS_AQADM.MIGRATE_QUEUE_TABLE`

If a database downgrade is necessary, all 8.1-style queue tables have to be either converted back to 8.0 compatibility or dropped before the database downgrade can be carried out. During the conversion, all Oracle9*i* or 8.1 security features on the queues, like the object privileges, will be dropped. When a queue is converted to 8.0 compatibility, the 8.0 security model applies to the queue, and only 8.0 security features are supported.

Security with 8.0-Style Queues

Table 4–4 lists the AQ security features and privilege equivalences supported with 8.0-style queues.

Table 4–4 Security with 8.0.x-Style Queues

Privilege	8.0.x-Style Queues in an 8.0.x Database	8.0.x Compatible Queues in a 8.1.x Database
AQ_USER_ROLE	Supported. The grantee is given the execute right of DBMS_AQ through the role.	Supported. The grantee is given the execute right of dbms_aq through the role.
AQ_ADMINISTRATOR_ROLE	Supported.	Supported.
Execute right on DBMS_AQ	Execute right on DBMS_AQ should be granted to developers who write AQ applications in PL/SQL.	Execute right on DBMS_AQ should be granted to developers who write AQ applications in PL/SQL.

Access to AQ Object Types

The procedure grant_type_access was made obsolete in release 8.1.5 for 8.0-style queues.

OCI Application Access to 8.0-Style Queues

For an OCI application to access an 8.0-style queue, the session user has to be granted the EXECUTE rights of DBMS_AQ.

Pluggable Tablespaces and 8.0-Style Multiconsumer Queues

A tablespace that contains 8.0-style multiconsumer queue tables should not be transported using the pluggable tablespace mechanism. The mechanism will work, however, with tablespaces that contain only single-consumer queues as well as 8.1 compatible multiconsumer queues. Before you can export a tablespace in pluggable mode, you have to alter the tablespace to read-only mode. If you try to import a read-only tablespace that contains 8.0-style multiconsumer queues, you will get an Oracle 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 purpose.

5

Performance and Scalability

This chapter discusses 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 IOTs. The code path of a dequeue operation is comparable to `SELECT, DELETE,` and `UPDATE` operations on similar tables.

Advanced Queuing in the Oracle Real Application Clusters Environment

Oracle Real Application Clusters can be used to ensure highly available access to queue data. The tail and the head of a queue can be extreme hot spots. Since Oracle Real Application Clusters may not scale well in the presence of hot spots, limit normal access to a queue from one instance only. If an instance failure occurs, messages managed by the failed instance can be processed immediately by one of the surviving instances.

Advanced Queuing in the Multi-threaded Server Environment

Queue operation scalability is similar to the underlying database operation scalability. If a dequeue operation with wait option is issued in a shared server environment, the shared server process will be 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, it is recommended that dequeue requests with wait option be issued via dedicated server processes. This restriction is not enforced.

Basic Tuning Tips

Advanced Queuing table layout should be considered similar to a layout with ordinary database tables and indexes.

> **See Also:** *Oracle9i Database Performance Guide and Reference* for tuning recommendations

Running Enqueue and Dequeue Processes Concurrently—Single Queue Table

Some environments need to process messages in a constant flow, thus requiring that both enqueue and dequeue processes run concurrently. If the message delivery system has only one queue table and one queue, all processes must work on the

same segment area at the same time, which impedes delivering a high number of messages at reasonable performance levels.

The best number for concurrent processes must be defined according to 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 optimal number of messages that should be delivered through the system has not been achieved, rather than increasing the number of processes, use several subscribers for load balancing.

Running Enqueue and Dequeue Processes in Serial—Single Queue Table

When enqueue and dequeue processes are not running concurrently, that is, messages are first enqueued and then dequeued, contention on the same data segment is lower than in the case of concurrent processes. In this case, the total time taken to deliver messages by the system 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. Normally, the dequeue operations throughput is much less than the enqueue operation (`INSERT`) throughput because dequeue operations perform `SELECT`, `DELETE`, and `UPDATE`.

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 determines the batching size for nontransactional queues, whereas for transactional queues, batch size is mainly determined by the user application.

6

Frequently Asked Questions

This section answers some of the most commonly asked questions about Advanced Queuing.

General Questions

How are messages that have been dequeued but are still retained in the queue table accessed?

Access them 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 "Selecting the Number of Messages in Different States for the Whole Database" on page 10-34).

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, 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 (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, 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.` It may also be affected by the value for `job_queue_interval.` Please refer to the MANAGING JOB QUEUES chapter of the Oracle9i Database Administrator's Guide for more information on job queues and SNP 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 via the `storage_clause` parameter in `DBMS_AQADM.CREATE_QUEUE_TABLE.` However, once you pick the tablespace, all IOTs and indexes created for that queue table will go to the specified tablespace. Currently, you do not have a choice to split them between different tablespaces.

How do you associate Oracle Parallel Server instance affinities with queue tables?

In 8.1 you can associate Oracle Parallel Server instance affinities with queue tables. If you are using `q1` and `q2` in different instances, 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")'

Note that 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 non-blocking and non-polling.

What is the use of non-persistent queues?

To provide a mechanism for notification to all users that are currently connected. The non-persistent queue mechanism supports the enqueue of a message to a non-persistent 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 AQ background process running for this to happen). You can dequeue these messages from the exception queue with a normal 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, 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 does not have to be connected to the database to receive notifications. The client specifies a callback function which will be 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 the other dequeue calls generate more undo which fills up the rollback segment and hence generates 1555.

The workaround is to use the `FIRST_MESSAGE` option to dequeue the message. This will re-execute the cursor and get 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.

What are the different subscriber types recorded on the subscriber table?

The `subscriber_types` and their values are:

1 - Current Subscriber. The subscribers name, address and protocol are in the same row.

2 - Ex subscriber - A subscriber that unsubscribed but had agent entries in the history `aq$_queuetable_h IOT.`

4 - Address - Used to store addresses of recipients. The name is always `NULL.` The address is always non-NULL.

8 - Proxy for Propagation - The name is always `NULL.`

database proxy to local queues, address=`NULL,` protocol=0

database proxy to remote queues, address=dblink address, protocol=0

3rd party proxies, address = `NULL,` protocol = 3rd party protocol.

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, 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 will be monotonically increasing (for each message that has the same `enq_time`). Hence this helps in maintaining the order of the messages. There will be 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 AQ and when should we use Oracle MessageBroker?

In Oracle9i, OMB functionality is provided in the Oracle database. So, if you are using the Oracle9i database, use the functionality offered by the database.

You do not need OMB.

With Oracle8i, use OMB in the following scenarios:

- To integrate with Tibco and MQ Series
- To use HTTP framework

Use JMS functionality directly from the database in other scenarios.

Can I use AQ with Virtual Private Database?

Yes, you can specify a security policy with 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, the dequeue will result in an error.

How do I clean up my retained messages?

The Advanced Queuing 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 will be cleaned up after their retention time expires.

I'm running propagation between multiple Oracle databases. 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.

JMS Questions

Why do the JMS dbms_aqadm.add_subscriber and dbms_aqadm.remove_subscriber calls sometimes hang when there are concurrent enqueues or dequeues happening on the same queue to which these calls are issued?

`Add_subscriber` and `remove_subscriber` are administrative operations on a queue. Though AQ does not prevent applications from issuing administrative and operational calls concurrently, they are executed serially. Both `add_subscriber` and `remove_subscriber` will block until pending transactions that have enqueued or dequeued messages commit and release the resources they hold. It is expected that adding and removing subscribers will not be a frequent event. It will mostly be part of the setup for the application. The behavior you observe will be acceptable in most cases. The solution is to try to isolate the calls to `add_subscriber` and `remove_subscriber` at the setup or cleanup phase when there are no other operations happening on the queue. That will make sure that they will not stay blocked waiting for operational calls to release resources.

Why do the TopicSession.createDurableSubscriber and TopicSession.unubscribe calls raise JMSException with the message "ORA - 4020 - deadlock detected while trying to lock object"?

`CreateDurableSubscriber` and unsubscribe calls require exclusive access to the Topics. If there are pending JMS operations (send / publish / receive) on the same Topic before these calls are issued, the ORA - 4020 exception is raised.

There are two solutions to the problem:

1. Try to isolate the calls to `createDurableSubscriber` and unsubscribe at the setup or cleanup phase when there are no other JMS operations happening on the Topic. That will make sure that the required resources are not held by other JMS operational calls. Hence the error ORA - 4020 will not be raised.
2. Issue a `TopicSession.commit` call before calling `createDurableSubscriber` and unsubscribe call.

Why doesn't AQ_ADMINISTRATOR_ROLE or AQ_USER_ROLE always work for AQ applications using Java/JMS API?

In addition to granting the roles, you would also need to grant execute to the user on the following packages:

- grant execute on `sys.dbms_aqin` to `<userid>`
- grant execute on `sys.dbms_aqjms` to `<userid>`

Why do I get java.security.AccessControlException when using JMS MessageListeners from Java stored procedures inside Oracle8*i* JServer?

To use MessageListeners inside Oracle8*i* JServer, you can do one for the following

1. `GRANT JAVASYSPRIV` to `<userid>`

Call `dbms_java.grant_permission ('JAVASYSPRIV', 'SYS:java.net.SocketPermission', '*', 'accept,connect,listen,resolve');`

Internet Access Questions

What is IDAP?

IDAP is Internet Data Access Presentation. An IDAP document encapsulates the AQ operation request and response in XML. IDAP is used to perform AQ operations such as enqueue, dequeue, send notifications, register for notifications, and propagation over the Internet standard transports-HTTP(s) and email. In addition, IDAP encapsulates transactions, security, transformation, and the character set ID for requests.

Which Web servers are supported for AQ Internet access functionality? Do I have to use Apache or can I use any Web server? Which servlet engines are supported for AQ Internet access? Can I use Tomcat?

Internet access functionality for 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 do I get transactional behavior while using email for AQ operations?

When you send IDAP messages via SMTP, each request is a separate transaction. The IDAP request must contain <AQXmlCommit/> as part of the message request to ensure that the operation is committed.

How does an Internet agent tie to an AQ agent stored in Oracle Internet Directory?

You can create an alias to an AQ agent in Oracle Internet Directory (OID). You can use these AQ agent aliases in the IDAP document sent over the Internet to perform AQ operations. Using aliases prevents exposing the internal name of the 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 AQ Servlet for 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>
```

You say IDAP is transport independent. Can I use my own transport to perform AQ operations using IDAP?

Yes, IDAP is transport independent. The IDAP document is sent over HTTP(s) or email. You can use IDAP to perform AQ operations over any transport. The IDAP client request can be sent over any transport to the AQ Servlet. The AQ Servlet parses the IDAP document and performs the AQ operation. The response for the AQ operation is also in IDAP. It can be sent to the client over your transport. In fact, you can implement AQ propagation over your transport.

Oracle Internet Directory (OID) Questions - Global Agents, Global Events, and Global Queues

Which events can be registered in OID?

All types of events—system events, user events, and notifications on queues—can be registered with OID. 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 email notifications.

How do I use agent information stored in an OID?

You can create aliases for an AQ agent in OID. These aliases can be specified while performing AQ operations-enqueue, dequeue, and notifications. This is specifically useful while performing AQ operations over the Internet when you do not want to expose an internal agent name. An alias can be used in an AQ operation (IDAP request).

Transformation Questions

What happens to enqueue, dequeue, or propagation if the transformation mapping raises an error?

Enqueue and dequeue of the message will raise the error to the application. If the error occurs during the dequeue operation, the retry count of the message is incremented. If the retry count exceeds `max_retries,` the message is moved to the exception queue. If the error occurs during propagation, it is handled in a manner similar to dequeue; propagation of the message will fail. It will be attempted again and the message will be moved to the exception queue when retry count exceeds `max_retries` for the queue.

How do you do transformation of XML data?

Transformation of XML data can be done in one of the following ways:

- Using the extract operator 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.`

Performance Questions

What is the maximum number of queues that a table can have without affecting performance?

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

When messages are moved from one queue to another using propagation, is there any optimization to move the messages in batches, rather than one at a time?

Yes, if it is optimized, propagation happens in batches.

If the remote queue is in a different database, we use a sequencing algorithm to avoid the need for a two-phase commit.

When a message needs to be sent to multiple queues in the same destination, it is sent multiple times. If the message needs to be sent to multiple consumers in the same queue at the destination, it is sent only once.

When is it useful to create indexes on a queue table? How do I create them?

Creating an index on the queue table is useful in the following scenarios:

a. Dequeuing using correlation ID: To expedite dequeue, an index can be created on the column `corr_id` of the underlying queue table `AQ$_<QueueTableName>`.

b. Dequeue using a condition: Assume this condition to the where-clause for the `SELECT` on the underlying queue table. An index on `<QueueTableName>` can be created to expedite the performance this `SELECT` statement.

What is the performance of Java (JMS) versus the PL/SQL API for AQ?

We do not have a specific performance evaluation of JMS versus the PL/SQL API. In general, the PL/SQL API is slightly better than the JMS API. The performance of the JMS and PL/SQL APIs in version 8.1.7 and higher should be comparable.

Installation Questions

How do I set up Internet access for AQ? What components are required?

See Chapter 17 for a full discussion. The following summarizes the steps required to set up Internet access for AQ queues:

1. Set up the AQ Servlet: If you are using a servlet execution engine that supports the Java Servlet 2.2 specification (such as Tomcat), 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), 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 AQ Servlet. Only authenticated users are allowed to access the AQ Servlet.
3. Set up user authorization: Register the AQ agent name that will be used to perform AQ operations using `DBMS_AQADM.CREATE_AQ_AGENT.` Map the AQ agent to the database users using `DBMS_AQADM.ENABLE_DB_ACCESS.`
4. Now clients can write IDAP requests and send to the AQ Servlet via HTTP `POST.`

How do I set up email notifications?

Here are the steps for setting up your database for email notifications:

1. Set the SMTP mail host: Invoke DBMS_AQELM.SET_MAILHOST as an AQ administrator.
2. Set the SMTP mail port: Invoke DBMS_AQELM.SET_MAILPORT as an 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 email notifications using the OCI or PL/SQL API.

How do I perform AQ operations via email?

See Chapter 17 for a full discussion. Currently, these operations are supported by Oracle Email Server 5.5 and higher. In summary, follow the steps for setting up Internet access for AQ. In addition, do the following:

1. Create an AQ Internet agent to access the servlet using SMTP. This agent's digital certificate should be registered in LDAP. The certificate location must be specified when the agent is registered using the `DBMS_AQADM.CREATE_AQ_AGENT` procedure.
2. Set up the Web server: Configure the Web server to receive requests from a user called `ORACLE_SMTP_AGENT.` This user will be used to access the AQ Servlet. Also specify `setEmailServerAddr` or `setEmailServerHost` in the `init()` method of the AQ Servlet.
3. Set up Oracle Email Server:
   a. Run `$ORACLE_HOME/admin/emailrule.sql` to create an AQ schema on the email server database.
   b. Create an email account for the destination database in which the AQ operations are to be performed.
   c. Set up an email rule for the destination database, so that it can route the AQ requests to the AQ Servlet on the web server. This can be done using the `DBMS_AQST.REGISTER_DB` procedure.
4. Now clients can write IDAP requests and send to the AQ Servlet via email.

How do I set up AQ propagation over the Internet?

See Chapter 17 for a full discussion. In summary, follow the steps for setting up Internet access for AQ. The destination databases need to be set up for Internet access, as follows:

1. At the source database, create the dblink with protocol as http, and host and port of the Web server running the AQ Servlet with the username password for authentication with the Web server/servlet runner. For example, if the Web server is running on machine `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, 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 AQ HTTP agent.

2. If SSL is used, 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 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 9i, 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').`

7

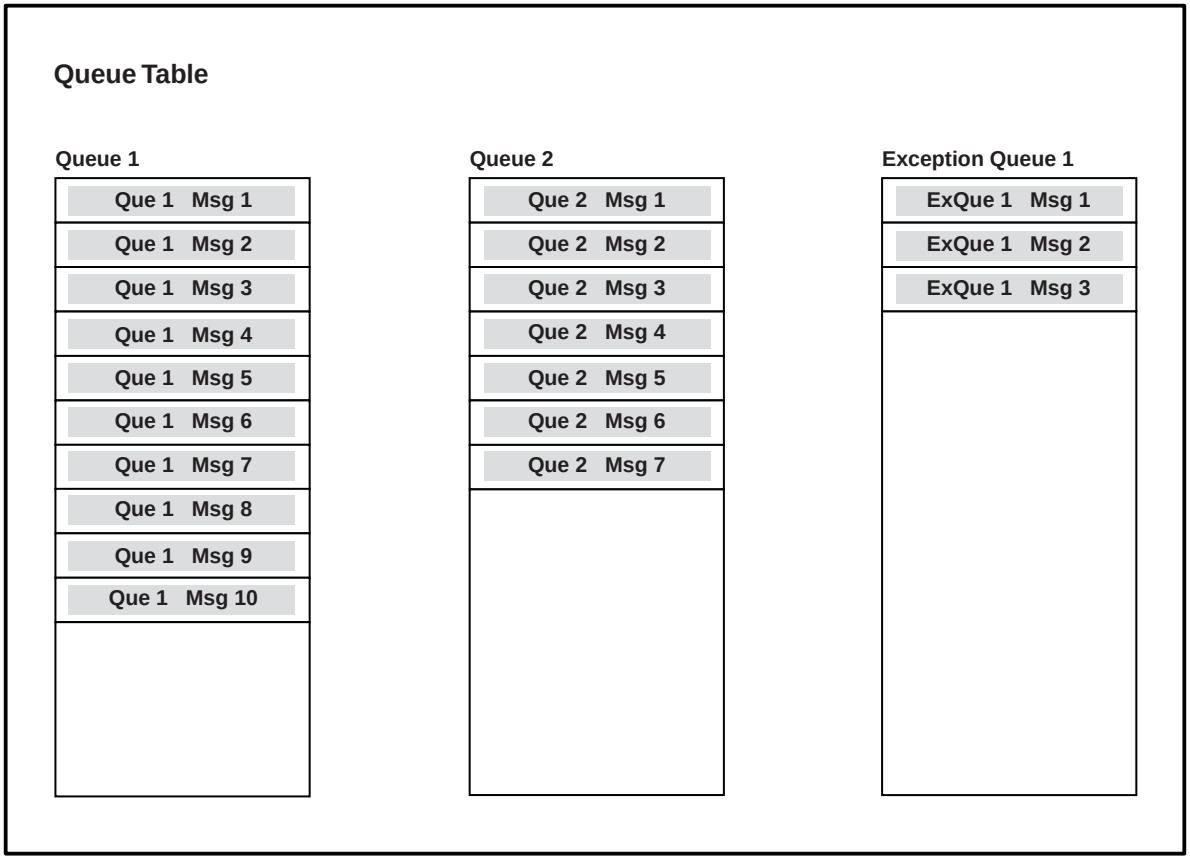
Modeling and Design

This chapter covers the fundamentals of AQ modeling and design in the following sections:

- Basic Queuing
- Basic Queuing Illustrated
- AQ Client-Server Communication
- Multiple-Consumer Dequeuing of the Same Message
- Dequeuing of Specified Messages by Specified Recipients
- AQ Implementation of Workflows
- AQ Implementation of Publish/Subscribe
- Message Propagation

Modeling Queue Entities

Figure 7–1 Basic Queues



The preceding figure portrays a queue table that contains two queues, and one exception queue:

- Queue1 — contains 10 messages.
- Queue2 — contains 7 messages.
- ExceptionQueue1 — contains 3 messages.

Basic Queuing

Basic Queuing — One Producer, One Consumer

At its most basic, one producer may enqueue different messages into one queue. Each message will be dequeued and processed once by one of the consumers. A message will stay 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. Note that an agent program, or application, can act as both a producer and a consumer.

Basic Queuing — Many Producers, One Consumer

At a slightly higher level of complexity, many producers may enqueue messages into a queue, all of which are processed by one consumer.

Basic Queuing — Many Producers, Many Consumers of Discrete Messages

In this next stage, many producers may enqueue messages, each message being processed by a different consumer depending on type and correlation identifier. The figure below shows this scenario.

Basic Queuing Illustrated

Figure 7–2, "Modeling Basic Queuing" portrays a queue table that contains one queue into which messages are being enqueued and from which messages are being dequeued.

Producers

The figure indicates that there are 6 producers of messages, although only four are shown. This assumes that two other producers (P4 and P5) have the right to enqueue messages even though there are no messages enqueued by them at the moment portrayed by the figure. The figure shows that:

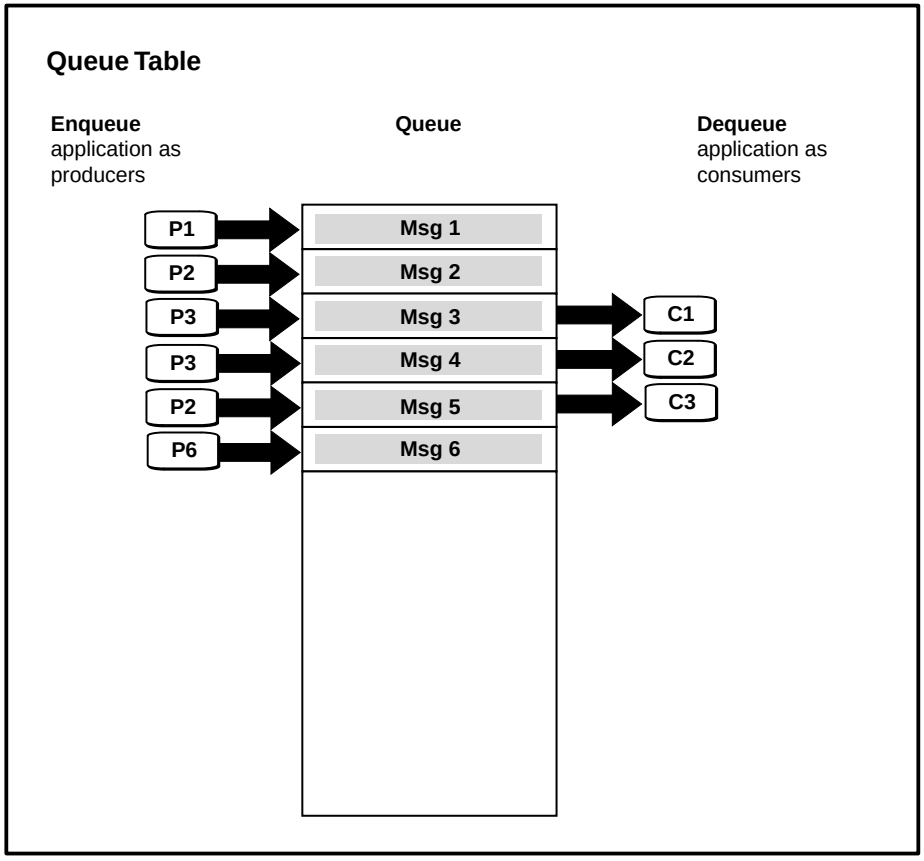
- A single producer may enqueue one or more messages.
- Producers may enqueue messages in any sequence.

Consumers

According to the figure, there are 3 consumers of messages, representing the total population of consumers. The figure shows that:

- Messages are not necessarily dequeued in the order in which they are enqueued.
- Messages may be enqueued without being dequeued.

Figure 7–2 Modeling Basic Queuing

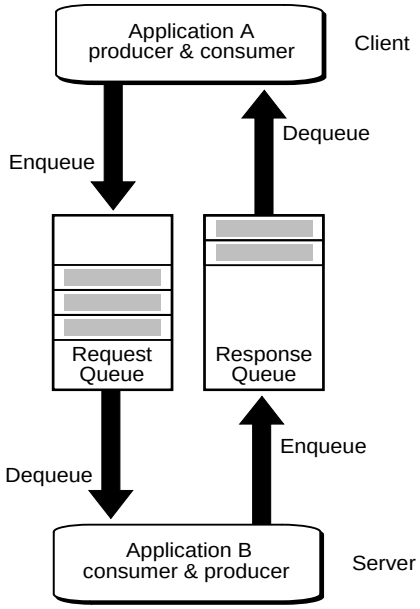


AQ Client-Server Communication

The figure portrays the enqueuing of multiple messages by a set of producers, and the dequeuing of messages by a set of consumers. What may not be readily evident in that sketch is the notion of time and the advantages offered by Oracle AQ.

Client-Server applications normally execute in a synchronous manner, with all the disadvantages of that tight coupling described above. Figure 7–3, "Client-Server Communication Using AQ" demonstrates the asynchronous alternative using AQ. In this example *Application B* (a server) provides service to *Application A* (a client) using a request/response queue.

Figure 7–3 Client-Server Communication Using AQ



1. *Application A* enqueues a request into the request queue.
2. *Application B* dequeues the request.
3. *Application B* processes the request.
4. *Application B* enqueues the result in the response queue.
5. *Application A* dequeues the result from the response queue.

In this way the client does not have to 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. In this way a process of double-deferral frees both client and server.

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

Multiple-Consumer 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, this would require management of a very large number of queues. Oracle AQ provides two mechanisms to allow for multiple consumers to dequeue the same message: *queue subscribers* and *message recipients*. The queue must reside in a queue table that is created with multiple consumer option to allow for subscriber and recipient lists. Each message remains in the queue until it is consumed by all its intended consumers.

Queue Subscribers Using this approach, multiple consumer-subscribers are associated with a queue. This will cause 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 AQ administrative package. The diagram below shows multiple producers enqueuing messages into queue, each of which is consumed by multiple consumer-subscribers.

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 may be selected from among the subscribers.

Figure 7–4 Multiple-Consumer Dequeuing of the Same Message

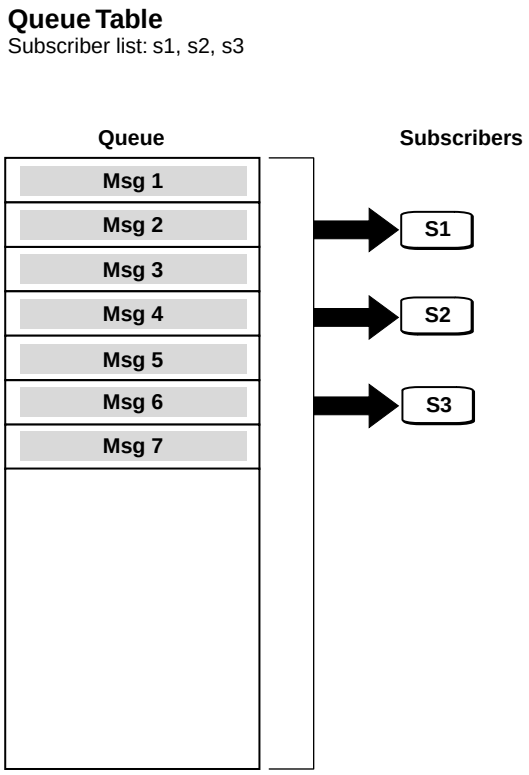


Figure 7–4 describes the case in which three consumers are all listed as subscribers of a queue. This is the same as saying that they all subscribe to all the messages that might ever be enqueued into that queue. The drawing illustrates a number of important points:

- The figure portrays the situation in which the 3 consumers are subscribers to 7 messages that have already been enqueued, and that they might become subscribers to messages that have not yet been enqueued.
- Every message will eventually be dequeued by every subscriber.

- There is no priority among subscribers. This means that there is no way of saying which subscriber will dequeue which message first, second, and so on. Or, put more formally: the order of dequeuing by subscribers is undetermined.
- We have no way of knowing from the figure about messages they might already have been dequeued, and which were then removed from the queue.

Figure 7–5 Communication Using a Multi-Consumer Queue

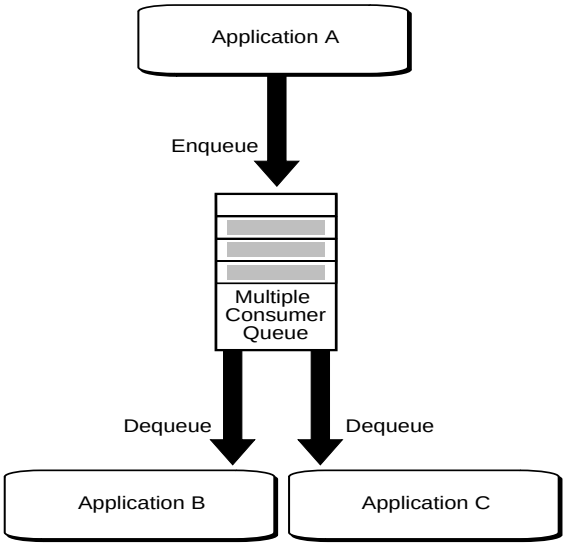
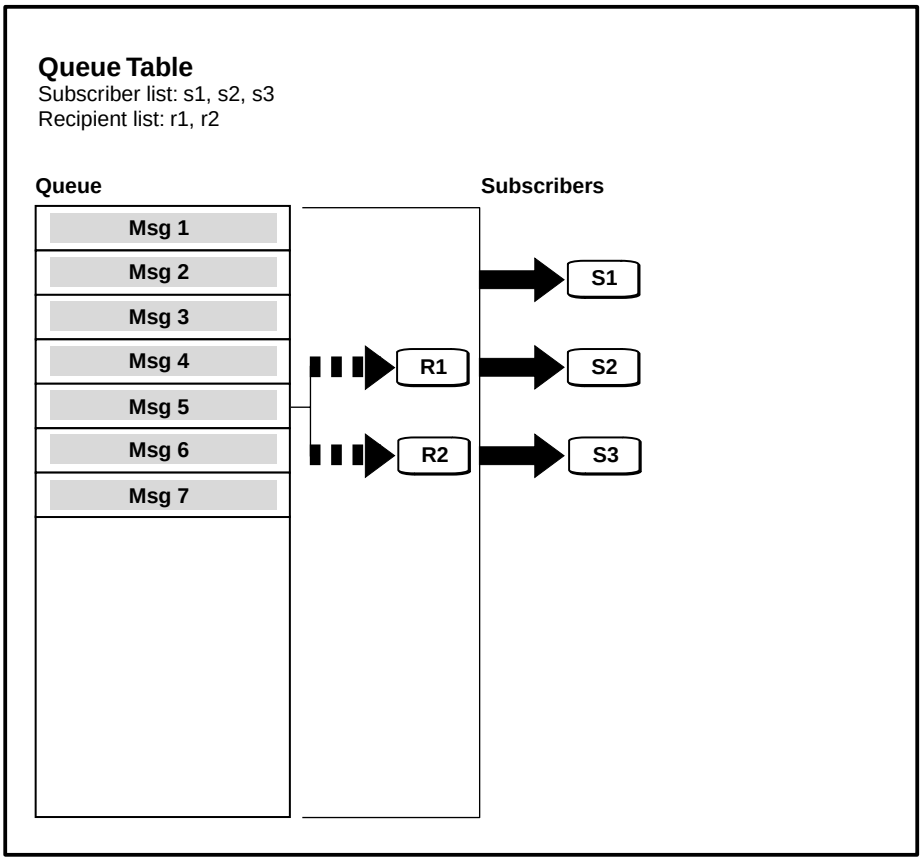


Figure 7–5 illustrates the same technology from a dynamic perspective. This examples concerns a scenario in which more than one application needs the result produced by an application. Every message enqueued by *Application A* is dequeued by *Application B* and *Application C*. To make this possible, the multiple consumer queue is specially configured with *Application B* and *Application C* as queue subscribers. Consequently, they are implicit recipients of every message placed in the queue.

Note: Queue subscribers can be applications or other queues.

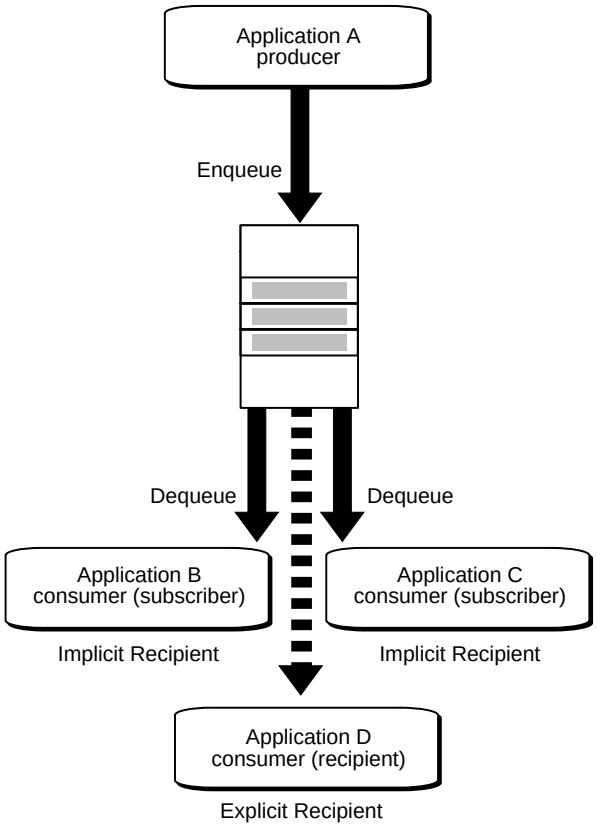
Figure 7–6 Dequeuing of Specified Messages by Specified Recipients



Dequeuing of Specified Messages by Specified Recipients

Figure 7–6 shows how a message can be specified for one or more recipients. In this case, *Message 5* is specified to be dequeued by *Recipient-1* and *Recipient-2*. As described by the drawing, neither of the recipients is one of the 3 subscribers to the queue.

Figure 7–7 Explicit and Implicit Recipients of Messages



We earlier referred to *subscribers* as implicit recipients in that they are able to dequeue all the messages placed into a specific queue. This is like subscribing to a magazine and thereby implicitly gaining access to all its articles. The category of consumers that we have referred to as *recipients* may also be viewed as explicit recipients in that they are designated targets of particular messages.

Figure 7–7 shows how Oracle AQ can adjust dynamically to accommodate both kinds of consumers. In this scenario *Application B* and *Application C* are implicit recipients (subscribers). But messages can also be explicitly directed toward specific consumers (recipients) who may or may not be subscribers to the queue. The list of such recipients is specified in the enqueue call for that message and overrides the

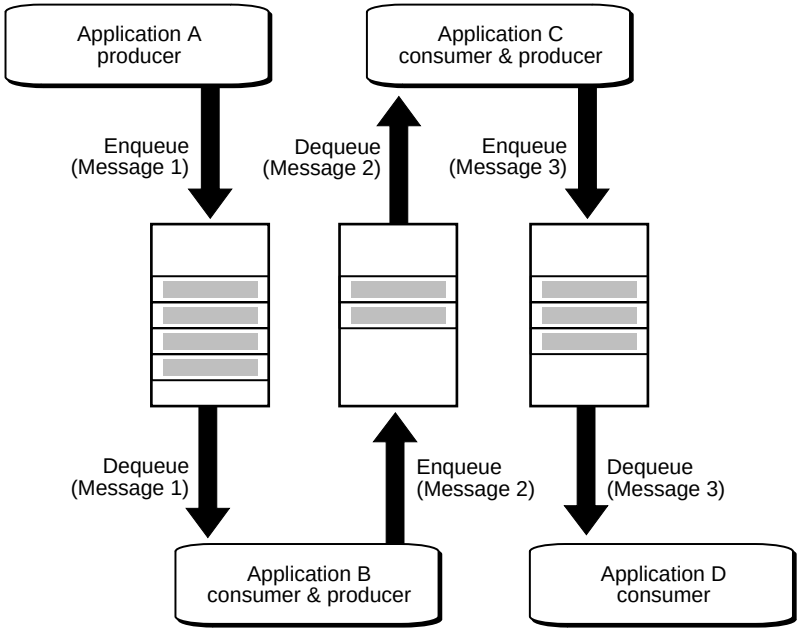
list of subscribers for that queue. In the figure, *Application D* is specified as the sole recipient of a message enqueued by *Application A*.

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

AQ Implementation of Workflows

Figure 7–8 illustrates the use of AQ for implementing workflows, also knows as chained application transactions. It shows a workflow consisting of 4 steps performed by *Applications A*, *B*, *C* and *D*. 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.

Figure 7–8 Implementing Workflows using AQ



From a workflow perspective, the passing of messages is a business asset above and beyond the value of the payload data. Hence, AQ supports the optional retention of messages for analysis of historical patterns and prediction of future trends. For instance, two of the three application scenarios at the head of the chapter are founded in an implementation of workflow analysis.

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

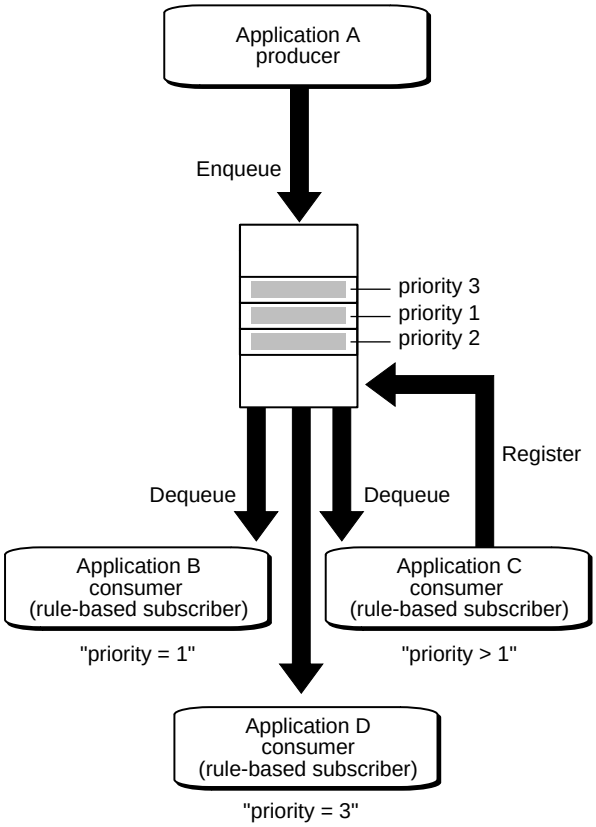
AQ Implementation of Publish/Subscribe

Figure 7–9 illustrates the use of AQ for implementing a publish/subscribe messaging scheme between applications. Application A is a publisher application which is publishing messages to a queue. Applications B, C, D are subscriber applications. Application A publishes messages anonymously to a queue. These

messages are then delivered to subscriber applications based on the rules specified by each application. Subscriber applications can specify interest in messages by defining a rule on message properties and message data content.

In the example shown, applications B has subscribed with rule "priority=1", application C has subscribed with rule "priority > 1" and application D has subscribed with rule "priority = 3". Application A enqueues 3 messages (priority 3, 1, 2). 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). Thus, message recipients are computed dynamically based on message properties and content. Additionally, the figure also illustrates how application C uses asynchronous notification for message delivery. Application C registers for messages on the queue. When messages arrive, application C is notified and can then dequeue the messages.

Figure 7–9 Implementing Publish/Subscribe using AQ



From a workflow perspective, the passing of messages is a business asset above and beyond the value of the payload data. Hence, AQ supports the optional retention of messages for analysis of historical patterns and prediction of future trends. For instance, two of the three application scenarios at the head of the chapter are founded in an implementation of workflow analysis.

Message Propagation

Fanning-Out of Messages

In AQ, message recipients can be either consumers or other queues. If the message recipient is a queue, the actual recipients are determined by the subscribers to the queue (which may in turn 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: A queue, *Source*, may have as its subscribers queues *dispatch1@dest1* and *dispatch2@dest2*. Queue *dispatch1@dest1* may in turn have as its subscribers the queues *outerreach1@dest3* and *outerreach2@dest4*, while queue *dispatch2@dest2* has as subscribers the queue *outerreach3@dest21* and *outerreach4@dest4*. In this way, messages enqueued in *Source* will be propagated to all the subscribers of four different queues.

Compositing (Funneling)

Another use of queues as a message recipient is the ability to combine messages from different queues into a single queue. This process is sometimes described as compositing.

For example, if queue *composite@endpoint* is a subscriber to both queues *funnel1@source1* and *funnel2@source2* then the subscribers to queue *composite@endpoint* can get all messages enqueued in those queues as well as messages enqueued directly into itself.

Figure 7–10 Message Propagation

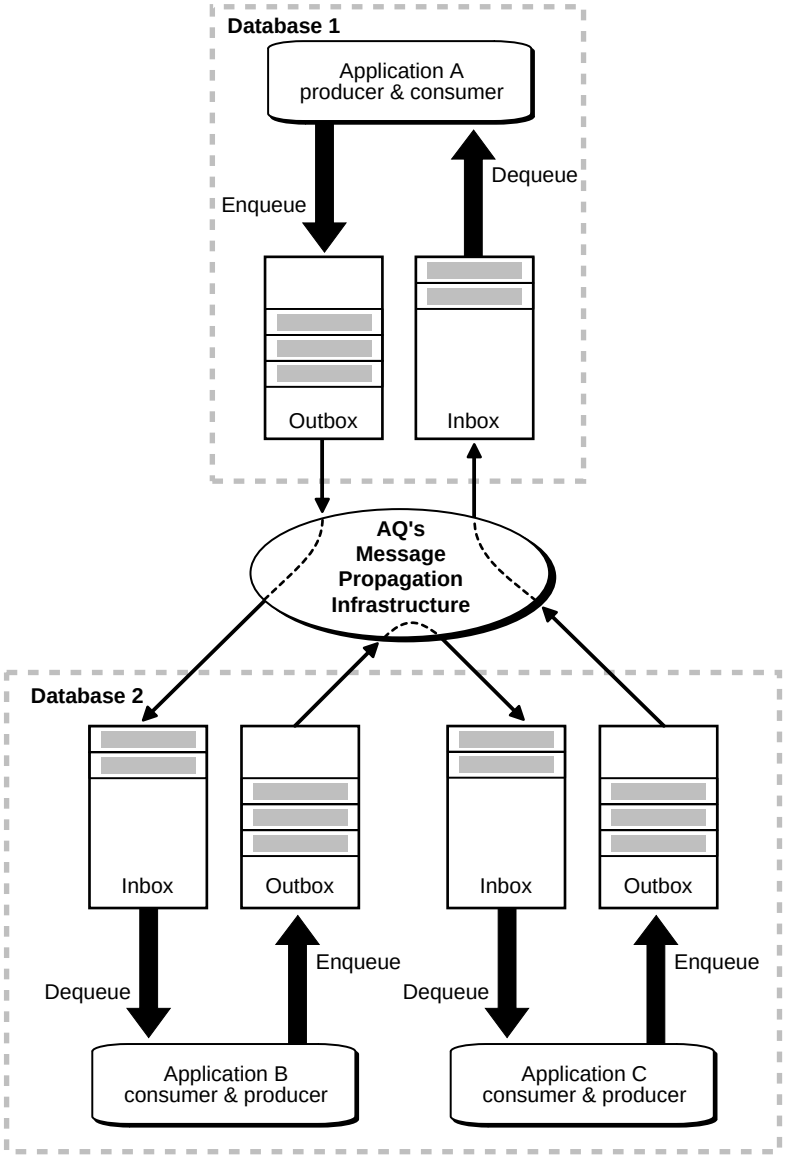


Figure 7–10 illustrates applications on different databases communicating via AQ.

Each application has an inbox and an outbox for handling incoming and outgoing messages. An application enqueues a message into its outbox irrespective of whether the message has to be sent to an application that is local (on the same node) or remote (on a different node).

Likewise, an application is not concerned as to whether a message originates locally or remotely. In all cases, an application dequeues messages from its inbox.

Oracle AQ facilitates all this interchange, treating messages on the same basis.

8

A Sample Application Using AQ

In Chapter 1, "Introduction to Oracle Advanced Queuing" a messaging system for a hypothetical company, BooksOnLine, was described. In this chapter the features of AQ in the BooksOnLine sample application are discussed under the following headings:

- A Sample Application
- General Features of Advanced Queuing
  - System-Level Access Control
  - Message Format Transformation
  - Structured Payloads
  - Queue Payloads Containing XMLType Attributes
  - Queue-Level Access Control
  - Nonpersistent Queues
  - Retention and Message History
  - Publish-Subscribe Support
  - Support for Oracle Real Application Clusters
  - Support for Statistics Views
  - Internet Access
- Enqueue Features
- Dequeue Features
- Asynchronous Notifications
- Propagation Features

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, one 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 AQ are exemplified within the BooksOnLine scenario to demonstrate the possibilities of AQ technology. A script for the sample code is provided in Appendix C, "Scripts for Implementing 'BooksOnLine'").

General Features of Advanced Queuing

- System-Level Access Control
- Structured Payloads
- Queue-Level Access Control
- Nonpersistent Queues
- Retention and Message History
- Publish-Subscribe Support
- Support for Oracle Real Application Clusters
- Support for Statistics Views

System-Level Access Control

Oracle 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 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. For more information, see "Oracle Enterprise Manager Support" on page 4-8.

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, the DBA should grant ENQUEUE_ANY and DEQUEUE_ANY system privileges to BOLADM:

```
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, BOLADM must be granted execute privileges on the DBMS_AQIN package:

```
GRANT EXECUTE ON DBMS_AQIN to BOLADM;
```

In the application, AQ propagators populate messages from the Order Entry(OE) schema to the Western Sales (WS), Eastern Sales (ES) and Worldwide Sales (OS) schemas. The WS, ES and OS schemas in turn populate messages to the Customer Billing (CB) and Customer Service (CS) schemas. Hence the OE, WS, ES and OS 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 need to grant schemas of the source queues enqueue privileges to the destination queues.

To simplify administration, all schemas that host a source queue in the BoooksOnLine 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','OS',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, 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

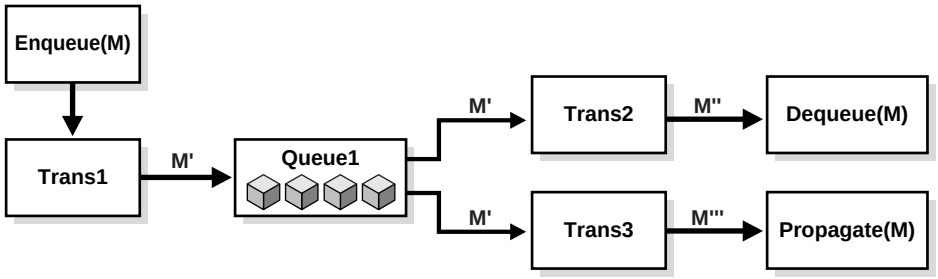
No example is provided with this release.

Message Format Transformation

You can specify transformations between different Oracle and user-defined types. Transformations must be expressed as SQL expressions, PL/SQL functions (including callouts), or Java stored procedures, with a return type of the target type. Only one-to-one message transformations are supported. The transformation engine is tightly integrated with Advanced Queuing to facilitate transformation of messages as they move through the database messaging system.

An Advanced Queuing application can enqueue or dequeue messages from a queue in the format specified by the application. An application can also specify a message format when subscribing to queues. The AQ propagator transforms messages to the format of the destination queue message, as specified by the remote subscription. The transformation function cannot write the database state or commit/rollback the current transaction. Transformations are exported with a schema or a full database export. Figure 8–1 shows how transformations are integrated with Advanced Queuing.

Figure 8–1 Transformations Integrated with Advanced Queuing



Trans1 (m, m')
Trans2 (m', m'')
Trans3 (m', m'')
Where, M, M' and M'' are messages of type m, m' and m''

PL/SQL (DBMS_TRANSFORM Package): Scenario and Code

In the BooksOnLine application, assume that the order type is represented differently in the order entry and the shipping applications.

The order type of the Order Entry application (in schema OE) is 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;
```

The order item of the shipping application is defined as follows

```
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 a composite type that contains an XMLType attribute. The payload type of the Overseas Shipping application is:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  XMLTYPE);
```

Creating Transformations

You can create transformations in the following ways:

- Create a single PL/SQL function that returns an object of the target type or the constructor of the target type.

  This representation is preferable for simple transformations or transformations that are not easily broken down into independent transformations for each attribute.

  ```
  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 `ORDER_XML_TYP`, with the order information in an XMLType attribute. The Order Entry site represents the order as an Oracle object, `ORDER_TYP`. Since 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.

ORDER_XML_TYP is a composite type that contains an XMLType attribute:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  XMLTYPE);
```

The transformation is defined as follows:

```
CREATE OR REPLACE FUNCTION CONVERT_TO_ORDER_XML(input_order TYPE OE.ORDER_
TYP)
RETURN OS.ORDER_XML_TYP AS
  xdata SYS.XMLType;
  new_order OS.ORDER_XML_TYP;
BEGIN
  select SYS_XMLGEN(input_order) into xdata from dual;
  new_order := OS.ORDER_XML_TYP(input_order.orderno, xdata);
  RETURN new_order;
END CONVERT_TO_ORDER_XML;

execute dbms_transform.create_transformation(
   schema =>         'OS',
   name   =>         'OE2XML',
   from_schema =>    'OE',
   from_type =>      'ORDER_TYP',
   to_schema =>      'OS',
   to_type =>        'ORDER_XML_TYP',
   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','OS.OS_bookedorders_que',null);

 dbms_aqadm.add_subscriber(
        queue_name     => 'OE.OE_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.orderregion = ''INTERNATIONAL'''
        transformation => 'OS.OE2XML');
END;
```

- Create a separate expression specified for each attribute of the target type. This representation is more readable and allows the application fine-grain control over the transformation.

```
/* 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 AQ, you can use object types to structure and manage the payload of messages. The object-relational capabilities of Oracle provide a rich set of data types that range from traditional relational data types to user-defined types (see "Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using PL/SQL" on page A-53 in Appendix A, "Oracle Advanced Queuing by Example").

Using strongly typed content, that is, content whose format is defined by an external type system, makes the following features available:

- Content-based routing: Advanced Queuing 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.
- Querying: the ability to execute queries on the content of messages allows users to examine current and processed messages for various applications, including message warehousing.

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 CLOBs.

- Query XMLType attributes using the operators `XMLType.existsNode()`, `XMLType.extract()`, and so on.

PL/SQL (DBMS_AQADM Package): Scenario and Code

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 above. 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 Object type called `order_xml_typ`. This type is a composite type that contains an XMLType attribute:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  XMLTYPE);
```

Visual Basic (OO4O): Example Code

Use the `dbexecutesql` interface from the database for this functionality.

Java (JDBC): Example Code

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 will create Java classes Customer, Book, OrderItem and OrderItemList that map to the SQL object types created above:

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 Oracle8i 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;
```

Queue Payloads Containing XMLType Attributes

You can create queues with payloads that contain 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 CLOBs.
- Selectively dequeue messages with XMLType attributes using the operators `XMLType.existsNode()`, `XMLType.extract()`, and so on.

  See Also: *Oracle9i Application Developer's Guide - XML* for details on XMLType operations

- Define transformations to convert Oracle objects to XMLType.

- Define rule-based subscribers that query message content using XMLType operators such as `XMLType.existsNode()` and `XMLType.extract()`.

In the BooksOnline application, assume that the Overseas Shipping site represents the order as `ORDER_XML_TYP`, with the order information in an XMLType attribute. The Order Entry site represents the order as an Oracle object, `ORDER_TYP`.

`ORDER_XML_TYP` is a composite type that contains an XMLType attribute:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  XMLTYPE);
```

The Overseas queue table and queue are created as follows:

```
BEGIN
dbms_aqadm.create_queue_table(
   queue_table        => 'OS_orders_pr_mqtab',
   comment            => 'Overseas Shipping MultiConsumer Orders queue table',
   multiple_consumers => TRUE,
   queue_payload_type => 'OS.order_xml_typ',
   compatible         => '8.1');
END;

BEGIN
dbms_aqadm.create_queue (
   queue_name   => 'OS_bookedorders_que',
   queue_table  => 'OS_orders_pr_mqtab');
END;
```

Since 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','OS.OS_bookedorders_que',null);

  dbms_aqadm.add_subscriber(
   queue_name     => 'OE.OE_bookedorders_que',
   subscriber     => subscriber,
   rule           => 'tab.user_data.orderregion = ''INTERNATIONAL''',
   transformation => 'OS.OE2XML');
```

```
END;
```

For more details on defining transformations that convert the type used by the Order Entry application to the type used by Overseas shipping, see "Creating Transformations" on page 8-7.

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           OS.order_xml_typ;
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.xdata.extract(
''/ORDER_TYP/CUSTOMER/COUNTRY/text()'').getStringVal()=''CANADA''';

            dopt.consumer_name : = 'Overseas_Shipping';

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name         => 'OS.OS_bookedorders_que',
                dequeue_options    => dopt,
                message_properties => mprop,
                payload            => deq_order_data,
                msgid              => deq_msgid);
            commit;

            dbms_output.put_line(' Order for Canada - Order No: ' ||
                                 deq_order_data.orderno);

          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE ORDERS  ---- ');
                 new_orders := FALSE;
```

```
            END;
        END LOOP;
end;
```

Queue-Level Access Control

Oracle 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 needs to 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. For more information, see "Oracle Enterprise Manager Support" on page 4-8.

Scenario

The BooksOnLine application processes customer billings in its CB and CBADM schemas. CB (Customer Billing) 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 illegal operations from the application, the following two grant calls are needed:

PL/SQL (DBMS_AQADM Package): Example Code

```
/* Grant dequeue privilege on the shopped 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.

Java (JDBC): Example Code

```
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);
    }
}
```

Nonpersistent Queues

A message in a nonpersistent queue is not stored in a database table. You create a nonpersistent queue, which can be of either single-consumer or multi-consumer type. These queues are created in a system-created queue table (AQ$_MEM_SC for single-consumer queues and AQ$_MEM_MC for multi-consumer queues) in the

schema specified by the `create_np_queue` command. Subscribers can be added to the multi-consumer queues (see "Creating a Nonpersistent Queue" on page 9-28). Nonpersistent queues can be destinations for propagation.

You use the enqueue interface to enqueue messages into a nonpersistent queue in the normal way. You can enqueue RAW and Object Type (ADT) messages into a nonpersistent queue. You retrieve messages from a nonpersistent queue through the asynchronous notification mechanism, registering for the notification (using `OCISubcriptionRegister` or `DBMS_AQADM.REGISTER`) for the queues you are interested in (see "Registering for Notification" on page 11-58).

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: Documentation on `DBMS_AQADM.REGISTER` in *Oracle9i Supplied PL/SQL Packages and Types Reference* and documentation on `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 per application process. The application processes are named `APP_1`, `APP_2`, `APP_3`. (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 multi-consumer 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/logout events per application process. Note that the dispatcher process needs to connect to the database for registering the subscriptions only. The notifications themselves can be received while the process is disconnected from the database.

PL/SQL (DBMS_AQADM Package): Example Code

```
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         => 'LOGON_LOGOFF',
                                   multiple_consumers => TRUE);

/* Enable the queue for enqueue and dequeue: */
EXECUTE dbms_aqadm.start_queue(queue_name => 'LOGON_LOGOFF');

/* Nonpersistent Queue Scenario - procedure to be executed upon logon: */
CREATE OR REPLACE PROCEDURE  User_Logon(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:= 'LOGON';
  msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
  /* payload is NULL */
  dbms_aq.enqueue(
        queue_name         => 'LOGON_LOGOFF',
        enqueue_options    => enqopt,
        message_properties => msgprop,
        payload            => payload,
        msgid              => enq_msgid);

END;

/* Nonpersistent queue scenario - procedure to be executed upon logoff: */
CREATE OR REPLACE PROCEDURE  User_Logoff(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:= 'LOGOFF';
  msgprop.recipient_list(0) := aq$_agent(app_process, NULL, NULL);
  /* Payload is NOT NULL: */
```

```
payload := adtmsg(1, 'Logging Off');

dbms_aq.enqueue(
        queue_name         => 'LOGON_LOGOFF',
        enqueue_options    => enqopt,
        message_properties => msgprop,
        payload            => payload,
        msgid              => enq_msgid);
 END;
/


/* If there is a login at APP1, enqueue a message into 'login_logoff' with
   correlation 'LOGIN': */
EXECUTE User_logon('APP1');

/* If there is a logout at APP3, enqueue a message into 'login_logoff' with
   correlation 'LOGOFF' and payload adtmsg(1, 'Logging Off'): */
EXECUTE User_logoff('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

/* LOGON / password:  */
static text *username = (text *) "OE";
static text *password = (text *) "OE";

/* The correlation strings of messages: */
static char  *logon = "LOGON";
static char  *logoff = "LOGOFF";

/* 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  logon;
  ub4  logoff;
};

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(logon) &&                  /* logon event */
       !memcmp((dvoid *)correlation, (dvoid *)logon, strlen(logon)))
  {
     user_count[appno].logon++;
                           /* increment logon count for the app process */
        printf("Logon by APP%d \n", (appno+1));
        printf("Logon Payload length = %d \n", pay1);
   }
  else if  (lcorrelation == strlen(logoff) &&           /* logoff event */
       !memcmp((dvoid *)correlation,(dvoid *)logoff, strlen(logoff)))
  {
```

```
      user_count[appno].logoff++;
                          /* increment logoff count for the app process */
      printf("Logoff by APP%d \n", (appno+1));
      printf("Logoff Payload length = %d \n", pay1);
  }
  else                              /* correlation is "LOGON" or "LOGOFF" */
    printf("Wrong correlation in notification");

  printf("Total  : \n");

  printf("App1 : %d \n", user_count[0].logon-user_count[0].logoff);
  printf("App2 : %d \n", user_count[1].logon-user_count[1].logoff);
  printf("App3 : %d \n", user_count[2].logon-user_count[2].logoff);

}

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.LOGON_LOGOFF:APP1",
                (ub4) strlen("OE.LOGON_LOGOFF: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.LOGON_LOGOFF:APP2",
                (ub4) strlen("OE.LOGON_LOGOFF: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.LOGON_LOGOFF:APP3",
                (ub4) strlen("OE.LOGON_LOGOFF: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

AQ allows users to retain messages in the queue table and therefore use SQL to query 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, 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 need to determine the average order processing time. This includes the time the order has to wait in the `backed_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 "tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues & Subscribers" on page C-2 of Appendix C, "Scripts for Implementing 'BooksOnLine'".

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 AQ adds 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 do not have to explicitly handle or manage message recipients. This allows 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 content (payload) and message header properties of a queue. The system automatically routes messages by computing recipients for published messages using the rule-based subscriptions.

You can implement a publish-subscribe model of communication using 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 may 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 may receive messages in the following manner:
  - A dequeue call retrieves messages that match the subscription criteria.
  - A listen call may 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 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.

Define 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.

Set Up Subscriptions The various shipping applications and the customer service application (i.e., 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. Rules are used 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);
```

Set 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');
```

Publish 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.

Receive Messages Each of the shipping applications and the Customer Service application will then receive 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 is marked RUSH. This application then dequeues messages and processes its orders for shipping.

Support for Oracle Real Application Clusters

The Oracle Real Application Clusters facility can be used to improve 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) on different queues to occur in parallel.

The 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, the queue monitor changes ownership to a suitable instance such as the secondary instance.

AQ propagation is able to make use of Oracle 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 will be handling the propagation from queues stored in that queue table, thereby minimizing pinging. Additional discussion on this topic can be found under AQ propagation scheduling (see "Scheduling a Queue Propagation" on page 9-74 in Chapter 9, "Administrative Interface").

See also: *Oracle9i Real Application Clusters Installation and Configuration*

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 done 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 will be set to instance 1. However, if Oracle Real Application Clusters are set up correctly and both instances 1 and 2 are available, then queue table `OE_orders_sqtab` will be owned by instance 1 and the other queue table will be 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 example below. Information about the primary, secondary and owner instance of a queue table can be obtained by querying the view `USER_QUEUE_TABLES` (see "Selecting Queue Tables in User Schema" on page 10-22 in "Administrative Interface: Views").

Note: Queue names and queue table names are converted to upper case. Mixed case (upper and lower case together) is not supported for queue names and queue table names.

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-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.setPrimaryInstance(1);
        sqt_prop.setSecondaryInstance(2);

        sq_table = aq_sess.createQueueTable("OE", "OE_orders_sqtab", sqt_prop);

        /* Create a multi-consumer orders queue table */
        mqt_prop = new AQQueueTableProperty("BOLADM.order_typ");
        mqt_prop.setComment("Order Entry Multi Consumer 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);
    }
}
```

Support for Statistics Views

Each instance keeps its own AQ statistics information in its own 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 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 "Selecting the Number of Messages in Different States for the Whole Database" on page 10-34 in Chapter 10, "Administrative Interface: Views").

PL/SQL (DBMS_AQADM Package): Example Code

```
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';
```

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

See Chapter 17, "Internet Access to Advanced Queuing" for information on Internet access to Advanced Queuing features.

Enqueue Features

- 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 AQ XML Servlet

Subscriptions and Recipient Lists

In a single-consumer queue, a message can be processed once by only one consumer. What happens when there are multiple processes or operating system threads concurrently dequeuing from the same queue? Given that a locked message cannot be dequeued by a process other than the one which created the lock, each process will dequeue the first unlocked message at the head of the queue.

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.

AQ allows a single message to be processed/consumed by more than one consumer. To use this feature, you must create multi-consumer queues and enqueue the messages into these multi-consumer queues. 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 (see "Adding a Subscriber" on page 9-61 in Chapter 9, "Administrative Interface"). 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 multi-consumer queue.

All consumers that are added as subscribers to a multi-consumer 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 "Agent Type (aq$_agent)" on page 2-3 for 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 will only be 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 will not be available for dequeue by this consumer.

You can remove a subscription by using the DBMS_AQADM.REMOVE_SUBSCRIBER procedure (see "Removing a Subscriber" in Chapter 9, "Administrative Interface"). AQ will automatically remove from the queue all metadata corresponding to the consumer identified by the AQ$_AGENT parameter. In other words, it is not an error to execute the REMOVE_SUBSCRIBER procedure even when there are pending messages that are available for dequeue by the consumer. These messages will be automatically made unavailable for dequeue after the REMOVE_SUBSCRIBER procedure is executed. In a queue table that is created with the compatible parameter set to '8.1' or higher, such messages that were not dequeued by the consumer will be shown as "UNDELIVERABLE" in the AQ$<queue_table> view. Note that a multi-consumer 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 do not need to specify subscriptions for a multi-consumer queue provided that producers of messages for enqueue supply a recipient list of consumers. In some situations it may 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, it overrides the subscription list. In other words, messages that have a specified recipient list will not be 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 "Enqueuing a Message" on page 11-5 in Chapter 11, "Operational Interface: Basic Operations").

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 "Creating a Queue Table" on page 9-6 in Chapter 9, "Administrative Interface").

Priority ordering of messages is achieved by specifying priority, enqueue time as the sort order for the message. If priority ordering is chosen, each message will be assigned a priority at enqueue time by the enqueuer. At dequeue time, the messages will be dequeued in the order of the priorities assigned. If two messages have the same priority, 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). or
- 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 enqueueing
     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 back orders 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 execute 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 enqueueing
'  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 back orders queues.
       */
    }
```

Time Specification: Delay

AQ supports delay delivery of messages by letting the enqueuer specify a delay interval on a message when enqueueing the message, that is, the time before that a message cannot be retrieved by a dequeue call. (see "Enqueuing a Message [Specify Message Properties]" on page 11-10 in Chapter 11, "Operational Interface: Basic Operations"). The delay interval determines when an enqueued message is marked as available to the dequeuers after the message is enqueued. The producer can also specify the time when a message expires, at which time the message is moved to an exception queue.

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 will then post 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. Note that expiration processing requires that the queue monitor be running.

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 on a back order queue. If the shipping policy is that all back orders 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 orders 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
/* Req-enqueue a back order into a back order queue and set a delay of 7 days;
   all back orders 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 back order 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 := 'OS.OS_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 back order into a back order queue and set a delay of 7 days;
   all back orders 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 back order 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("OS", "OS_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 (see "Creating a Queue Table" on page 9-6 in Chapter 9, "Administrative Interface"). All messages belonging to a group have to be created in the same transaction and all messages created in one transaction belong to the same group. This feature allows you 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. 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, it is important to note that the destination queue where messages have to be propagated must also be enabled for transactional grouping. There are also some restrictions you need to keep in mind if the message grouping property is to be preserved while dequeuing messages from a queue enabled for transactional grouping (see "Dequeue Methods" on page 8-59 and "Modes of Dequeuing" on page 8-70 for additional information).

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 behavior.

In the example given below, 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. Note that 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: Queue names and queue table names are converted to upper case. Mixed case (upper and lower case together) is not supported for queue names and queue table names.

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 8-4, 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 ADT `OE.order_typ`. The Western shipping application represents the order item in the form of the ADT `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`.

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 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 AQ XML Servlet

You can perform enqueue requests over the Internet using IDAP. See Chapter 17, "Internet Access to Advanced Queuing" for more information on sending AQ requests using IDAP.

Scenario

In the `BooksOnLine` application, a customer can request:

- FedEx shipping (priority 1),
- Priority air shipping (priority 2). or
- 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 IDAP. The XML message will have the following format.

```
<?xml version="1.0"?>
   <Envelope xmlns= "http://ns.oracle.com/AQ/schemas/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

- 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 AQ XML Servlet

Dequeue Methods

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

- Correlation identifier
- Message identifier
- Dequeue condition

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. A dequeue call with a correlation identifier will directly remove a message of specific interest, rather than using a combination of locked and remove mode to first examine the content and then remove the message. Hence, the correlation identifier usually contains the most useful attribute of a payload. If there are multiple messages with the same correlation identifier, 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.

Note that dequeueing a message with any of the dequeue methods will not preserve the message grouping property (see "Message Grouping" on page 8-52 and "Message Navigation in Dequeue" on page 8-66 for further information).

Scenario

In the BooksOnLine example, rush orders received by the East shipping site are processed first. This is achieved by dequeueing the message using the correlation identifier, which has been defined to contain the order type (rush / normal). For an illustration of dequeueing using a message identifier, refer to the `get_northamerican_orders` procedure discussed in the example under "Modes of Dequeuing" on page 8-70.

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 := 'OS.OS_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("OS", "OS_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 multi-consumer normal 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. (See "Adding a Subscriber" on page 9-61 or Enqueuing a Message [Specify Message Properties] on page 11-10).

- 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 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 AQ will provide 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 multi-consumer queue. Expired messages from multi-consumer 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, multi-consumer exception queues behave like single-consumer queues because each expired message can be dequeued only once using a NULL consumer name. Note that expired messages can be dequeued only by specifying a message ID if the multi-consumer 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 multi-consumer queues. This allows dequeuers to complete the dequeue operation by not locking the message in the queue table. Since the queue monitor removes messages that have been processed by all consumers from multi-consumer queues approximately once every minute, users may see a delay when the messages have been completely processed and when they are physically removed from the queue.

Local and Remote Recipients

Consumers of a message in multi-consumer 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$_AGENT type (see "Agent Type (aq$_agent)" on page 2-3 in Chapter 2, "Basic Components").
- A Remote consumer dequeues from a queue that is different (but has the same payload type as the source queue) from the queue where the message was enqueued. As such, users need to be familiar with and use the 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 will dequeue the message from a different queue in the same database. These addresses will be 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, the schema of the current user

executing the ADD_SUBSCRIBER procedure or the enqueue is used (see "Adding a Subscriber" on page 9-61, or "Enqueuing a Message" on page 11-5 in Chapter 11, "Operational Interface: Basic Operations"). Use the DBMS_AQADM.SCHEDULE_PROPAGATION command with a NULL destination (which is the default) to schedule propagation to such remote consumers (see "Scheduling a Queue Propagation" on page 9-74 in Chapter 9, "Administrative Interface").

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 will be of the form [schema].queue_name@dblink. If the schema is not specified, 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), the default domain as specified by the db_domain init.ora parameter will be used. Use the DBMS_AQADM.SCHEDULE_PROPAGATION procedure with the database link as the destination to schedule the propagation. 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 will need to 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 will be 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, 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. 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, the expiration at the remote queue will be set to 45 minutes.

Since 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".

These selections work in a slightly different way if the queue is enabled for transactional grouping.

- If the "first message" is requested, the dequeue position is reset to the beginning of the queue.
- If the "next message" is requested, the position is set to the next message of the same transaction
- If the "next transaction" is requested, the position is set to the first message of the next transaction.

Note that 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 dequeueing some of the messages of a transaction and committing (see "Dequeue Methods" on page 8-59).

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 non-zero wait time, then the navigating position is automatically changed to the beginning of the queue. If a zero wait time is specified, you may get an exception when the end of the queue is reached.

Scenario

The following scenario in the `BooksOnLine` example continues the message grouping example already discussed with regard to enqueuing (see "Dequeue Methods" on page 8-59).

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 message in the current group / transaction have been fetched, 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 (see "Dequeuing a Message" on page 11-47 in Chapter 11, "Operational Interface: Basic Operations").

- 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, it remains available for further processing. Similarly if a message is locked, 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.

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 the message will not be used.

Note that after a message has been browsed, there is no guarantee that the message can be dequeued again since 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 non-zero 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 non-zero wait time can dequeue the same message over and over again. We recommend that you use a non-zero 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, 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.

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, the message is consumed (deleted from the queue) using `REMOVE_NODATA` (since the payload is already known). Otherwise, the lock on the message is released by the commit call. Note that 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 := 'OS.OS_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 OS;

CONNECT ES/ES;

/*  Browse all booked orders for East_Shipping: */
EXECUTE BOLADM.shipping_bookedorder_deq('East_Shipping', DBMS_AQ.BROWSE);

CONNECT OS/OS;

/*  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 above. 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("OS", "OS_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

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 (see

"Dequeuing a Message" on page 11-47) or the LISTEN operation to wait for the arrival of a message in more than one queue (see "Listening to One (Many) Queue(s)" on page 11-24 in Chapter 11, "Operational Interface: Basic Operations").

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 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 resource 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, provided that propagation has been scheduled to the remote queue using DBMS_AQADM.SCHEDULE_PROPAGATION. In this case, the AQ propagator will wake up the blocked dequeuer after a message has been propagated.

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 code below.

- If the wait time is specified as 10 seconds, the dequeue call is blocked with a time out 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, a wait time of 0 seconds is implemented. The dequeue call in this case will return immediately even if there are no messages in the queue.
- If the wait time is specified as DBMS_AQ.FOREVER, the dequeue call is blocked without a time out 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, it is regarded as an unsuccessful attempt to consume the message. 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, 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, the message is moved to the exception queue and is no longer available to applications.

Retry Delay

A bad condition can cause the transaction receiving a message to abort. AQ allows users to hide the bad message for a pre-specified interval. A retry_delay can be specified along with maximum retries. This means that a message that has had a failed attempt will be visible in the queue for dequeue after the retry_delay interval. Until then it will be in the WAITING state. In the 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. Note that maximum retries and retry delay are not available with 8.0-compatible multi-consumer 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, abort 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, abort 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

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. Also, a multi-consumer 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. The exception queue created for messages intended for a multi-consumer queue must itself be a multi-consumer queue. Like any other queue, the exception queue must be enabled for dequeue using the DBMS_AQADM.START_QUEUE procedure. You will get an Oracle error if you try to enable an exception queue for enqueue.

When a message has expired, it is moved to an exception queue. The exception queue for a message in multi-consumer queue must also be a multi-consumer queue. Expired messages from multi-consumer 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 multi-consumer exception queues behave 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. Note that expired messages can be dequeued only by specifying a message ID if the multi-consumer exception queue was created in a queue table without the compatible parameter or with the compatible parameter set to '8.0'.

The exception queue is a message property that can be specified during enqueue time (see "Enqueuing a Message [Specify Message Properties]" on page 11-10 in Chapter 11, "Operational Interface: Basic Operations"). 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.

If an exception queue is not specified, the default exception queue is used. If the queue is created in a queue table, for example, QTAB, the default exception queue will be 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 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 will be 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 will not expire.
- The message is being dequeued successfully. However, because of an error that arises while processing the message, the application that dequeues the message chooses to roll back the transaction. In this case, the message is returned to the queue and will be 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, the message will be 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 multi-consumer queues. No retry limit is not supported for 8.0-compatible multi-consumer queues.
- 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, AQ will attempt 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 multi-consumer 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 multi-consumer 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 will be available for dequeue immediately

after the RETRY_COUNT is incremented. Otherwise the message will be unavailable for RETRY_DELAY seconds. After RETRY_DELAY seconds, the queue monitor marks the message as READY.

For a multi-consumer queue, RETRY_DELAY is per subscriber.

Scenario

In the BooksOnLine application, the business rule for each shipping region is that an order will be placed in a back order queue if the order cannot be filled immediately. The back order application will try to fill the order once a day. If the order cannot be filled within 5 days, 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 AQ.

The example below 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 back order 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 back order 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 back order 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, the default exception queue of the queue will be 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 back order 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 back order 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 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;
/
```

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 East Shipping -
    East shipping handles all Eastern region orders,
    East 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 OS 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. Since 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','OS.OS_bookedorders_que',null);

  dbms_aqadm.add_subscriber(
        queue_name     => 'OE.OE_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.orderregion = ''INTERNATIONAL''',
        transformation => 'OS.OE2XML');
```

```
END;
```

See "Message Format Transformation" on page 8-4 for more details on defining transformations.

Assume that the Overseas Shipping site has a subscriber, `Overseas_DHL`, for handling `RUSH` orders. Since `OS_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     => 'OS.OS_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.xdata.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 East Shipping -
                East shipping handles all Eastern region orders,
                East 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", "OS.OS_bookedorders_que");
            rule = "tab.user_data.orderregion = 'INTERNATIONAL'";

            bookedorders_q.addSubscriber(agt3, rule);
        }
        catch (Exception ex)
        {
            System.out.println("Exception: " + ex);
        }
    }
```

Listen Capability

Advanced Queuing 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, a dequeue must be used to retrieve the message (see Listening to One (Many) Queue(s) on page 11-24 in Chapter 11, "Operational Interface: Basic Operations").

Without the `LISTEN` call, an application which sought to dequeue from a set of queues would have to 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, these approaches will create unacceptable overhead. The `LISTEN` call is well suited for such applications.

Note that 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 done 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 8-4 and "Message Transformation During Enqueue" on page 8-55, 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 AQ XML Servlet

You can perform dequeue requests over the Internet using IDAP. See Chapter 17, "Internet Access to Advanced Queuing" for more information on receiving AQ messages using IDAP.

In the BooksOnline scenario, assume that the East shipping application receives AQ messages with a correlation identifier 'RUSH' over the Internet. The dequeue request will have the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/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 notification of a message of interest. The client can use it to monitor multiple subscriptions. The client does not have to be connected to the database to receive notifications regarding its subscriptions.

Clients can use the OCI function, `OCISubcriptionRegister`, or the PL/SQL procedure `DBMS_AQ.REGISTER` to register interest in messages in a queue (see

"Registering for Notification" in Chapter 11, "Operational Interface: Basic Operations").

> **See also:** Documentation on `Register for Notification` in the *Oracle Call Interface Programmer's Guide* and documentation on `DBMS_AQ.REGISTER` the *Oracle Supplied PL/SQL Packages reference*

The client can specify a callback function that is invoked for every new message that is enqueued. The callback can be a C function in an OCI client, a user-defined PL/SQL procedure. The user can also specify an email address to which notifications will be mailed.

For nonpersistent queues, the message is delivered to the client as part of the notification. For persistent queues, only the message properties are delivered as part of the notification. Consequently, in the case of persistent queues, the client has to make an explicit dequeue to access the contents of the message.

Clients can also specify the presentation for notifications, 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 desired.

```
#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 email 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 done as follows:

First define a PL/SQL procedure that will be 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, email 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 email 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 info 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 AQ XML Servlet

Clients can register for AQ notifications over the Internet. See Chapter 17, "Internet Access to Advanced Queuing" for more information on registering for AQ notifications using IDAP.

The register request has the following format:

```
?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/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 email notification sent to `john@company.com` will have 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

- Propagation
- Propagation Scheduling
- Scenario
- Enhanced Propagation Scheduling Capabilities
- Exception Handling During Propagation
- Message Format Transformation During Propagation

Propagation

This feature enables applications to communicate with each other without having to be connected to the same database, or to the same queue. Messages can be propagated from one Oracle AQ to another, irrespective of whether these are local or remote. Propagation is performed by snapshot (job_queue_processes) background processes. Propagation to remote queues is done using database links and Net8.

The propagation feature is used as follows. First one or more subscribers are defined for the queue from which messages are to be propagated (see "Subscriptions and Recipient Lists" on page 8-37). Second, a schedule is defined for each destination where messages are to be propagated from the queue. Enqueued messages will be propagated and automatically available for dequeuing at the destination queues.

Note that 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 wish to deploy remote propagation, you must ensure that the database link specified for the schedule is valid and have proper privileges for enqueuing into the destination queue. For more information about the administrative commands for managing propagation schedules, see "Propagation Scheduling" on page 8-108.

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 "Enhanced Propagation Scheduling Capabilities" 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 queues. If a queue has messages to be propagated to several queues, a schedule has to 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 may depend on a number of factors such as network traffic, load at source database, load at destination database, and so on. The schedule therefore has to 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 (see "Scheduling a Queue Propagation" in Chapter 9, "Administrative Interface"). The duration or propagation window parameter of a schedule specifies the time frame during which propagation has to take place. If the duration is unspecified, the time frame is an infinite single window. If a window has to be repeated periodically, a finite duration is specified along with a `next_time` function that defines the periodic interval between successive windows.

The latency parameter for a schedule is relevant only when a queue does not have any messages to be propagated. This parameter specifies the time interval within which a queue has to be rechecked for messages. Note that if the latency parameter is to be enforced, then the `job_queue_interval` parameter for the `job_queue_processes` should be less than or equal to the `latency` parameter.

The propagation schedules defined for a queue can be changed or dropped at anytime 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, it will take 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 (see "Selecting Propagation Schedules in User Schema" in Chapter 10, "Administrative Interface: Views").

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 OS/OS;
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 execute 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 AQ using two methods:

- Propagation from RAW queues. In RAW queues the message payload is stored as a Binary Large Object (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's LOB handling routines. The LOB attributes can be BLOBs or CLOBs (not NCLOBs). If the attribute is a CLOB AQ will automatically perform any necessary characterset conversion between the source queue and the destination queue. This method is supported by all releases from 8.1.3 inclusive.

For more information about working with LOBs, see:

- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

Note that AQ does not support propagation from Object queues that have BFILE or REF attributes in the payload.

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 LOBs.

When the order information is sent to the shipping warehouses, the coupon contents are also sent to the warehouses. In the code shown below the 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 has additional examples regarding for enqueue and dequeue of messages with LOBs as part the payload.

A COMMIT is issued 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 code below 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);

/* Issue 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 instance handling a schedule (relevant if Oracle 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

This includes 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 and the 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, 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, 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, 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, the propagation job is also migrated to the new instance. This will minimize pinging between instances and thus offer better performance. Propagation has been designed to handle any number of concurrent

schedules. Note that the number of job_queue_processes is limited to a maximum of 36 and some of these may be used to handle non-propagation related jobs. Hence, propagation has built in support for multi-tasking 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 queues.

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

Scenario

In the `BooksOnLine` example, the `OE_bookedorders_que` is a busy queue since 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, Advanced Queuing continues to attempt to propagate messages using an exponential backoff algorithm. In some situations that indicate application errors, AQ will mark messages as `UNDELIVERABLE` if there is an error in propagating the message.

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` will display 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 will return 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 will indicate 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 Oracle9*i*, you can set up Advanced Queuing propagation over HTTP and HTTPS (HTTP over SSL). HTTP propagation uses the Internet access infrastructure and requires that the 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 (formerly Net8) 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 will propagate to `WS_bookedorders_que`.

Propagation Setup

1. Deploy the 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 AQ Propagation Servlet. */
public class AQPropServlet extends oracle.AQ.xml.AQxmlServlet

/* getDBDrv - specify the database to which the servlet will connect */
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 Oracle9*i*, 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 will be used for authentication with the Web server / servlet runner as the host and port of the Web server running the 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))'
;
```

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 AQ HTTP agent used to access the AQ (propagation) servlet. `Welcome` is the password used to authenticate with the Web server.

3. Make sure that the AQ HTTP agent, John, is authorized to perform AQ operations. Do the following at the destination database.

   a. Register the AQ agent.

   ```
   dbms_aqadm.create_aq_agent(agent_name => 'John', enable_http => true);
   ```

   b. Map the 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 views, DBA_QUEUE_SCHEDULES, to check the propagation statistics for propagation schedules using HTTP.

9

Administrative Interface

This chapter describes the administrative interface to Oracle Advanced Queuing. We discuss each operation (such as "Creating a Queue Table") in terms of a use case by that name. Each use case is laid out as follows:

- *Use case figure*. A figure that depicts the use case.
- *Purpose*. The purpose of this use case.
- *Usage Notes*. Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment which illustrate the use case.

Use Case Model: Administrative Interface — Basic Operations

Table 9–1, "Use Case Model: Administrative Interface — Basic Operations" indicates with a + where examples are provided for specific use cases and in which programmatic environment.

The table refers to programmatic environments with the following abbreviations:

- **P** — PL/SQL using the DBMS_AQADM and DBMS_AQ packages
- **O** — C using OCI (Oracle Call Interface)
- **V** — Visual Basic using OO4O (Oracle Objects for OLE)
- **J** — Java (native AQ) using JDBC (Java Database Connectivity)
- **JM** — Java (JMS standard) using JDBC (Java Database Connectivity)

Table 9–1 Use Case Model: Administrative Interface — Basic Operations

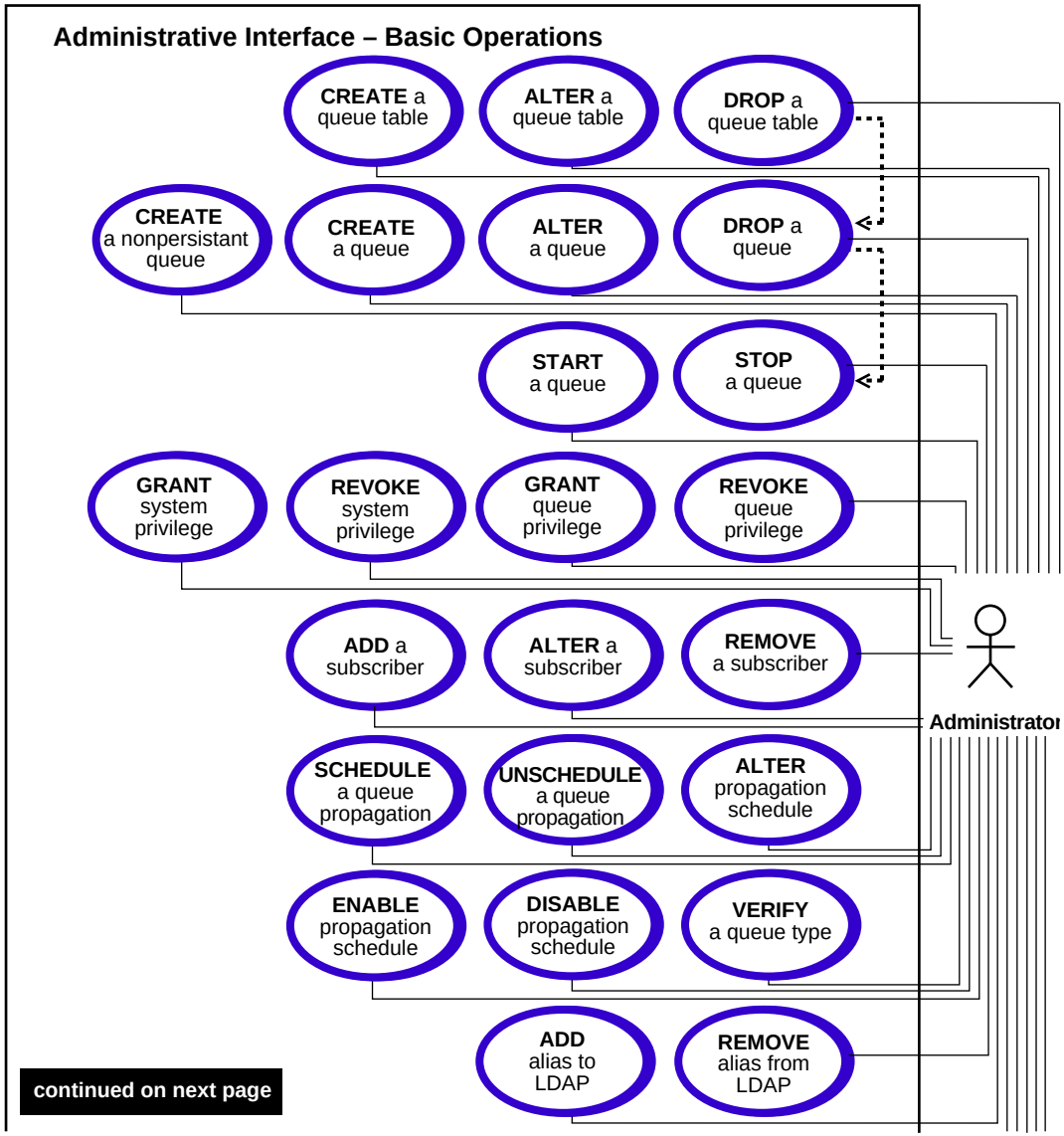
Use Case	Programmatic Environment Examples				
	P	O	V	J	JM
Creating a Queue Table on page 9-6	+		+	+	
Creating a Queue Table [Set Storage Clause] on page 9-15	+			+	
Altering a Queue Table on page 9-16	+			+	
Dropping a Queue Table on page 9-19	+			+	
Creating a Queue on page 9-22	+			+	
Creating a Nonpersistent Queue on page 9-28	+				
Altering a Queue on page 9-31	+			+	
Dropping a Queue on page 9-34	+			+	
Creating a Transformation on page 9-37					
Modifying a Transformation on page 9-40					
Applying a Transformation on page 9-42					
Dropping a Transformation on page 9-43					
Starting a Queue on page 9-44	+			+	
Stopping a Queue on page 9-47	+			+	
Granting System Privilege on page 9-50	+			+	

Table 9–1 Use Case Model: Administrative Interface — Basic Operations

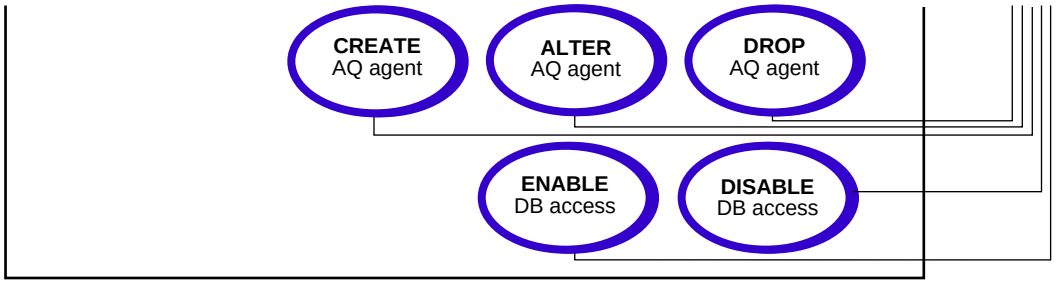
Use Case	Programmatic Environment Examples				
	P	O	V	J	JM
Revoking System Privilege on page 9-53	+				
Granting Queue Privilege on page 9-55	+			+	
Revoking Queue Privilege on page 9-58	+			+	
Adding a Subscriber on page 9-61	+			+	
Altering a Subscriber on page 9-67	+			+	
Removing a Subscriber on page 9-71	+			+	
Scheduling a Queue Propagation on page 9-74	+			+	
Unscheduling a Queue Propagation on page 9-78	+			+	
Verifying a Queue Type on page 9-81	+				
Altering a Propagation Schedule on page 9-84	+			+	
Enabling a Propagation Schedule on page 9-88	+			+	
Disabling a Propagation Schedule on page 9-91	+			+	
Creating an AQ Agent on page 9-94					
Altering an AQ Agent on page 9-97					
Dropping an AQ Agent on page 9-99					
Enabling Database Access on page 9-101					
Disabling Database Access on page 9-103					
Adding an Alias to the LDAP Server on page 9-105					
Removing an Alias from the LDAP Server on page 9-107					

Figure 9–1 lists all use cases of the administrative interface.

Figure 9–1 Use Case Diagram: Administrative Interface — Basic Operations

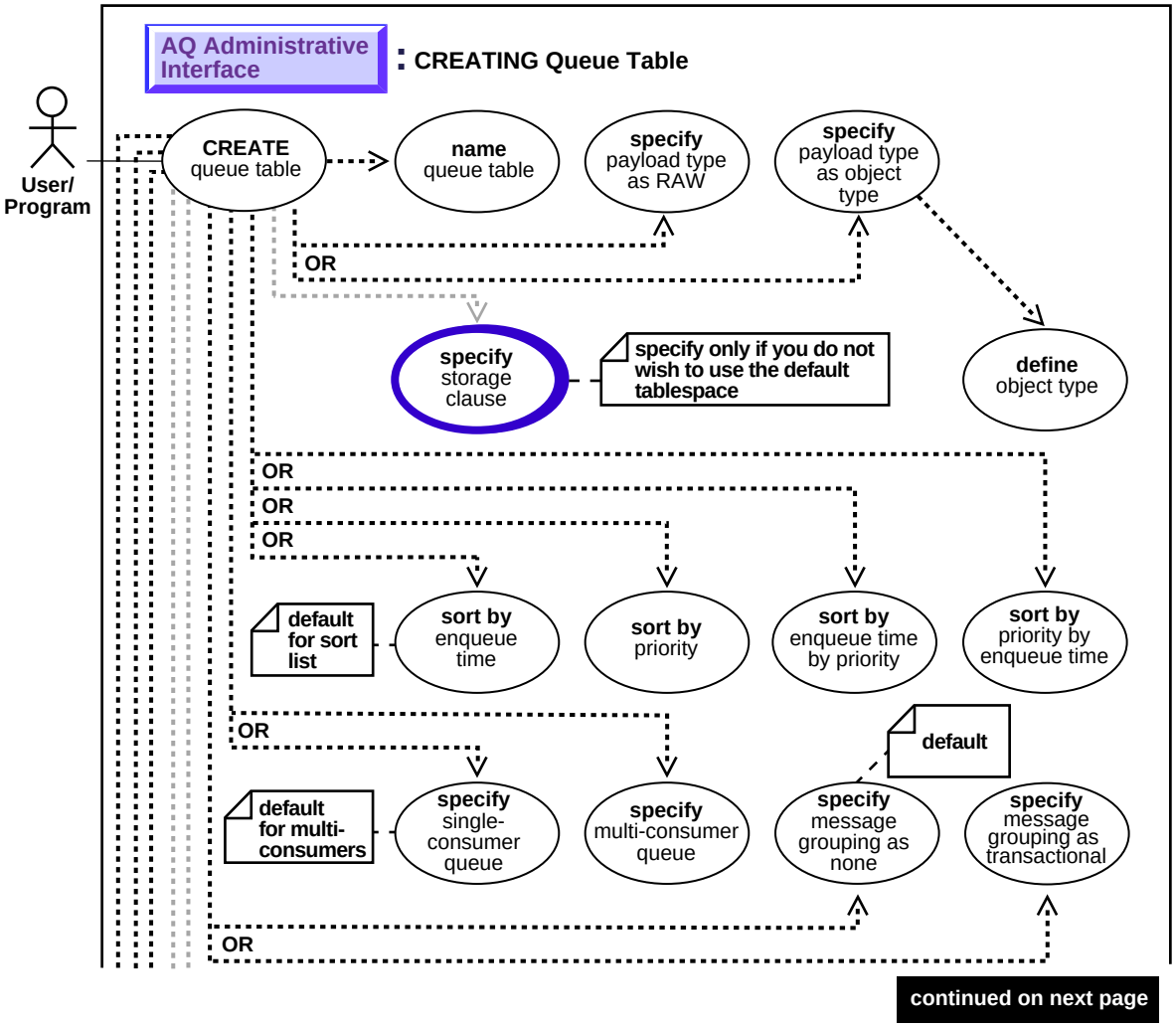


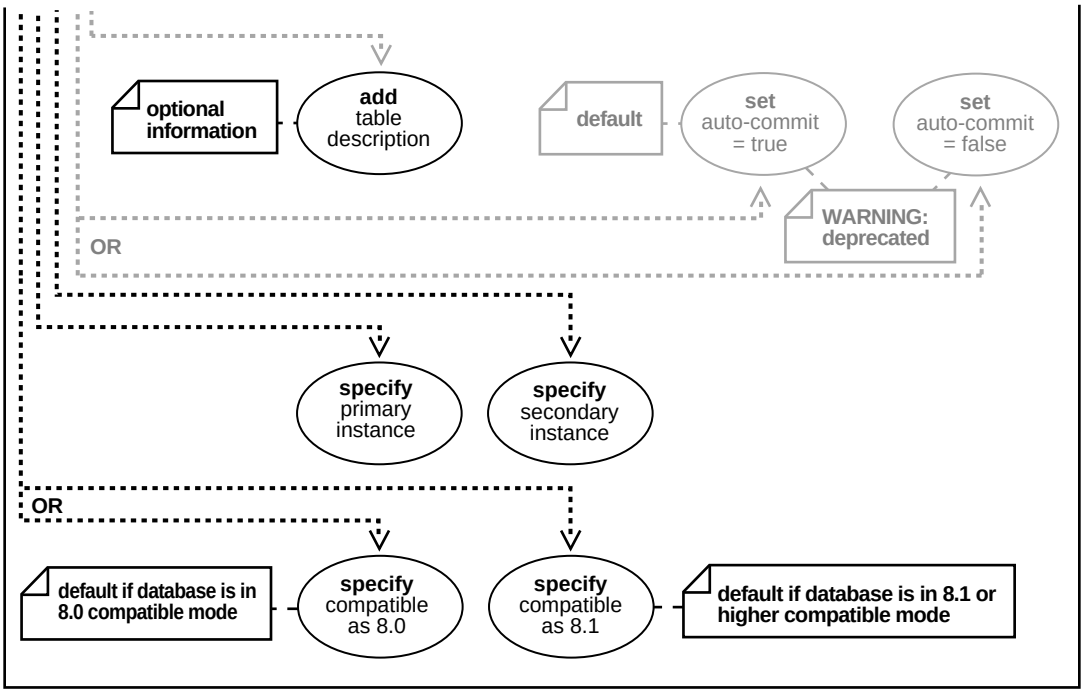
continued on next page


CREATE
AQ agent
ALTER
AQ agent
DROP
AQ agent
ENABLE
DB access
DISABLE
DB access

Creating a Queue Table

Figure 9–2 Use Case Diagram: Creating a Queue Table





To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Create a queue table for messages of a predefined type.

Usage Notes

- Queue names and queue table names are converted to upper case. Mixed case (upper and lower case together) is not supported.
- The default value of the compatible parameter depends on the database compatibility mode in the init.ora

  - If the database is in 8.1 or higher compatible mode, the default value is '8.1'
  - If the database is in 8.0 compatible mode, the default value is '8.0'
- The sort keys for dequeue ordering, if any, need to be defined at table creation time. The following objects are created at this time:
  - The default exception queue associated with the queue table called `aq$_<queue_table_name>_e`.
  - A read-only view which is used by AQ applications for querying queue data called `aq$<queue_table_name>`.
  - An index for the queue monitor operations called `aq$_<queue_table_name>_t`.
  - An index or an index organized table (IOT) in the case of multiple consumer queues for dequeue operations called `aq$_<queue_table_name>_i`.
- For 8.1-compatible multiconsumer queue tables, the following additional objects are created:
  - A table called `aq$_<queue_table_name>_s`. This table stores information about the subscribers.
  - A table called `aq$_<queue_table_name>_r`. This table stores information about rules on subscriptions.
  - An index organized table (IOT) called `aq$_<queue_table_name>_h`. This table stores the dequeue history data.
- `CLOB`, `BLOB`, or `BFILE` objects are valid in an AQ message. You can propagate these object types using AQ propagation with Oracle since release 8.1.x. To enqueue an object type that has an `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. See the *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information.
- You can specify and modify the primary_instance and secondary_instance only in 8.1 compatible mode.
- You cannot specify a secondary instance unless there is a primary instance.
- When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, CREATE_QUEUE_TABLE
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ *createQueueTable*

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.:

- PL/SQL (DBMS_AQADM Package): Creating a Queue Table on page 9-10
- VB (OO4O): Creating a Queue Table on page 9-12
- Java (JDBC): Creating a Queue Table on page 9-12

PL/SQL (DBMS_AQADM Package): Creating a Queue Table

Note: You may need to 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;
```

Create queue table for queues containing messages of object type

```
CREATE type aq.Message_typ as object (
   Subject                VARCHAR2(30),
   Text                   VARCHAR2(80));

/* Note: if you do not stipulate a schema, you default to the user's schema. */
EXECUTE dbms_aqadm.create_queue_table (
   Queue_table            => 'aq.ObjMsgs_qtab',
   Queue_payload_type     => 'aq.Message_typ');
```

Create queue table for queues containing messages of RAW type

```
EXECUTE dbms_aqadm.create_queue_table (
   Queue_table            => 'aq.RawMsgs_qtab',
   Queue_payload_type     => 'RAW');
```

Create a queue table for queues containing messages of object type with XMLType attributes

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
   orderno  NUMBER,
   details  XMLTYPE);

execute dbms_aqadm.create_queue_table(
        queue_table     => 'OS_orders_pr_mqtab',
        comment         => 'Overseas Shipping MultiConsumer Orders queue table',
        multiple_consumers  => TRUE,
```

```
        queue_payload_type  => 'OS.order_xml_typ',
        compatible          => '8.1');
```

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 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’);
```

Create 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’);
```

Create 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');
```

Create 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;
```

VB (OO4O): Creating a Queue Table

OO4O uses database functionality for this operation.

Java (JDBC): Creating a Queue Table

Three examples follow of how to create a queue table using Java.

Note: You may need to 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;

CREATE type aq.Message_typ as object (
   Subject                VARCHAR2(30),
   Text                   VARCHAR2(80));
```

Create queue table for queues containing messages of object type

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

    /* Create a AQQueueTableProperty object (payload type Message_typ): */
    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");

    /* Create a queue table in aq schema */
    q_table = aq_sess.createQueueTable ("aq", "ObjMsgs_qtab", qtable_prop);

    System.out.println("Successfully created ObjMsgs_qtab in aq schema");
}
```

Create queue table for queues containing messages of RAW type

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

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

    /* Create a queue table in aq schema */
    q_table = aq_sess.createQueueTable ("aq", "RawMsgs_qtab", qtable_prop);

    System.out.println("Successfully created RawMsgs_qtab in aq schema");
}
```

3. Create a queue table for multiple consumers and prioritized messages

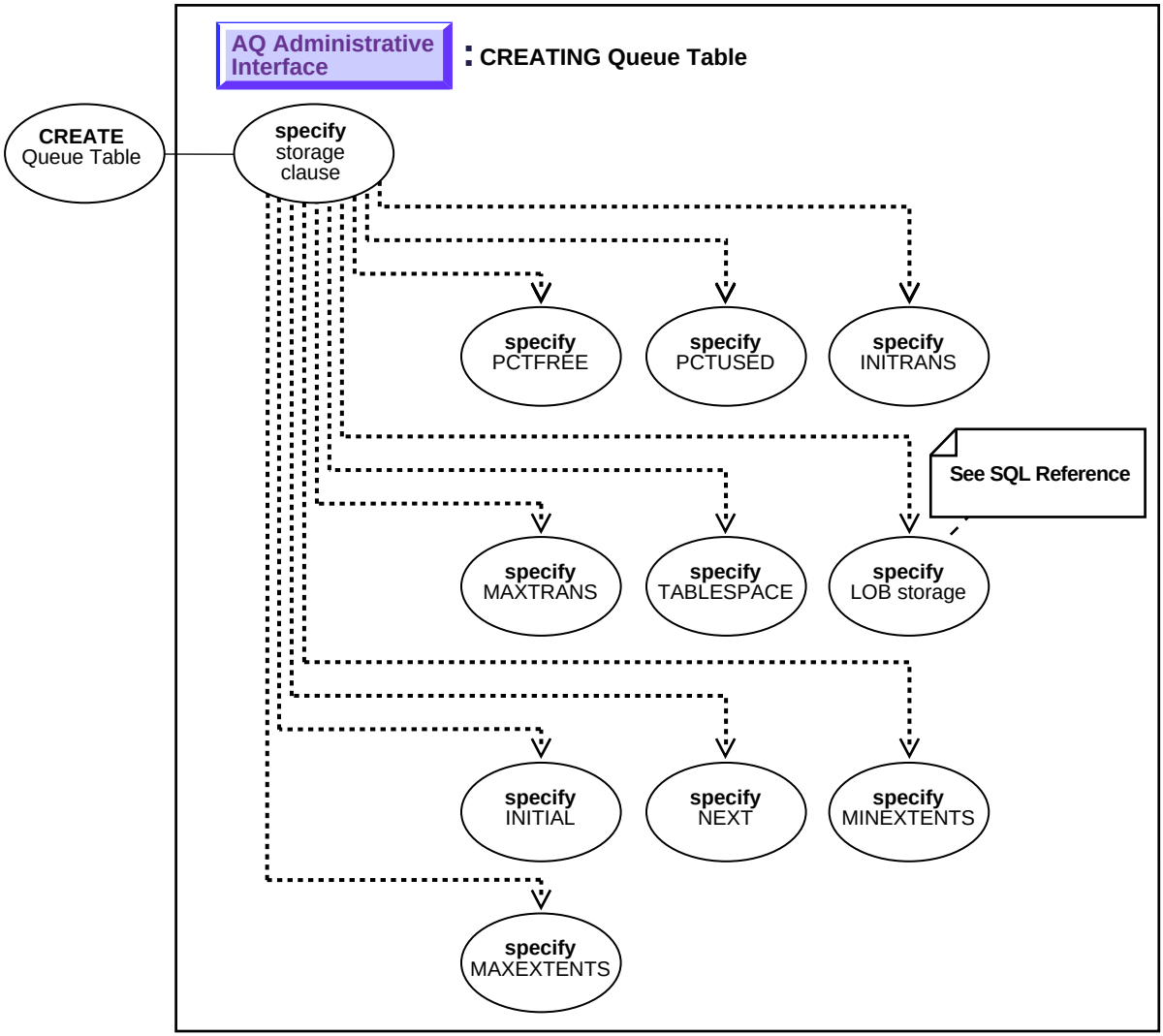
```
public static void example(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty     qtable_prop;
    AQQueueProperty          queue_prop;
    AQQueueTable             q_table;
    AQQueue                  queue;

    qtable_prop = new AQQueueTableProperty("RAW");

    /* Enable multiple consumers */
    qtable_prop.setMultiConsumer(true);
    qtable_prop.setCompatible("8.1");

    /* Specify sort order as priority,enqueue_time */
    qtable_prop.setSortOrder("PRIORITY,ENQ_TIME");

    /* Create a queue table in aq schema */
    q_table = aq_sess.createQueueTable ("aq", "PriorityMsgs_qtab",
 qtable_prop);

    System.out.println("Successfully created PriorityMsgs_qtab in aq schema");
}
```

Create queue table in specified tablespace

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

    /* Create a AQQueueTableProperty object (payload type Message_typ): */
    qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");

    /* Specify tablespace for queue table */
    qtable_prop.setStorageClause("tablespace aq_tbs");

    /* Create a queue table in aq schema */
    q_table = aq_sess.createQueueTable ("aq", "aq_tbsMsg_qtab", qtable_prop);
}
```

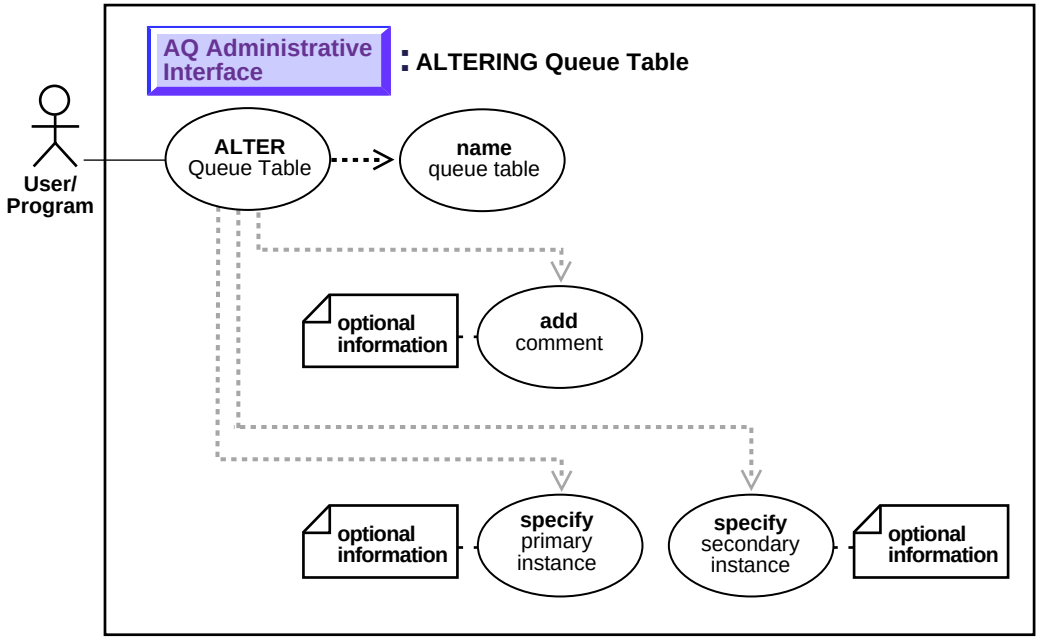
Creating a Queue Table [Set Storage Clause]

Figure 9–3 Use Case Diagram: Create a Queue Table [Set Storage Clause]



Altering a Queue Table

Figure 9–4 Use Case Diagram: Altering a Queue Table



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Alter the existing properties of a queue table.

Usage Notes

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ALTER_QUEUE_TABLE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ,alterQueue

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments.

- PL/SQL (DBMS_AQADM Package): Altering a Queue Table on page 9-17
- VB (OO4O): Example not provided.
- Java (JDBC): Altering a Queue Table on page 9-18

PL/SQL (DBMS_AQADM Package): Altering a Queue Table

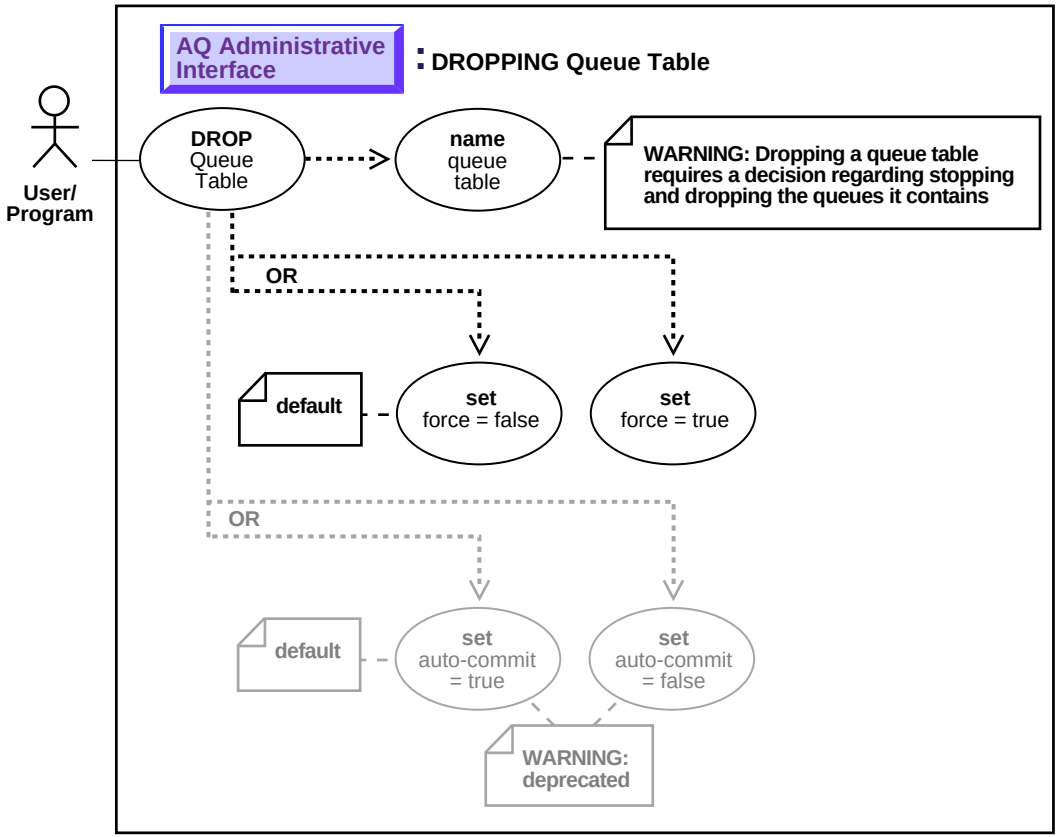
```
/* Altering the table to change the primary, secondary instances for queue owner
(only applies to Real Application Clusters environments). The primary instance
is the instance number of the primary owner of the queue table. The secondary
instance is the instance number of the secondary owner of the queue table. */
 EXECUTE dbms_aqadm.alter_queue_table (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Primary_instance     => 3,
   Secondary_instance   => 2);

/* Altering the table to change the comment for a queue table: */
EXECUTE dbms_aqadm.alter_queue_table (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Comment              => 'revised usage for queue table');

/* Altering the table to change the comment for a queue table and use
nonrepudiation: */
EXECUTE dbms_aqadm.alter_queue_table (
   Queue_table          => 'aq.ObjMsgs_qtab',
   Comment              => 'revised usage for queue table',
```

Java (JDBC): Altering a Queue Table

```
/* Alter a queue table */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueueTableProperty    qtable_prop;
     AQQueueTable            q_table;

     q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

     /* Get queue table properties: */
     qtable_prop = q_table.getProperty();

     /* Alter the queue table comment and instance affinity */
     q_table.alter("altered queue table", 3, 2);
}
```

Dropping a Queue Table

Figure 9–5 Use Case Diagram: Drop a Queue Table



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Drop an existing queue table. Note that you must stop and drop all the queues in a queue tables before the queue table can be dropped. You must do this explicitly unless the `force` option is used in which case this done automatically.

Usage Notes

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, DROP_QUEUE_TABLE procedure.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ AQQueueTable.drop

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments.

- PL/SQL (DBMS_AQADM Package): Dropping a Queue Table on page 9-20
- VB (OO4O): Example not provided.
- Java (JDBC): Dropping a Queue Table on page 9-21

PL/SQL (DBMS_AQADM Package): Dropping a Queue Table

```
/* Drop the queue table (for which all queues have been previously dropped by
   the user) */
EXECUTE dbms_aqadm.drop_queue_table (
   queue_table        => 'aq.Objmsgs_qtab');
```

Caution: You may need to set up or drop data structures for certain examples to work:

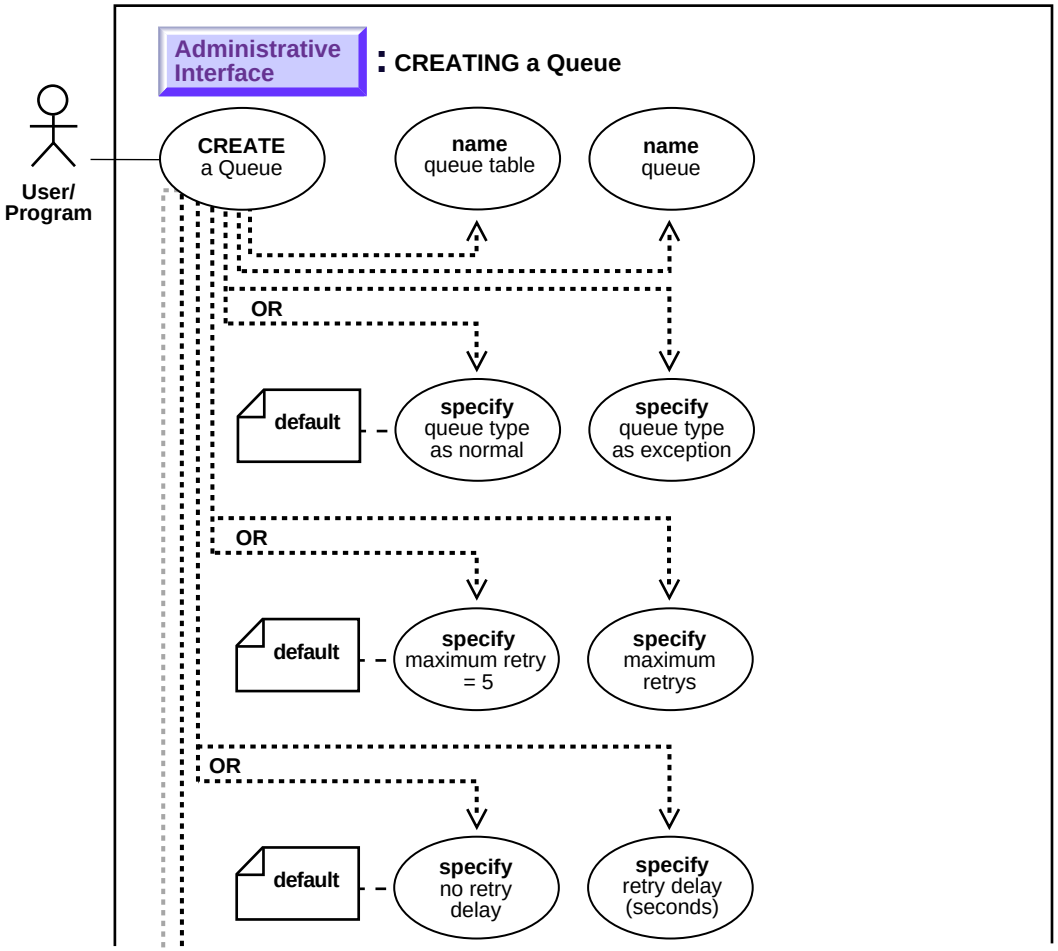
```
/* Drop the queue table and force all queues to be stopped and dropped by the
   system */
EXECUTE dbms_aqadm.drop_queue_table (
   queue_table        => 'aq.Objmsgs_qtab',
   force              => TRUE);
```

Java (JDBC): Dropping a Queue Table

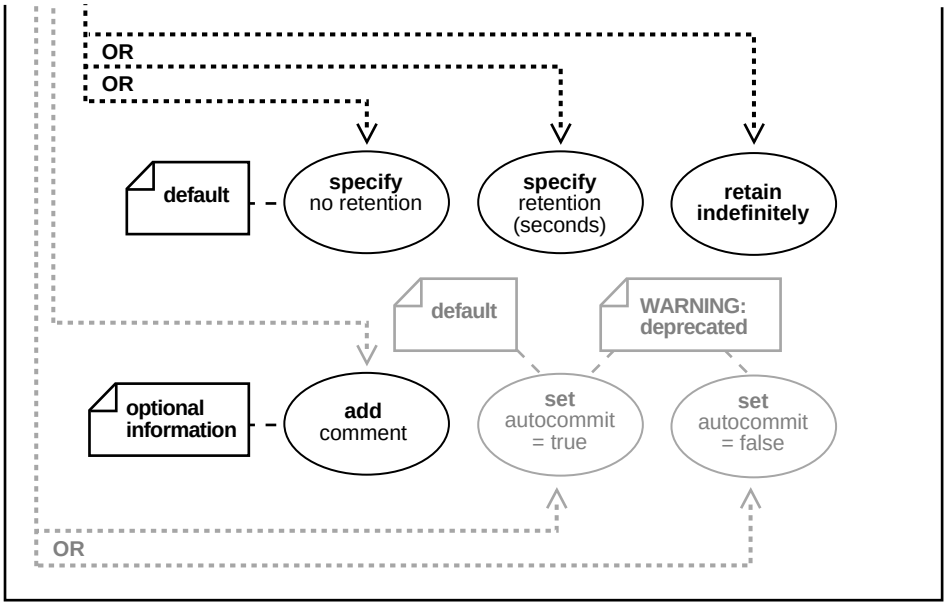
```
/* Drop a queue table - for which all queues have already been dropped by
   the user */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueueTable            q_table;

     q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

     /* Drop the queue table*/
     q_table.drop(false);
     System.out.println("Successful drop");
}


/* Drop the queue table (and force all queues to be stopped and dropped by
   the user */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueueTable            q_table;

     q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

     /* Drop the queue table (and automatically drop all queues inside it */
     q_table.drop(true);
     System.out.println("Successful drop");
}
```

Creating a Queue

Figure 9–6 Use Case Diagram: Creating a Queue



continued on next page



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Create a queue in the specified queue table.

Usage Notes

- Queue names and queue table names are converted to upper case. Mixed case (upper and lower case together) is not supported.
- 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.

- When a queue, queue table, or subscriber is created and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, CREATE_QUEUE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, CreateQueue

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments.

- PL/SQL (DBMS_AQADM Package): Dropping a Queue Table on page 9-20
- VB (OO4O): Example not provided.
- Java (JDBC): Dropping a Queue Table on page 9-21

PL/SQL (DBMS_AQADM): Creating a Queue

Create 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');
```

Create 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');
```

Create a prioritized message queue table and queue

Caution: You may need to set up or drop data structures for certain examples to work:

```
/* Create a queue table for priortized 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');
```

Create a queue table and queue meant for multiple consumers

Caution: You may need to set up or drop data structures for certain examples to work:

```
/* Create a queue table for multi-consumers: */
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');
```

Create a queue table and queue to demonstrate propagation

```
/* Create queue: */
EXECUTE dbms_aqadm.create_queue (
   Queue_name        => 'AnotherMsg_queue',
   queue_table       => 'aq.MultiConsumerMsgs_qtab');
```

Create a queue table and queue for multiple consumers compatible with 8.1

```
/* Create a queue table for multi-consumers 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');

EXECUTE dbms_aqadm.create_queue (
   Queue_name         => 'MultiConsumerMsg81_queue',
   Queue_table        => 'aq.MultiConsumerMsgs81_qtab');
```

Java (JDBC): Creating a Queue

Create a queue within a queue table for messages of object type

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

     q_table = aq_sess.getQueueTable ("aq", "ObjMsgs_qtab");

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

     queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);
     System.out.println("Successful createQueue");

}
```

Create a queue within a queue table for messages of raw type

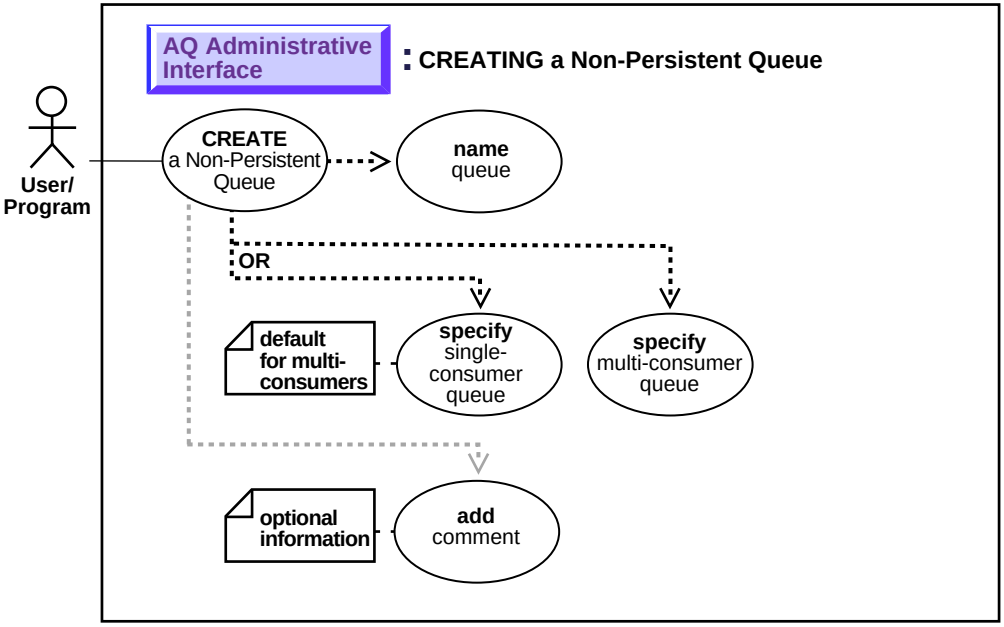
```
public static void example(AQSession aq_sess) throws AQException
{
     AQQueueProperty         queue_prop;
     AQQueueTable            q_table;
     AQQueue                 queue;

     q_table = aq_sess.getQueueTable ("aq", "RawMsgs_qtab");

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

     queue = aq_sess.createQueue (q_table, "msg_queue", queue_prop);
     System.out.println("Successful createQueue");

}
```

Create a Multi-Consumer queue with prioritized messages

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

     qtable_prop = new AQQueueTableProperty("RAW");
     qtable_prop.setMultiConsumer(true);

     qtable_prop.setSortOrder("priority,enq_time");
     q_table = aq_sess.createQueueTable ("aq", "PriorityMsgs_qtab",
 qtable_prop);

     queue_prop = new AQQueueProperty();
     queue = aq_sess.createQueue (q_table, "priority_msg_queue", queue_prop);

}
```

Creating a Nonpersistent Queue

Figure 9–7 Use Case Diagram: Creating a Nonpersistent Queue



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Create a nonpersistent queue.

Usage Notes

The queue may be either single-consumer or multiconsumer queue. 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, 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 Object Type (ADT) 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 OCI notification mechanism (see Registering for Notification on page 11-58).

You cannot invoke the `listen` call on a nonpersistent queue (see Listening to One (Many) Queue(s) on page 11-24).

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, CREATE_NP_QUEUE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): There is no applicable syntax reference for this use case

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments.

- PL/SQL (DBMS_AQADM Package): Dropping a Queue Table on page 9-20
- VB (OO4O): Example not provided.
- Java (JDBC): Dropping a Queue Table on page 9-21

PL/SQL (DBMS_AQADM): Creating a Nonpersistent Queue

```
/* Create a nonpersistent single-consumer queue (Note: this is not preceded by
   creation of a queue table) */
EXECUTE dbms_aqadm.create_np_queue(
   Queue_name          => 'Singleconsumersmsg_npque',
   Multiple_consumers  => FALSE);
```

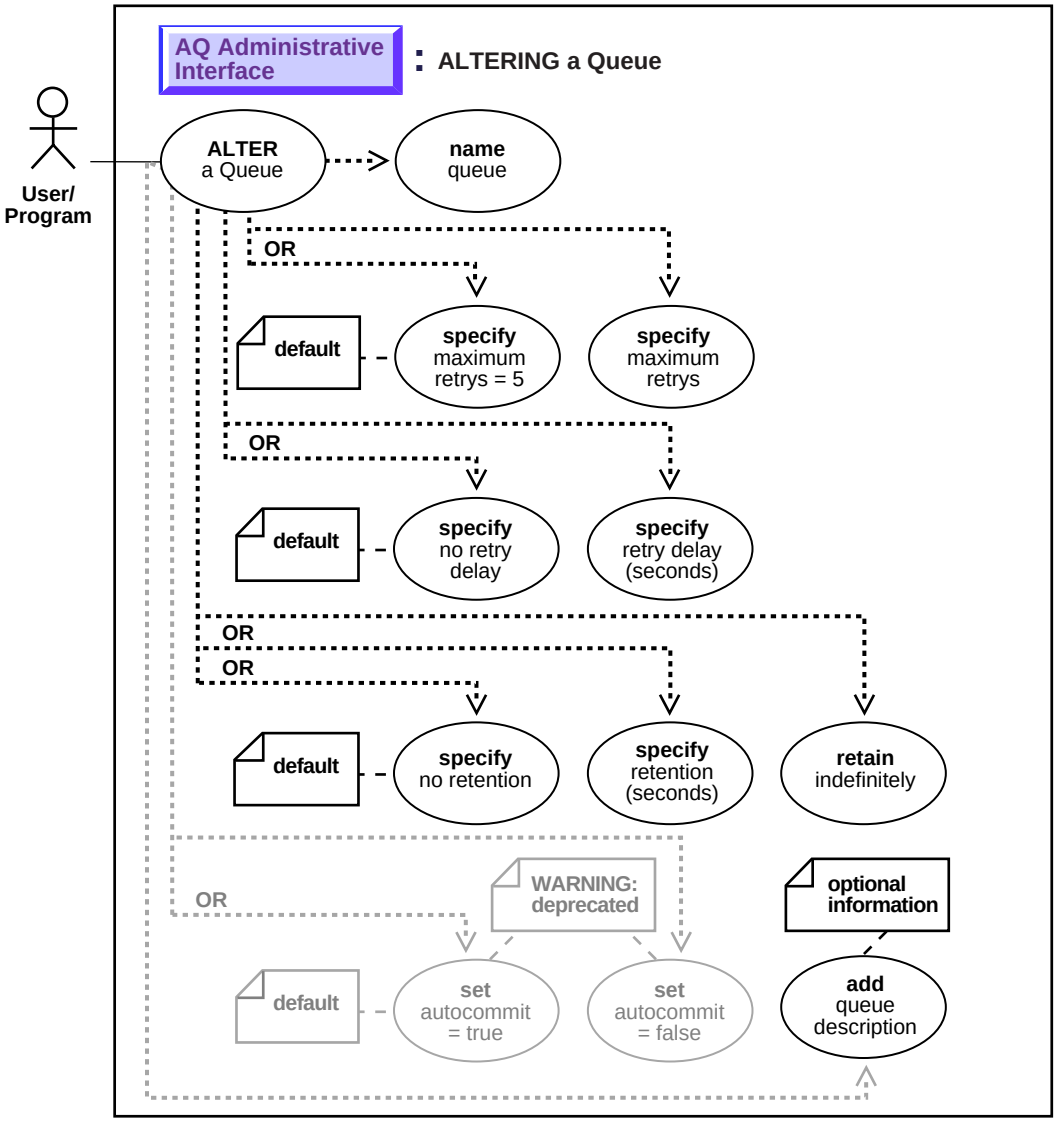
```
/* Create a nonpersistent multi-consumer queue (Note: this is not preceded by
   creation of a queue table) */
EXECUTE dbms_aqadm.create_np_queue(
   Queue_name          => 'Multiconsumersmsg_npque',
   Multiple_consumers   => TRUE);
```

Java (JDBC): Creating a Nonpersistent Queue

Feature not available through Java API.

Altering a Queue

Figure 9–8 Use Case Diagram: Alter a Queue



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Alter existing properties of a queue. Only max_retries, comment, retry_delay, and retention_time can be altered.

Usage Notes

To view retained messages, you can either dequeue by message ID or use SQL.

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ALTER_QUEUE_TABLE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, alter

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Altering a Queue on page 9-33
- VB (OO4O): Example not provided.
- Java (JDBC): Altering a Queue on page 9-33

PL/SQL (DBMS_AQADM): Altering a Queue

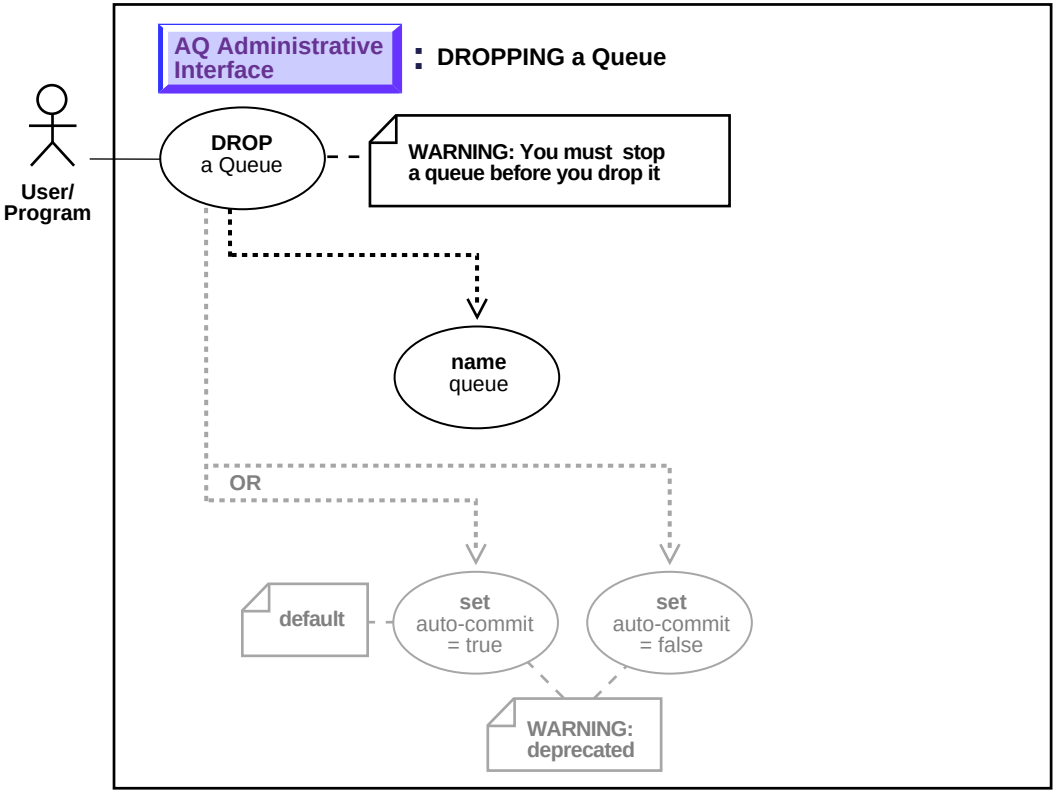
```
/* Alter queue to change retention time, saving messages for 1 day after
   dequeueing:  */
EXECUTE dbms_aqadm.alter_queue (
   queue_name        => 'aq.Anothermsg_queue',
   retention_time    => 86400);
```

Java (JDBC): Altering a Queue

```
/* Alter a queue to change retention time, saving messages for 1 day
   after dequeuing */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueueProperty         queue_prop;
     AQQueue                 queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "Anothermsg_queue");

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

     /* Change retention time to 1 day */
     queue_prop.setRetentionTime(new Double(86400));

     /* Alter the queue */
     queue.alterQueue(queue_prop);

}
```

Dropping a Queue

Figure 9–9 Use Case Diagram: Drop a Queue



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

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.

Usage Notes

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, DROP_QUEUE_TABLE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, dropQueue

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Creating a Transformation on page 9-38
- VB (OO4O): Example not provided.
- Java (JDBC): Dropping a Queue on page 9-36

PL/SQL (DBMS_AQADM): Dropping a Queue

Drop a Standard Queue

```
/* Stop the queue preparatory to dropping it (a queue may be dropped only after
   it has been succesfully stopped for enqueing and dequeing): */
EXECUTE dbms_aqadm.stop_queue (
   Queue_name        => 'aq.Msg_queue');

/* Drop queue: */
EXECUTE dbms_aqadm.drop_queue (
   Queue_name         => 'aq.Msg_queue');
```

Drop a Nonpersistent Queue

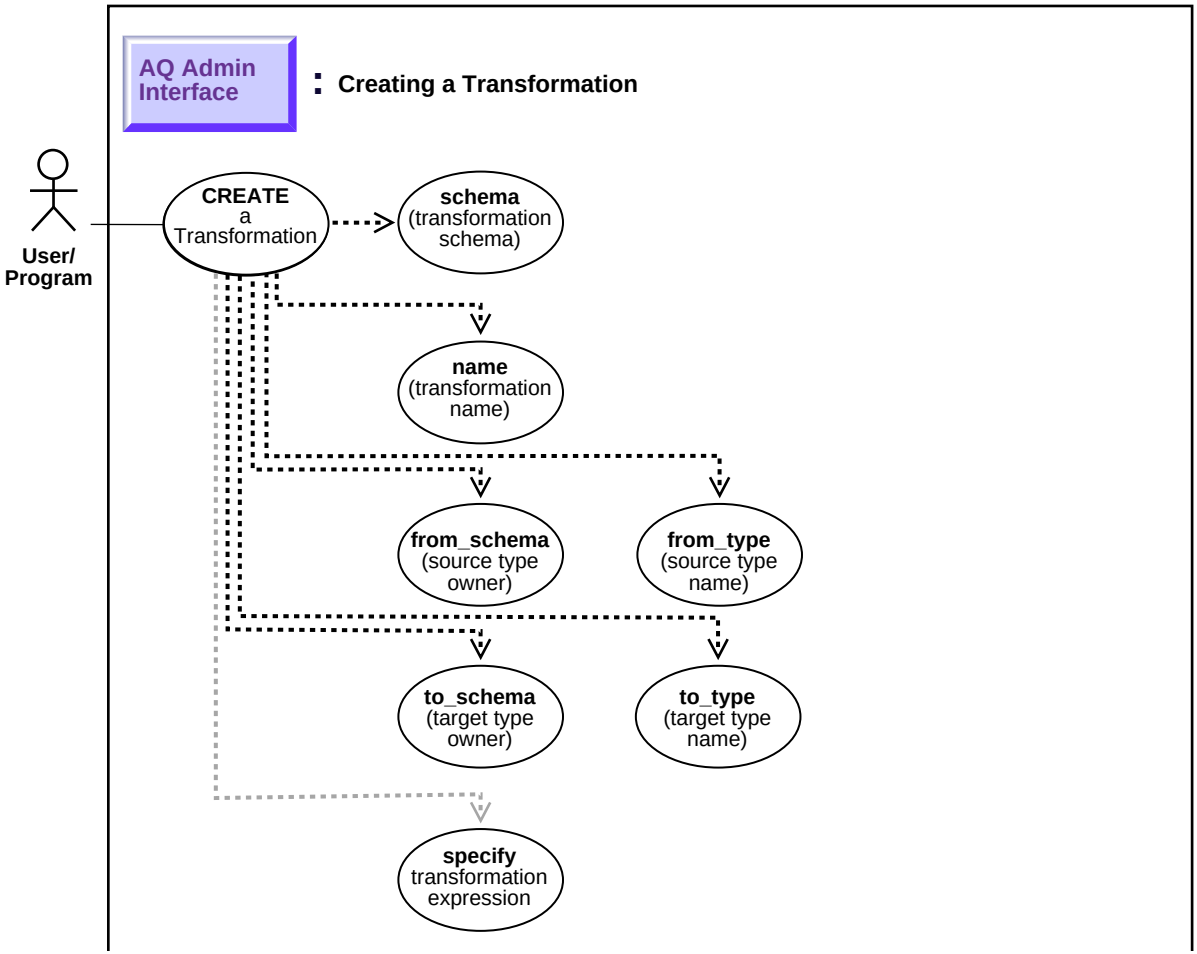
```
EXECUTE DBMS_AQADM.DROP_QUEUE( queue_name => 'Nonpersistent_singleconsumerq1');
EXECUTE DBMS_AQADM.DROP_QUEUE( queue_name => 'Nonpersistent_multiconsumerq1');
```

Java (JDBC): Dropping a Queue

```
/* Drop a queue */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue                queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "Msg_queue");

     /* Stop the queue first */
     queue.stop(true);

     /* Drop the queue */
     queue.drop();
}
```

Creating a Transformation

Figure 9–10 Use Case Diagram: 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`.

Usage Notes

To use this feature, you must be granted execute privileges on `dbms_transform`. 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) or commit or rollback the current transaction.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, DROP_QUEUE_TABLE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Creating a Transformation on page 9-38
- VB (OO4O): Example not provided.
- Java (JDBC): Dropping a Queue on page 9-36

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',
```

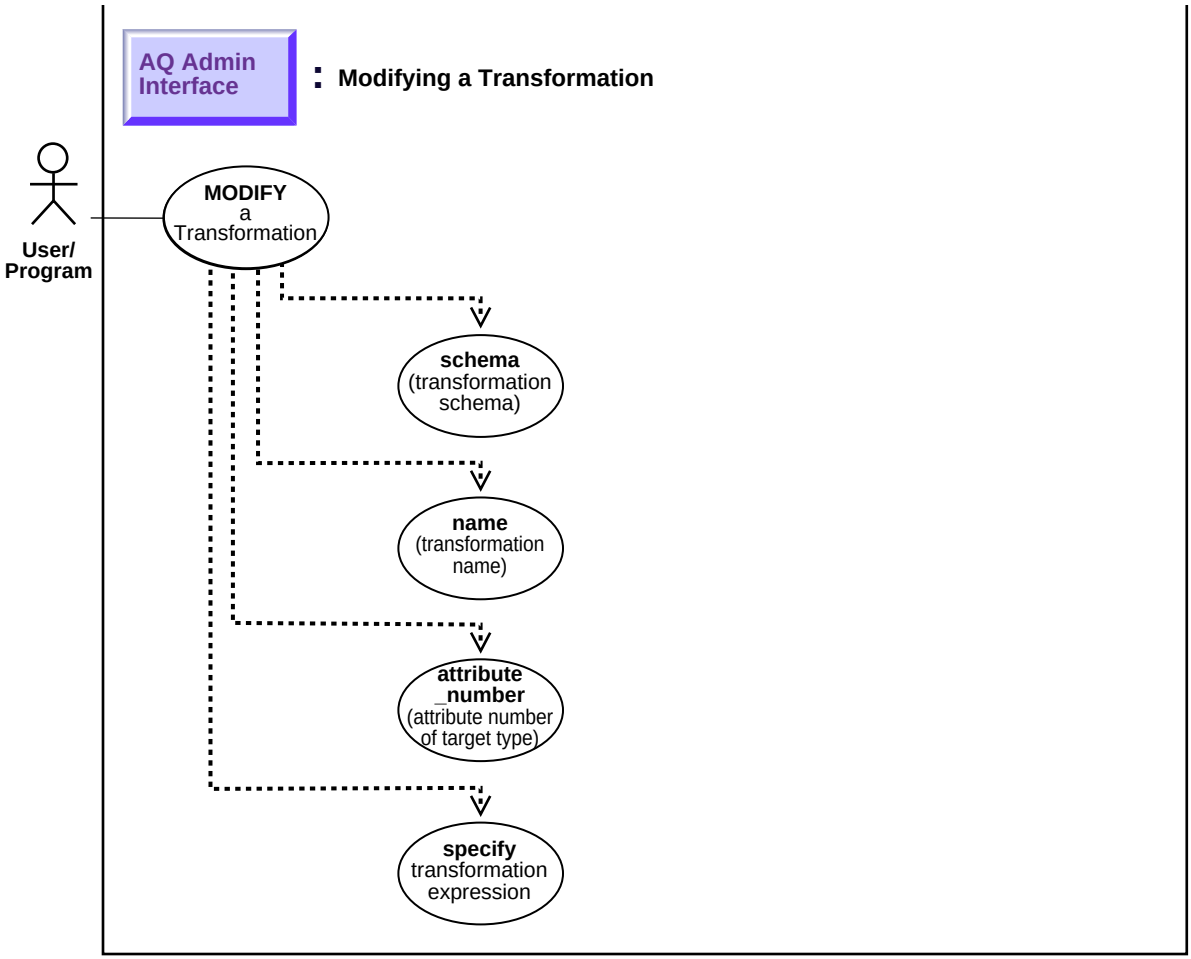
```
from_schema    => 'scott',
from_type      => 'type1,
to_schema      => 'scott',
to_type        => 'type2',
transformation => 'scott.type2(source.user_data.attr2,
    source.user_data.attr1)');
```

Java (JDBC)

No example is provided with this release.

Modifying a Transformation

Figure 9–11 Use Case Diagram: Modifying a Transformation



Purpose

This feature is used to change the transformation function and to specify transformations per an attribute of the target type. If the attribute number 0 is specified, then the transformation expression singularly defines the transformation

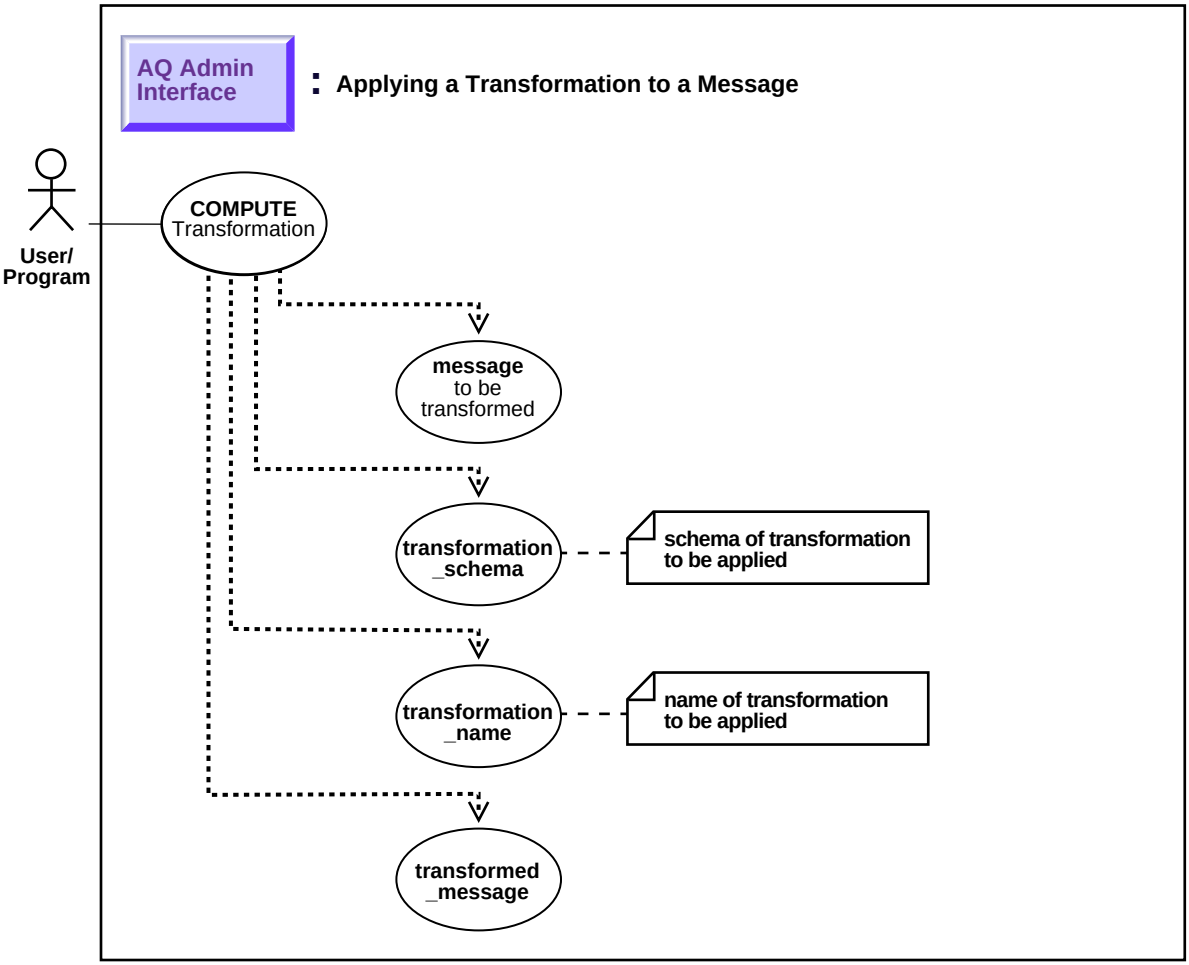
from the source to target types. All references to `from_type` must be of the form `source.user_data`. All references to the attributes of the source type must be prefixed by `source.user_data`.

Usage Notes

To use this feature, you must be granted execute privileges on `dbms_transform`. 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.

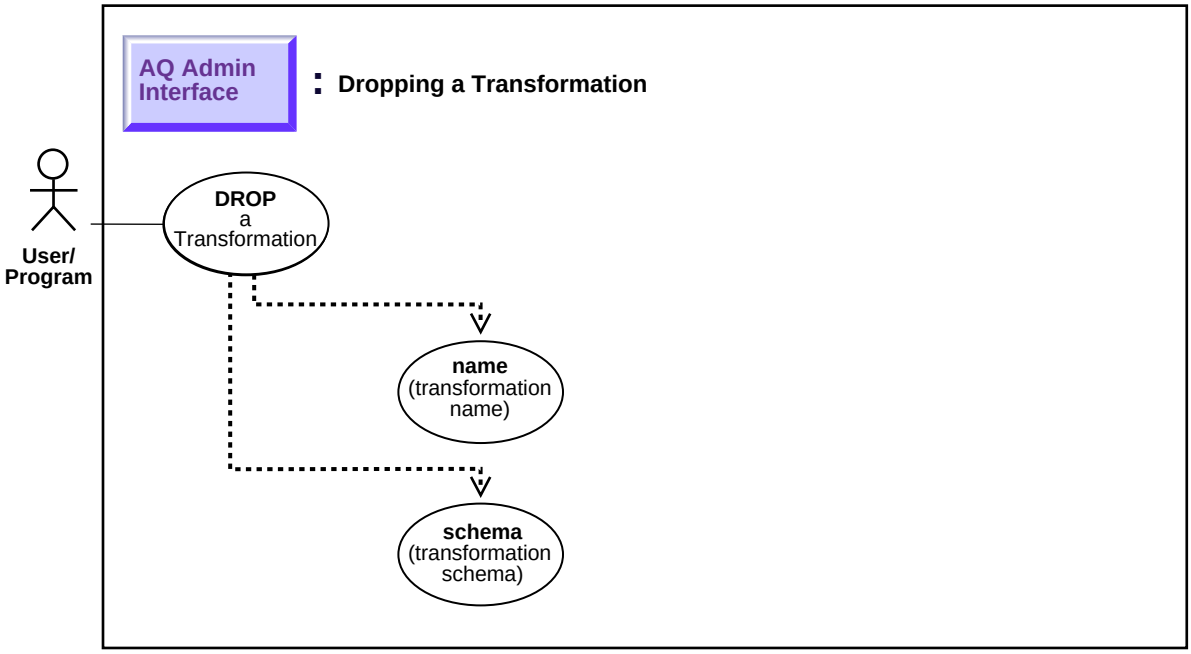
Applying a Transformation

Figure 9–12 Applying a Transformation



Dropping a Transformation

Figure 9–13 Use Case Diagram: Dropping a Transformation



Purpose

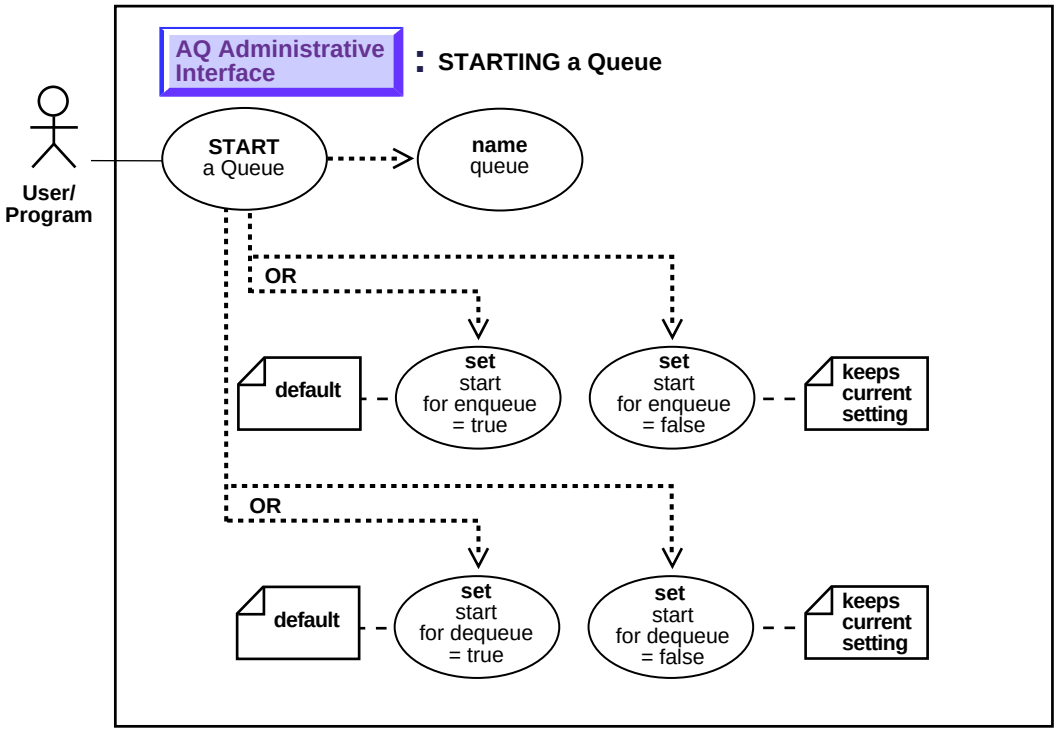
To drop a transformation.

Usage Notes

To use this feature, you must be granted execute privileges on `dbms_transform`. 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.

Starting a Queue

Figure 9–14 Use Case Diagram: Start a Queue



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Enables the specified queue for enqueuing and / or dequeueing.

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

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, START_QUEUE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, AQQueueAdmin.start

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM Package): Starting a Queue on page 9-45
- VB (OO4O): Example not provided.
- Java (JDBC): Starting a Queue on page 9-46

PL/SQL (DBMS_AQADM Package): Starting a Queue

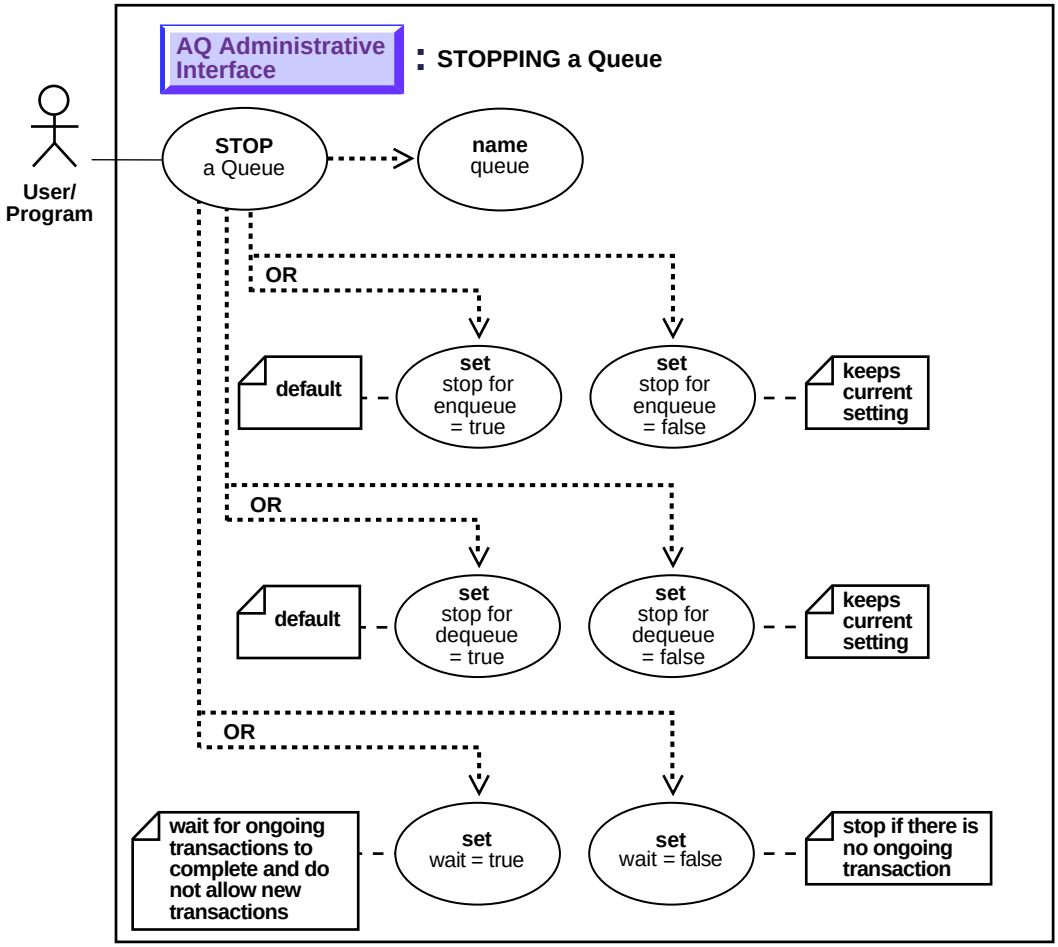
```
/* Start a queue and enable both enqueue and dequeue: */
EXECUTE dbms_aqadm.start_queue (
   queue_name          => 'Msg_queue');

/* Start a previously stopped queue for dequeue only */
EXECUTE dbms_aqadm.start_queue (
   queue_name          => 'aq.msg_queue',
   dequeue             => TRUE,
   enqueue             => FALSE);
```

Java (JDBC): Starting a Queue

```
/* Start a queue - enable both enqueue and dequeue */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue                 queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "Msg_queue");

     /* Enable enqueue and dequeue */
     queue.start();
}


/* Start a previously stopped queue for dequeue only */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue                 queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "Msg_queue");

     /* Enable enqueue and dequeue */
     queue.start(false, true);
}
```

Stopping a Queue

Figure 9–15 Use Case Diagram: Stop a Queue



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Disables enqueuing or dequeuing on the specified queue.

Usage Notes

By default, this call disables both ENQUEUEs or DEQUEUEs. 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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, STOP_QUEUE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ*)* AQQueueAdmin.stop

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Stopping a Queue on page 9-48
- VB (OO4O): Example not provided.
- Java (JDBC): Stopping a Queue on page 9-49

PL/SQL (DBMS_AQADM): Stopping a Queue

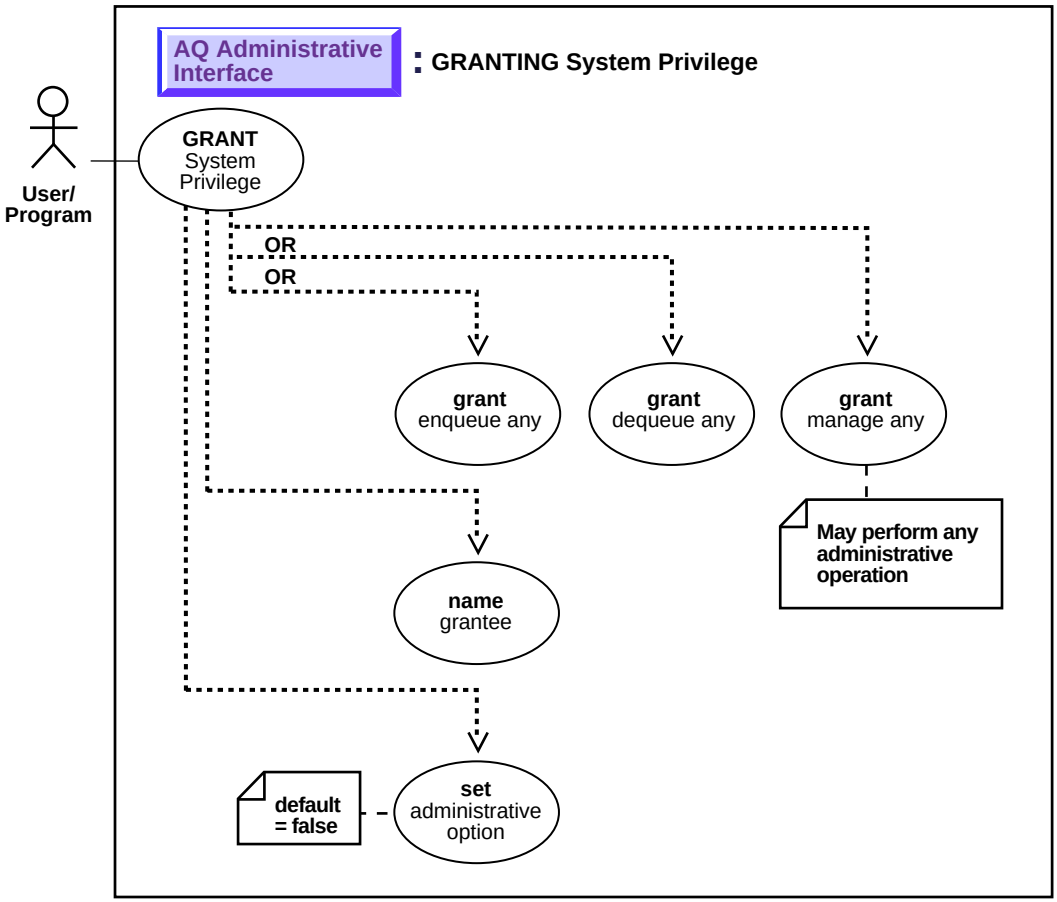
```
/* Stop the queue: */
EXECUTE dbms_aqadm.stop_queue (
   queue_name        => 'aq.Msg_queue');
```

Java (JDBC): Stopping a Queue

```
/* Stop a queue - wait for oustanding transactions */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue                 queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "Msg_queue");

     /* Enable enqueue and dequeue */
     queue.stop(true);
}
```

Granting System Privilege

Figure 9–16 Use Case Diagram: Grant System Privilege



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To grant 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.

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, GRANT_SYSTEM_PRIVILEGE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): There is no applicable syntax reference for this use case

Usage Notes

Not applicable.

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Granting System Privilege on page 9-52
- VB (OO4O): Example not provided.
- Java (JDBC): Granting System Privilege on page 9-52

PL/SQL (DBMS_AQADM): Granting System Privilege

```
/* User AQADM grants the rights to enqueue and dequeue to ANY queues: */
```

Note: You may need to set up the following data structures for certain examples 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;
```

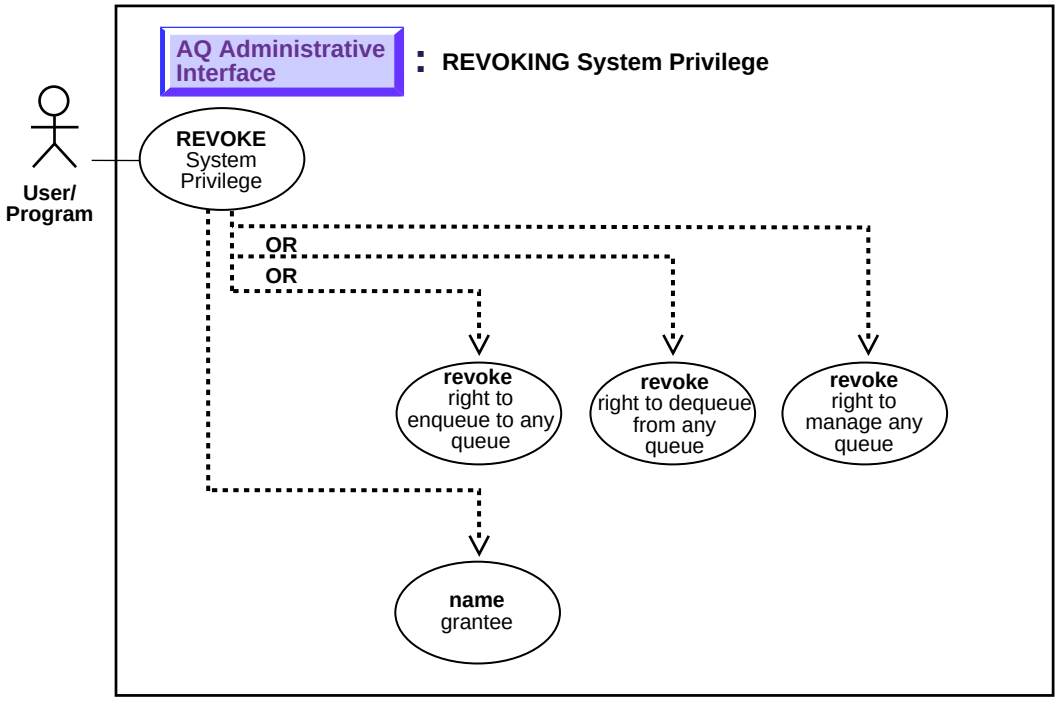
```
CONNECT aqadm/aqadm;
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege          =>    'ENQUEUE_ANY',
   grantee            =>    'Jones',
   admin_option       =>     FALSE);
EXECUTE DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege          =>     'DEQUEUE_ANY',
   grantee            =>     'Jones',
   admin_option       =>      FALSE);
```

Java (JDBC): Granting System Privilege

Feature not available through Java API

Revoking System Privilege

Figure 9–17 Use Case Diagram: Revoke System Privilege



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To revoke 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.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, REVOKE_SYSTEM_PRIVILEGE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): There is no applicable syntax reference for this use case

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

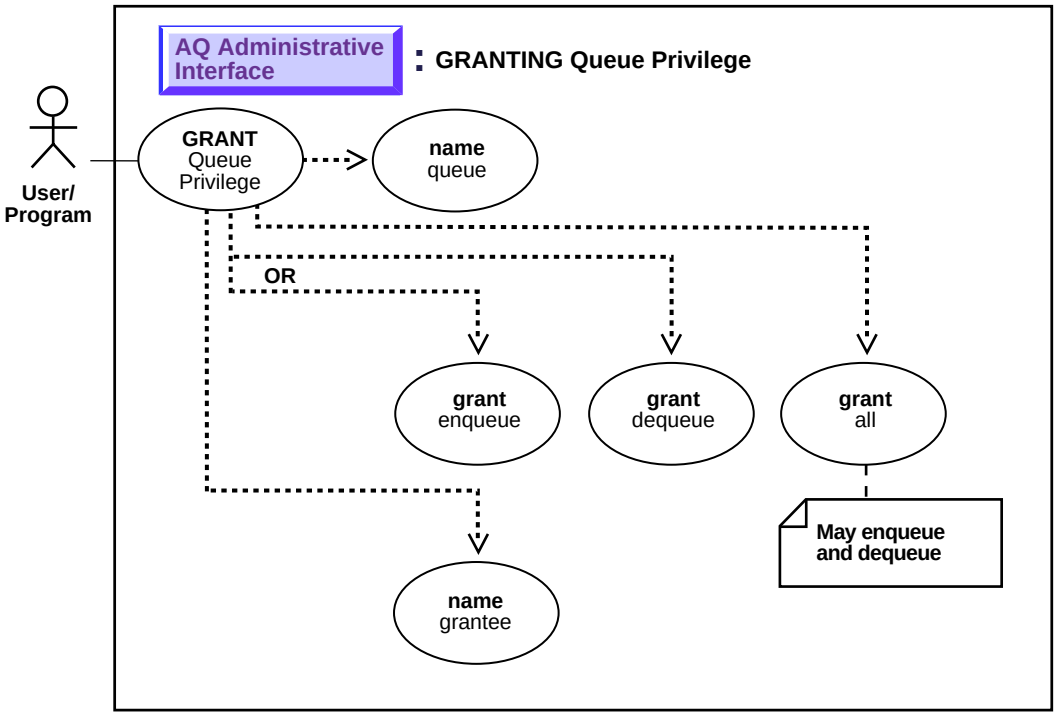
- Using PL/SQL (DBMS_AQADM): Revoking System Privilege on page 9-54
- VB (OO4O): Example not provided.
- Java (JDBC): Example not provided.

Using PL/SQL (DBMS_AQADM): Revoking System Privilege

```
/* To revoke the DEQUEUE_ANY system privilege from Jones. */
CONNECT system/manager;
   execute DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(privilege=>'DEQUEUE_ANY',
                                grantee=>'Jones');
```

Granting Queue Privilege

Figure 9–18 Use Case Diagram: Grant Queue Privilege



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To grant 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.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, GRANT_QUEUE_PRIVILEGE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, grantQueuePrivilege

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Granting Queue Privilege on page 9-56
- VB (OO4O): Example not provided.
- Java (JDBC): Granting Queue Privilege on page 9-56

PL/SQL (DBMS_AQADM): Granting Queue Privilege

```
/* User grants the access right for both enqueue and dequeue rights using
   DBMS_AQADM.GRANT. */
EXECUTE DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege     =>     'ALL',
   queue_name    =>     'aq.multiconsumermsg81_queue',
   grantee       =>     'Jones',
   grant_option  =>      TRUE);
```

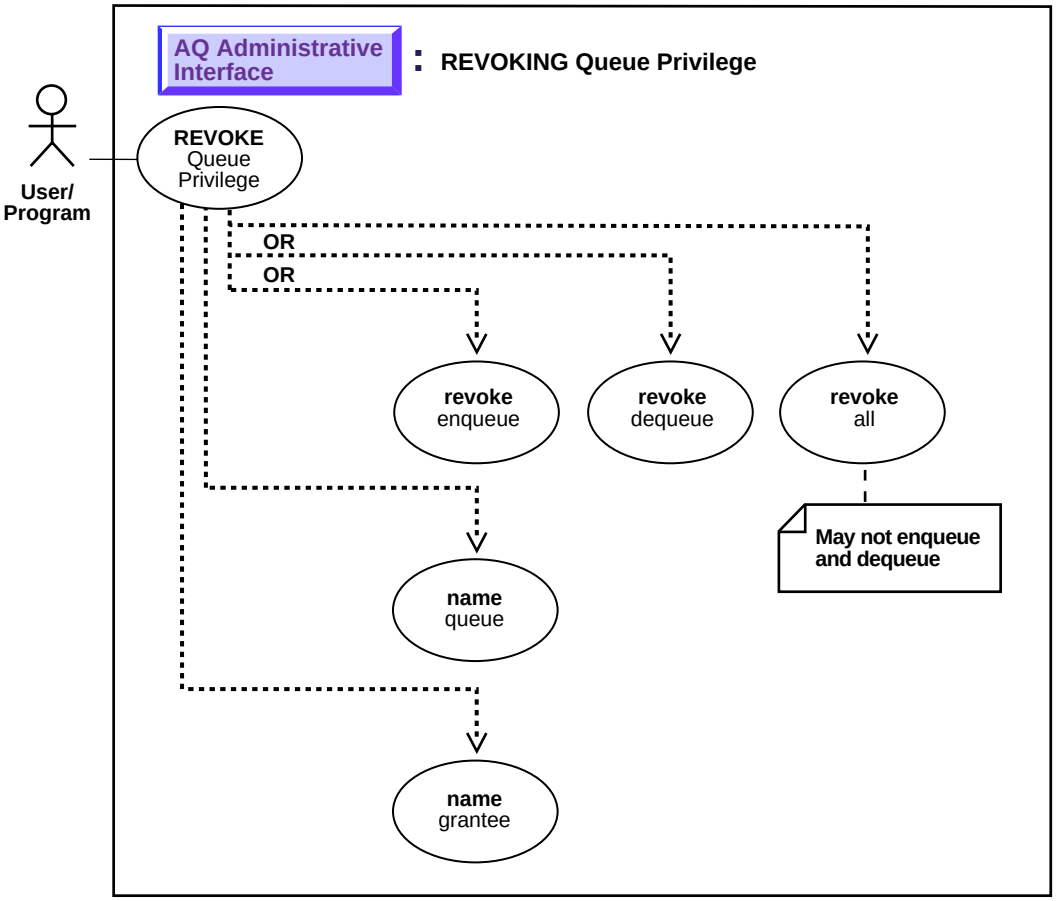
Java (JDBC): Granting Queue Privilege

```
/* Grant enqueue and dequeue privileges on queue to user 'Jones' */
public static void example(AQSession aq_sess) throws AQException
{
```

```
    AQQueue                 queue;

    /* Get the queue object */
    queue = aq_sess.getQueue("AQ", "multiconsumermsg81_queue");

    /* Enable enqueue and dequeue */
    queue.grantQueuePrivilege("ALL", "Jones", true);
}
```

Revoking Queue Privilege

Figure 9–19 Use Case Diagram: Revoke Queue Privilege



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To revoke privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE.

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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, REVOKE_QUEUE_PRIVILEGE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, revokeQueuePrivledge

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Revoking Queue Privilege on page 9-59
- VB (OO4O): Example not provided.
- Java (JDBC): Revoking Queue Privilege on page 9-60

PL/SQL (DBMS_AQADM): Revoking Queue Privilege

```
/* User can revoke the dequeue right of a grantee on a specific queue
   leaving the grantee with only the enqueue right: */
CONNECT scott/tiger;
EXECUTE DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(
   privilege     =>     'DEQUEUE',
   queue_name    =>     'scott.ScottMsgs_queue',
```

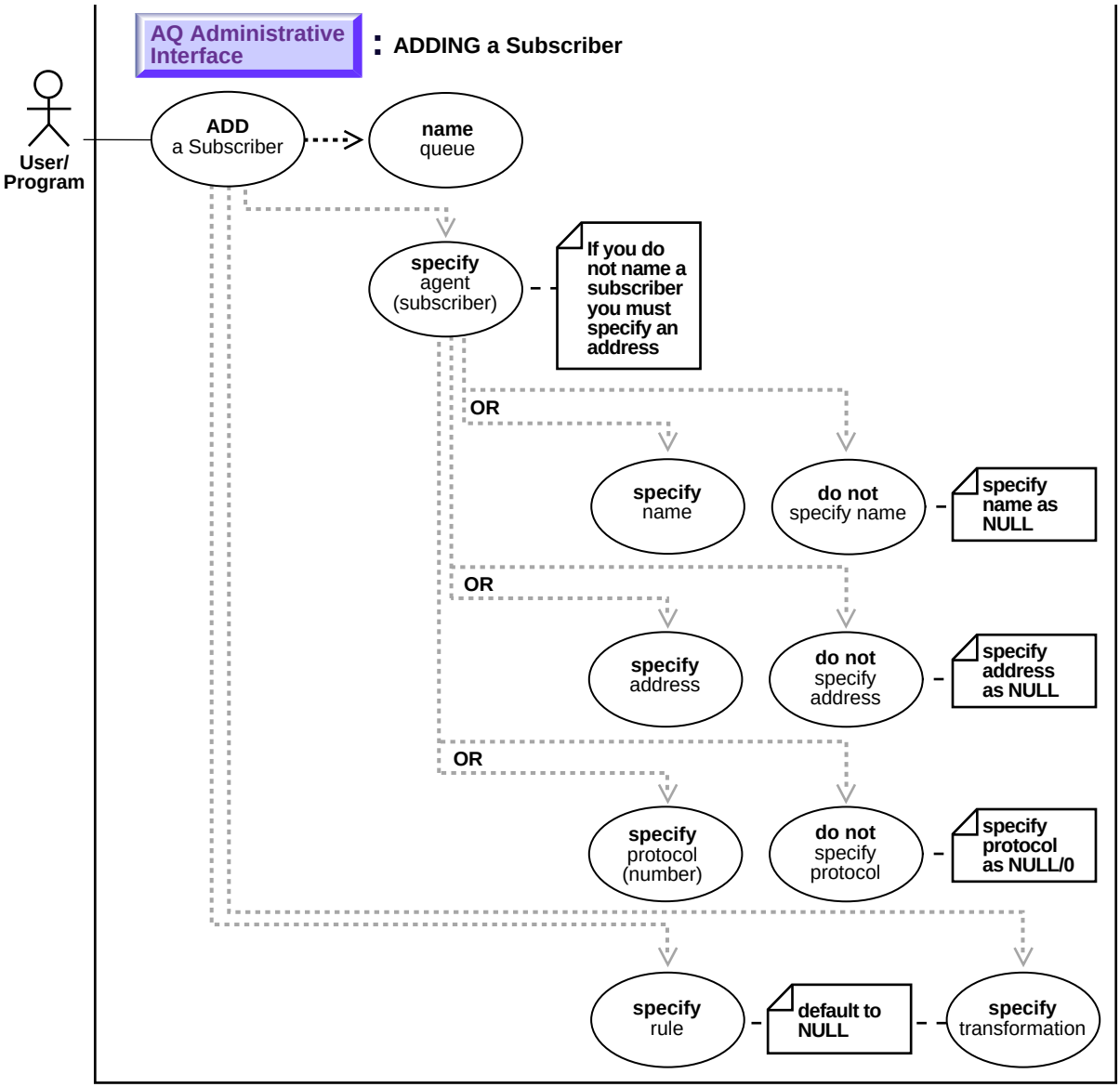
```
   grantee       =>     'Jones');
```

Java (JDBC): Revoking Queue Privilege

```
/* User can revoke the dequeue right of a grantee on a specific
   queue, leaving only the enqueue right */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue                 queue;

     /* Get the queue object */
     queue = aq_sess.getQueue("SCOTT", "ScottMsgs_queue");

     /* Enable enqueue and dequeue */
     queue.revokeQueuePrivilege("DEQUEUE", "Jones");
}
```

Adding a Subscriber

Figure 9–20 Use Case Diagram: Adding a Subscriber



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Adds a default subscriber to a queue.

Usage Note

- A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation will only succeed 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 will reflect the new behavior.
- Note that any string within the rule has to be quoted as shown below;

  ```
  rule   => 'PRIORITY <= 3 AND CORRID =  ''FROM JAPAN'''
  ```

  Note that these are all single quotation marks.
- When a queue, queue table, or subscriber is created and if GLOBAL_TOPIC_ENABLED = TRUE, 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 the*Oracle9i Supplied PL/SQL Packages and Types Reference* for more information.)
- For queues that contain payloads with XMLType attributes, you can specify rules that contain operators such as `XMLType.existsNode()` and `XMLType.extract()`.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ADD_SUBSCRIBER procedure

- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, addSubscriber

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Adding Subscriber on page 9-63
- VB (OO4O): Example not provided.
- Java (JDBC): Adding a Subscriber on page 9-63

PL/SQL (DBMS_AQADM): Adding 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;

/* Add 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;
```

Add a Subscriber and Specify a Transformation

```
/* Add a subscriber with a rule 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;
```

PL/SQL (DBMS_AQADM): Adding a Rule-Based Subscriber

```
DECLARE
   subscriber          sys.aq$_agent;
BEGIN
   subscriber :=  sys.aq$_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;

/* Add a rule-based subscriber for Overseas Shipping */
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('Overseas_DHL', null, null);

  dbms_aqadm.add_subscriber(
        queue_name     => 'OS.OS_bookedorders_que',
        subscriber     => subscriber,
        rule           => 'tab.user_data.xdata.extract(''/ORDER_
TYP/ORDERTYPE/text()'').getStringVal()=''RUSH''');

END;
```

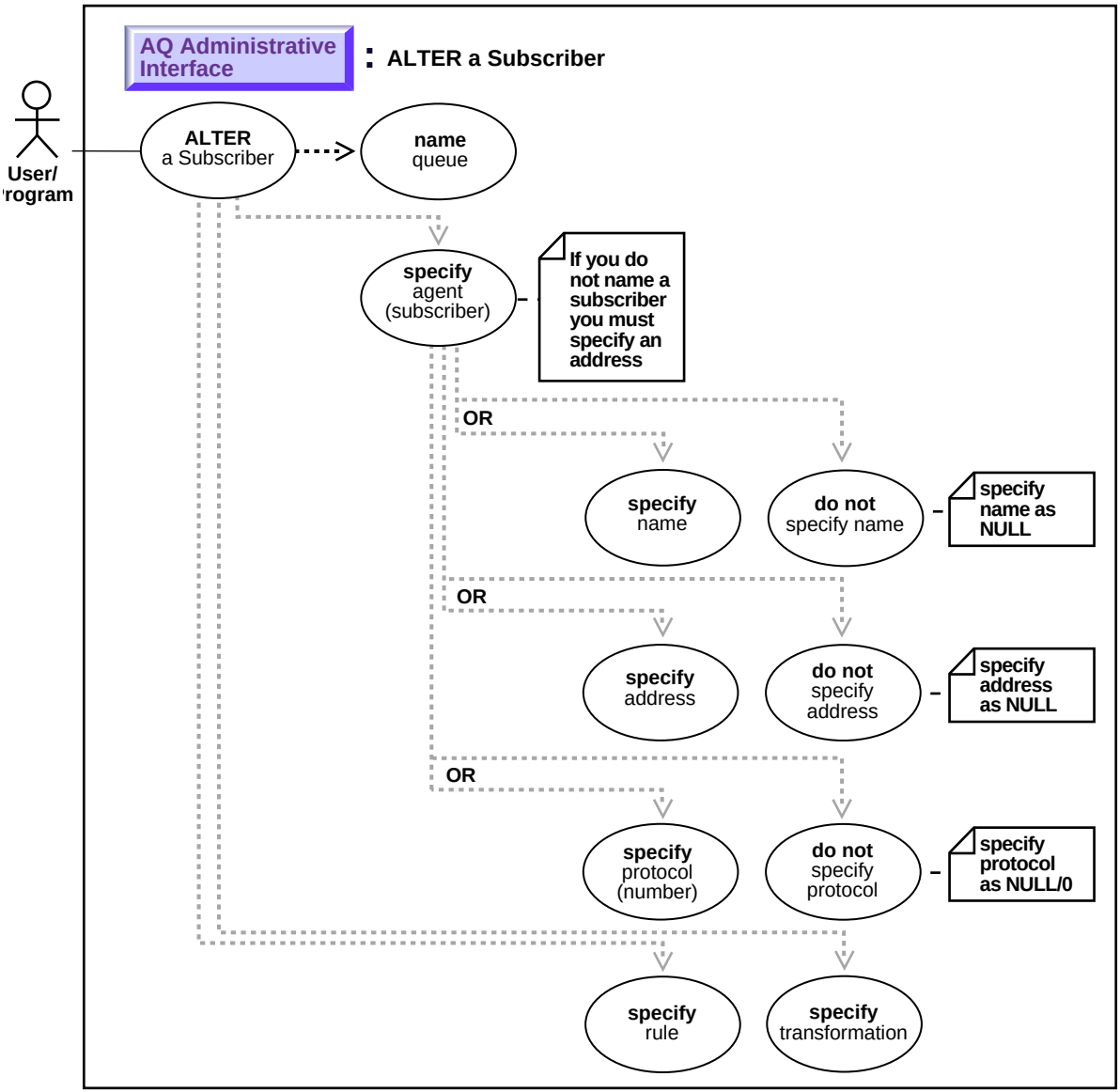
Java (JDBC): Adding a Subscriber

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

     /* Create a AQQueueTable property object */
     qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");
     qtable_prop.setMultiConsumer(true);

     q_table = aq_sess.createQueueTable ("aq", "multi_qtab", qtable_prop);

     /* Create a new AQQueueProperty object: */
     queue_prop = new AQQueueProperty();
     queue = aq_sess.createQueue (q_table, "multi_queue", queue_prop);

}

/* Add subscribers to a queue */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "multi_queue");

     /* add a subscriber */
     agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");
     queue.addSubscriber(agent1, null);

     /* add a subscriber with a rule */
     agent2 = new AQAgent("subscriber2", "aq2.msg_queue2@london");

     queue.addSubscriber(agent2, "priority < 2");
}

/* Add a subscriber with a rule */
public static void example(AQSession aq_sess) throws AQException
{
```

```
    AQQueue         queue;
    AQAgent         agent1;

    /* Get the queue object */
    queue = aq_sess.getQueue("OE", "OE_bookedorders_que");

    /* add a subscriber */
    agent1 = new AQAgent("East_Shipping", "ES.ES_bookedorders_que");

    queue.addSubscriber(agent1,
 "tab.user_data.orderregion='EASTERN' OR " +
 "(tab.user_data.ordertype='RUSH' AND " +
 "tab.user_data.customer.country='USA')");
}
```

Altering a Subscriber

Figure 9–21 Use Case Diagram: Alter a Subscriber



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Alter existing properties of a subscriber to a specified queue. Only the rule can be altered.

Usage Notes

The rule, the transformation, or both can be altered. If you only alter one of the attributes, the rule, or the transformation of the subscriber, specify the existing value of the other attribute to the alter call.

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ALTER_SUBSCRIBER procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, alterSubscriber

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Altering Subscriber on page 9-69
- VB (OO4O): Example not provided.
- Java (JDBC): Altering a Subscriber on page 9-70

PL/SQL (DBMS_AQADM): Altering Subscriber

Note: You may need to set up the following data structures for certain examples 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');
```

```
/* Add a subscriber with a 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;
```

Add a Subscriber with a Transformation

```
/* Add a subscriber with transformation */
EXECUTE DBMS_AQADM.ADD_SUBSCRIBER
   ('aq.msg_queue',
     aq$_agent('subscriber1',
        'aq2.msg_queue2@london',
            null),
```

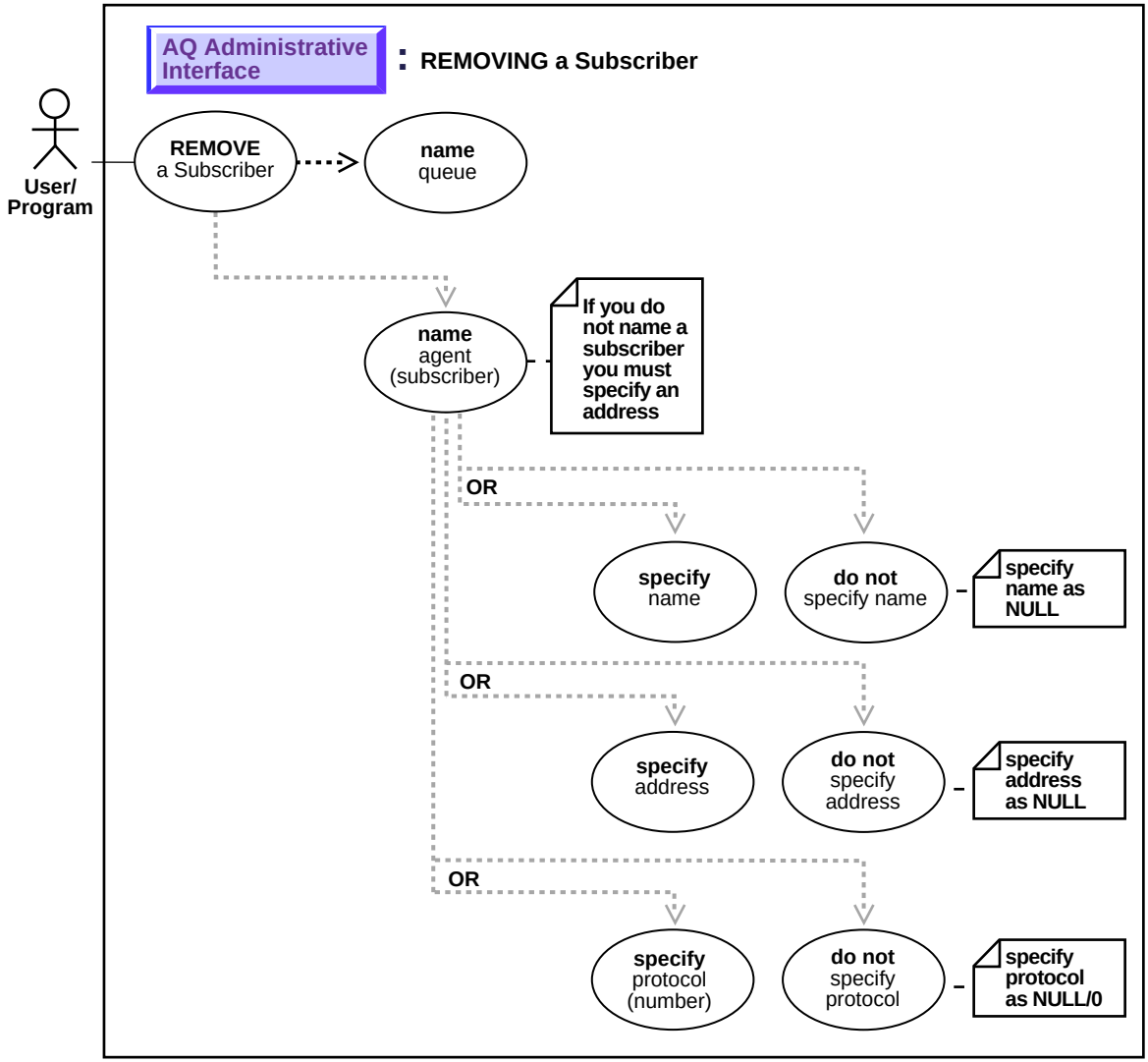
```
'AQ.MSG_MAP1');
/* Alter the subscriber*/
EXECUTE DBMS_AQADM.ALTER_SUBSCRIBER
   ('aq.msg_queue',
     aq$_agent ('subscriber1',
      'aq2.msg_queue2@london',
                null),
       'AQ.MSG.MAP2');
```

Java (JDBC): Altering a Subscriber

```
/* Alter the rule for a subscriber */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "multi_queue");

     /* add a subscriber */
     agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");

     queue.alterSubscriber(agent1, "priority=1");
}
```

Removing a Subscriber

Figure 9–22 Use Case Diagram: Remove a Subscriber



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Remove a default subscriber from a queue.

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.

When a queue, queue table, or subscriber is created, modified, or dropped, and if GLOBAL_TOPIC_ENABLED = TRUE, a corresponding LDAP entry is also created.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, REMOVE_SUBSCRIBER procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ*)* removeSubscriber

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments.

Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQADM): Removing Subscriber on page 9-73
- VB (OO4O): Example not provided.

- Java (JDBC): Removing a Subscriber on page 9-73

PL/SQL (DBMS_AQADM): Removing Subscriber

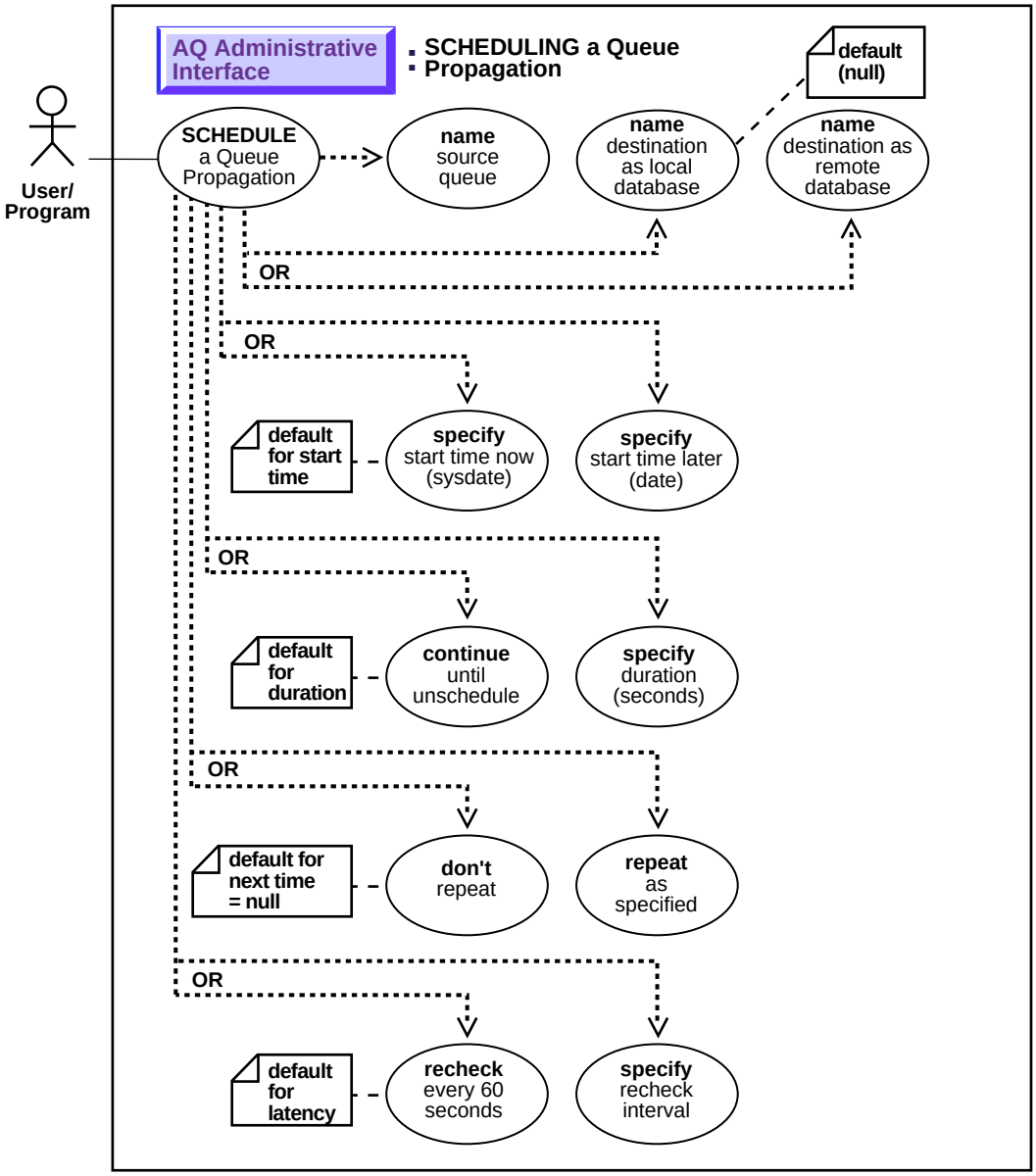
```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber1','aq2.msg_queue2', NULL);
   DBMS_AQADM.REMOVE_SUBSCRIBER(
      queue_name => 'aq.multi_queue',
      subscriber => subscriber);
END;
```

Java (JDBC): Removing a Subscriber

```
/* Remove a subscriber */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "multi_queue");

     /* add a subscriber */
     agent1 = new AQAgent("subscriber1", "aq2.msg_queue2@london");

     queue.removeSubscriber(agent1);
}
```

Scheduling a Queue Propagation

Figure 9–23 Use Case Diagram: Schedule a Queue Propagation



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Schedule propagation of messages from a queue to a destination identified by a specific dblink.

Usage Notes

Messages may 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 the message will be propagated to all of them at the same time.

See Chapter 17, "Internet Access to Advanced Queuing" for information on propagating messages over HTTP rather than Net8.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, SCHEDULE_PROPAGATION procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, schedulePropagation

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Scheduling a Queue Propagation on page 9-76
- VB (OO4O): Example not provided.

- Java (JDBC): Scheduling a Queue Propagation on page 9-76

PL/SQL (DBMS_AQADM): Scheduling a Queue Propagation

Note: You may need to set up the following data structures for certain examples 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');
```

Schedule a Propagation from a Queue to other Queues in the Same Database

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

Schedule a Propagation from a Queue to other Queues in Another Database

```
/* Schedule a propagation from queue aq.q1def to other queues in another
   database */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
   Queue_name    =>    'aq.q1def',
   Destination   =>    'another_db.world');
```

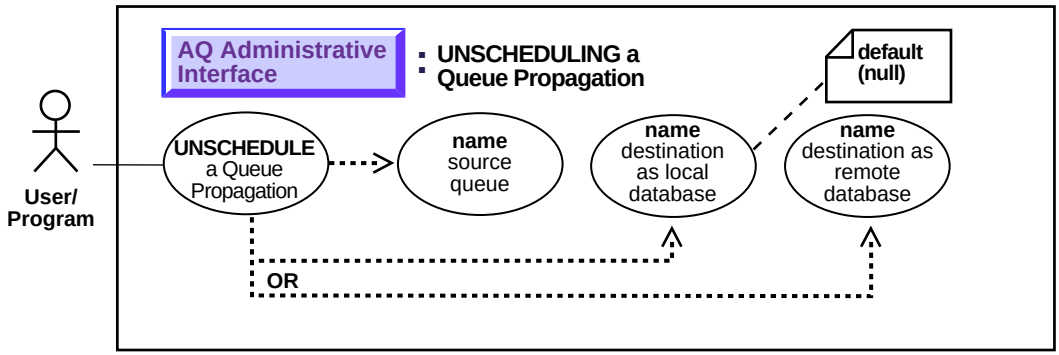
Java (JDBC): Scheduling a Queue Propagation

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

     qtable_prop = new AQQueueTableProperty("AQ.MESSAGE_TYP");
     qtable_prop.setMultiConsumer(true);
```

```
     q_table = aq_sess.createQueueTable ("aq", "objmsgs_qtab", qtable_prop);

     /* Create a new AQQueueProperty object: */
     queue_prop = new AQQueueProperty();
     queue = aq_sess.createQueue (q_table, "q1def", queue_prop);
}

/* Schedule propagation from a queue to other queues in the same database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.schedulePropagation(null, null, null, null, null);
}

/* Schedule propagation from a queue to other queues in another database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.schedulePropagation("another_db.world", null, null, null, null);
}
```

Unscheduling a Queue Propagation

Figure 9–24 Use Case Diagram: Unschedule a Queue Propagation



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Unscheduled previously scheduled propagation of messages from a queue to a destination identified by a specific `dblink`.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, UNSCHEDULE_PROPAGATION procedure

- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ*)* schedulePropagation

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Unscheduling a Propagation on page 9-79
- VB (OO4O): Example not provided.
- Java (JDBC): Unscheduling a Queue propagation on page 9-79

PL/SQL (DBMS_AQADM): Unscheduling a Propagation

Unscheduling Propagation from Queue To Other Queues in the Same Database

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

Unscheduling Propagation from a Queue to other Queues in Another Database

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

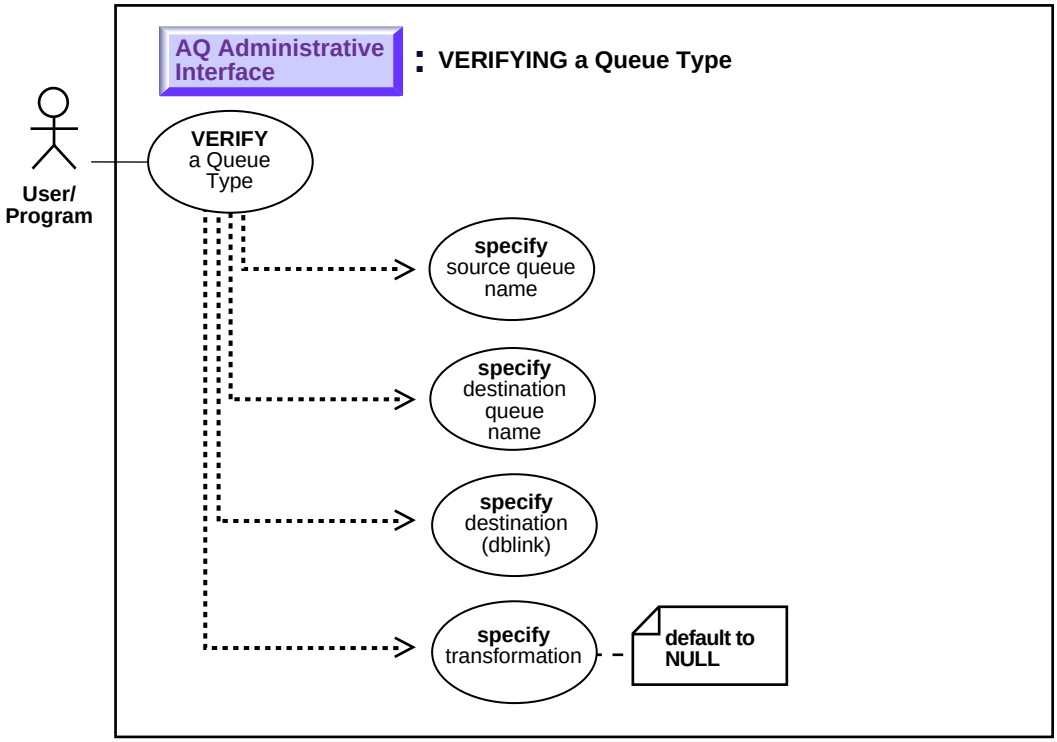
Java (JDBC): Unscheduling a Queue propagation

```
/* Unschedule propagation from a queue to other queues in the same database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");
```

```
     queue.unschedulePropagation(null);
}

/* Unschedule propagation from a queue to other queues in another database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.unschedulePropagation("another_db.world");
}
```

Verifying a Queue Type

Figure 9–25 Use Case Diagram: Verify a Queue Type



Purpose

Verify 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.

Usage Notes

If the source and destination queues do not have identical types and a transformation was specified, 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 dblink, but with different transformations.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, VERIFY_QUEUE_TYPES procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): There is no applicable syntax reference for this use case

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Verifying a Queue Type on page 9-82
- VB (OO4O): Example not provided.
- Java (JDBC): Verifying a Queue type on page 9-83

PL/SQL (DBMS_AQADM): Verifying a Queue Type

Note: You may need to set up the following data structures for certain examples to work:

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
   queue_name          => 'aq.q2def',
   queue_table         => 'aq.objmsgs_qtab');
```

```
/*  Verify if 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 */
DECLARE
```

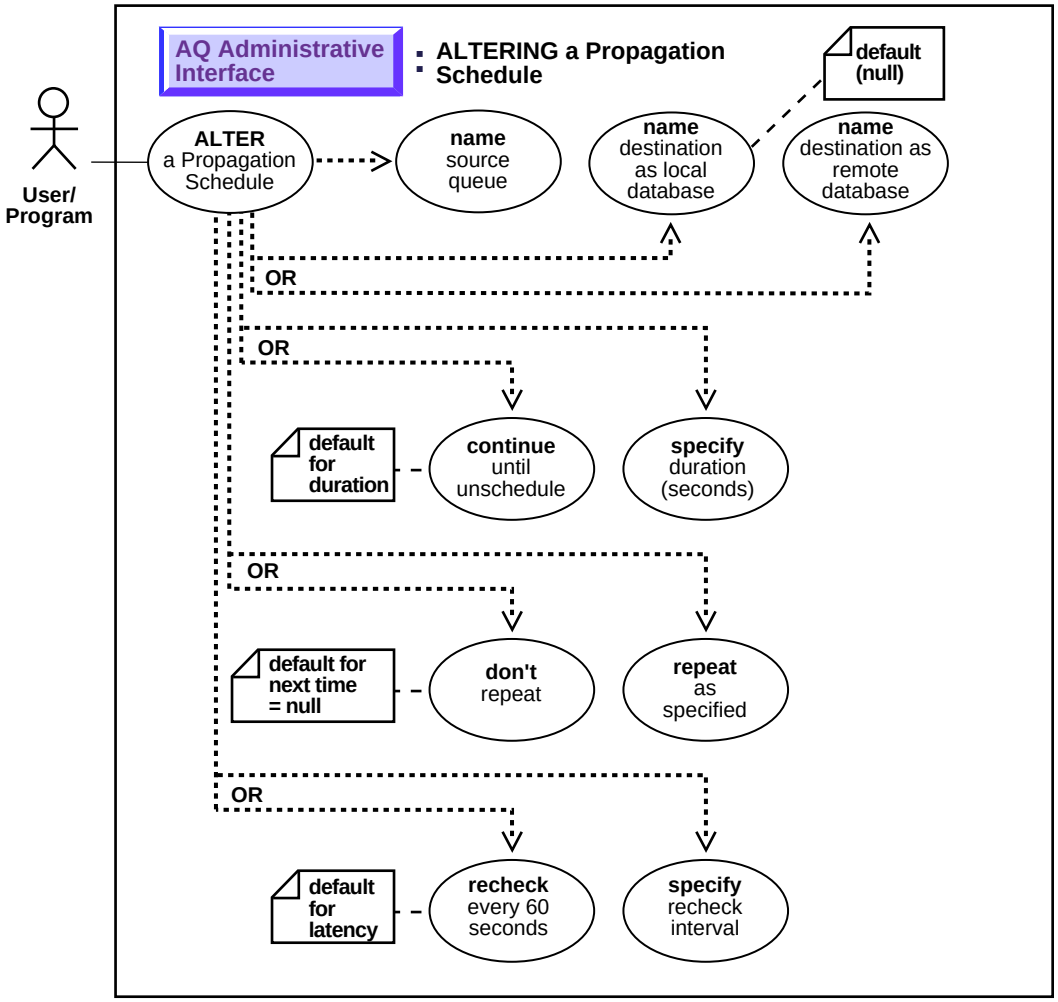
```
rc      BINARY_INTEGER;
BEGIN
/* Verify if the queues aquser.q1def and aquser.q2def in the local database
   have the same payload type */
   DBMS_AQADM.VERIFY_QUEUE_TYPES(
      src_queue_name  => 'aq.q1def',
      dest_queue_name => 'aq.q2def',
      rc              => rc);
   DBMS_OUTPUT.PUT_LINE(rc);
END;
```

Java (JDBC): Verifying a Queue type

Feature not available through Java API

Altering a Propagation Schedule

Figure 9–26 Use Case Diagram: Alter a Propagation Schedule



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To alter parameters for a propagation schedule.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ALTER_QUEUE_TABLE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ*)* alterPropagationSchedule

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Altering a Propagation Schedule on page 9-86
- VB (OO4O): Example not provided.
- PL/SQL (DBMS_AQADM): Altering a Propagation Schedule on page 9-86

PL/SQL (DBMS_AQADM): Altering a Propagation Schedule

Alter a Schedule from a Queue to Other Queues in the Same Database

```
/* Alter schedule from queue aq.q1def 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');
```

Alter a Schedule from a Queue to Other Queues in Another Database

```
/* Alter schedule from queue aq.q1def to other queues in another database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   Queue_name    =>    'aq.q1def',
   Destination   =>    'another_db.world',
   Duration      =>    '2000',
   Next_time     =>    'SYSDATE + 3600/86400',
   Latency       =>    '32');
```

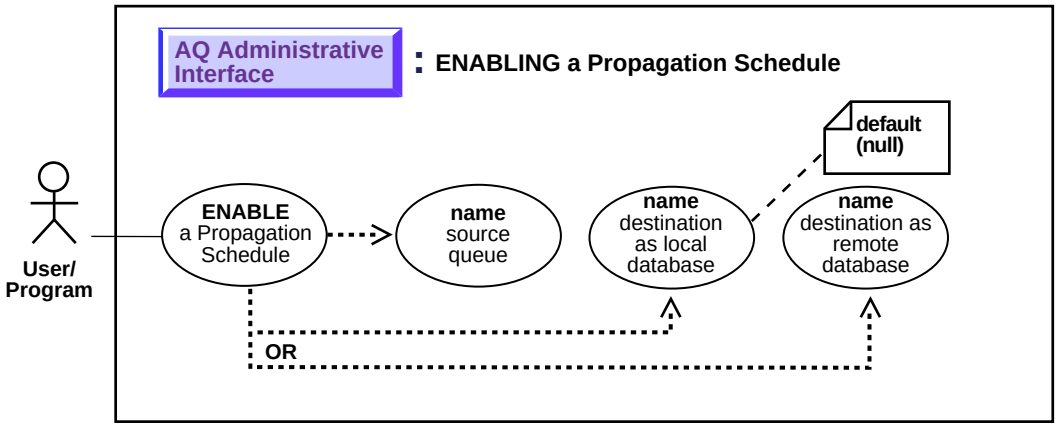
Java (JDBC): Altering a Propagation Schedule

```
/* Alter propagation schedule from a queue to other queues
   in the same database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.alterPropagationSchedule(null, new Double(2000),
    "SYSDATE + 3600/86400", new Double(32));
}

/* Unschedule propagation from a queue to other queues in another database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
```

```
    AQAgent         agent2;

    /* Get the queue object */
    queue = aq_sess.getQueue("AQ", "q1def");

    queue.alterPropagationSchedule("another_db.world", new Double(2000),
   "SYSDATE + 3600/86400", new Double(32));
}
```

Enabling a Propagation Schedule

Figure 9–27 Use Case Diagram: Enable a Propagation Schedule



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To enable a previously disabled propagation schedule.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, ENABLE_PROPAGATION_SCHEDULE procedure

- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ, enablePropagationSchedule

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Enabling a Propagation on page 9-89
- VB (OO4O): Example not provided.
- Java (JDBC): Enabling a Propagation Schedule on page 9-89

PL/SQL (DBMS_AQADM): Enabling a Propagation

Enable Propagation from a Queue to Other Queues in the Same Database

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

Enable Propagation from a Queue to Queues in Another Database

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

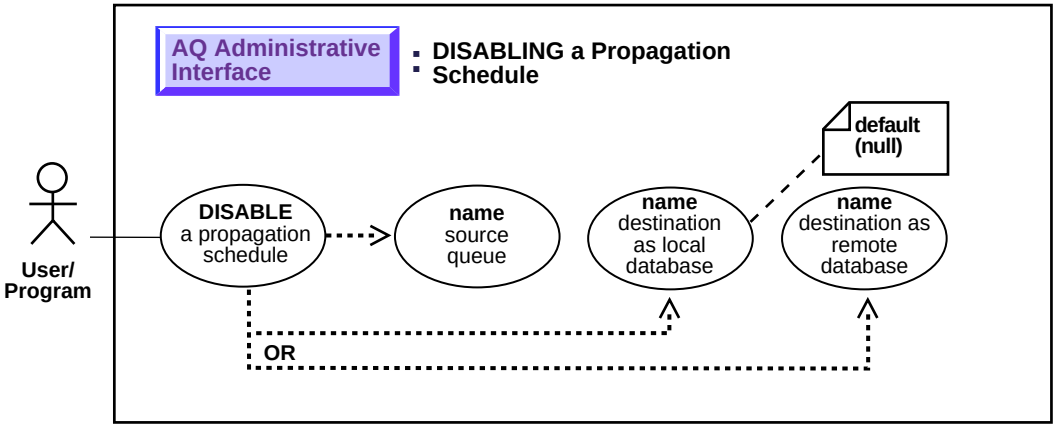
Java (JDBC): Enabling a Propagation Schedule

```
/* Enable propagation from a queue to other queues in the same database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
```

```
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.enablePropagationSchedule(null);
}

/* Enable propagation from a queue to other queues in another database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.enablePropagationSchedule("another_db.world");
}
```

Disabling a Propagation Schedule

Figure 9–28 Use Case Diagram: Disable a Propagation Schedule



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To disable a previously enabled propagation schedule.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM, DISABLE_PROPAGATION_SCHEDULE Procedure

- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ*)* disablePropagationSchedule

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): Enabling a Propagation on page 9-89
- VB (OO4O): Example not provided.
- Java (JDBC): Enabling a Propagation Schedule on page 9-89

PL/SQL (DBMS_AQADM): Disabling a Propagation

Enable Propagation from a Queue to Other Queues in the Same Database

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

Enable Propagation from a Queue to Queues in Another Database

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

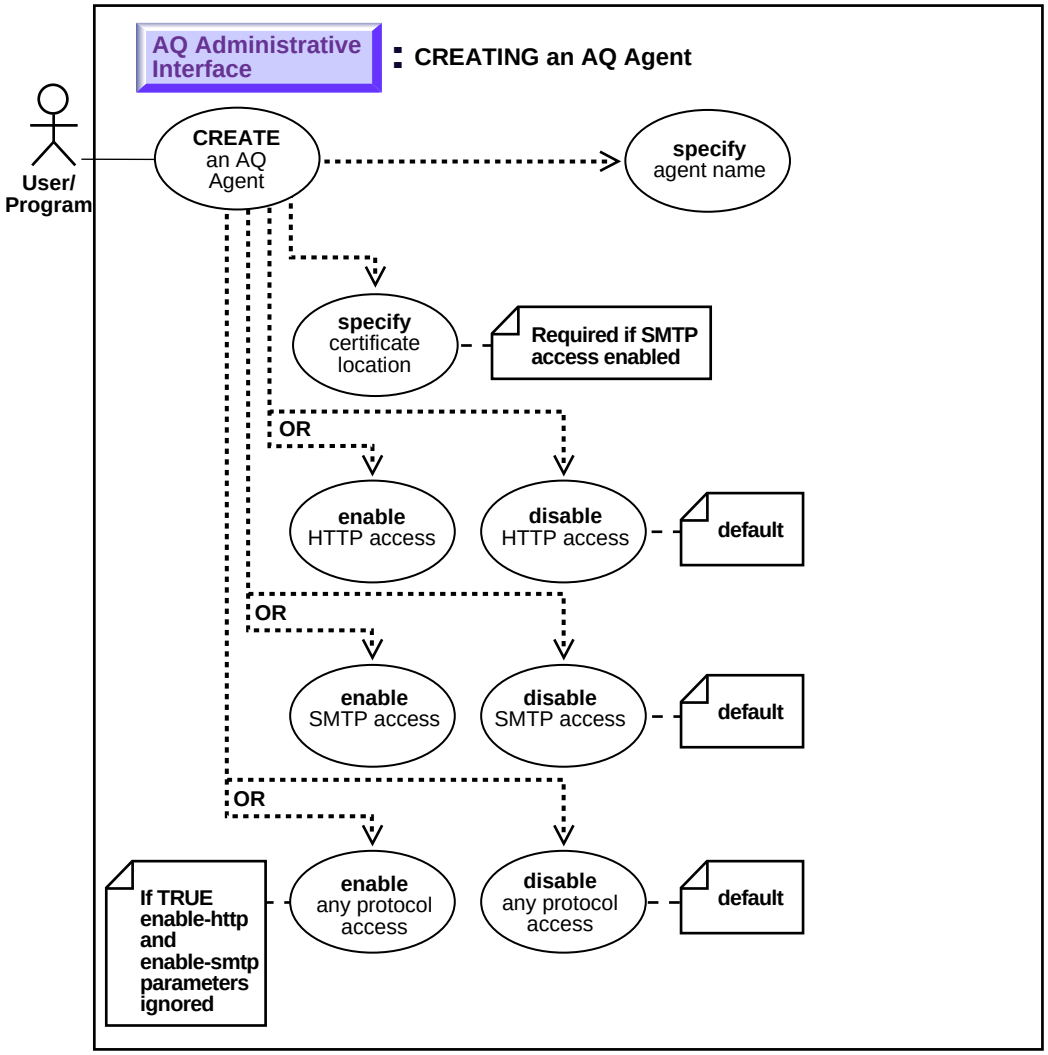
Java (JDBC): Disabling a Propagation Schedule

```
/* Disable propagation from a queue to other queues in the same database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");
```

```
     queue.disablePropagationSchedule(null);
}

/* Disable propagation from a queue to other queues in another database */
public static void example(AQSession aq_sess) throws AQException
{
     AQQueue         queue;
     AQAgent         agent1;
     AQAgent         agent2;

     /* Get the queue object */
     queue = aq_sess.getQueue("AQ", "q1def");

     queue.disablePropagationSchedule("another_db.world");
}
```

Creating an AQ Agent

Figure 9–29 Use Case Diagram: Creating an AQ Agent



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Registers an agent for AQ Internet access using HTTP/SMTP protocols.

Usage Notes

The `SYS.AQ$INTERNET_USERS` view has a list of all AQ Internet agents.

When an 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
- The user is registered for SMTP access

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.CREATE_AQ_AGENT Procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ

Examples

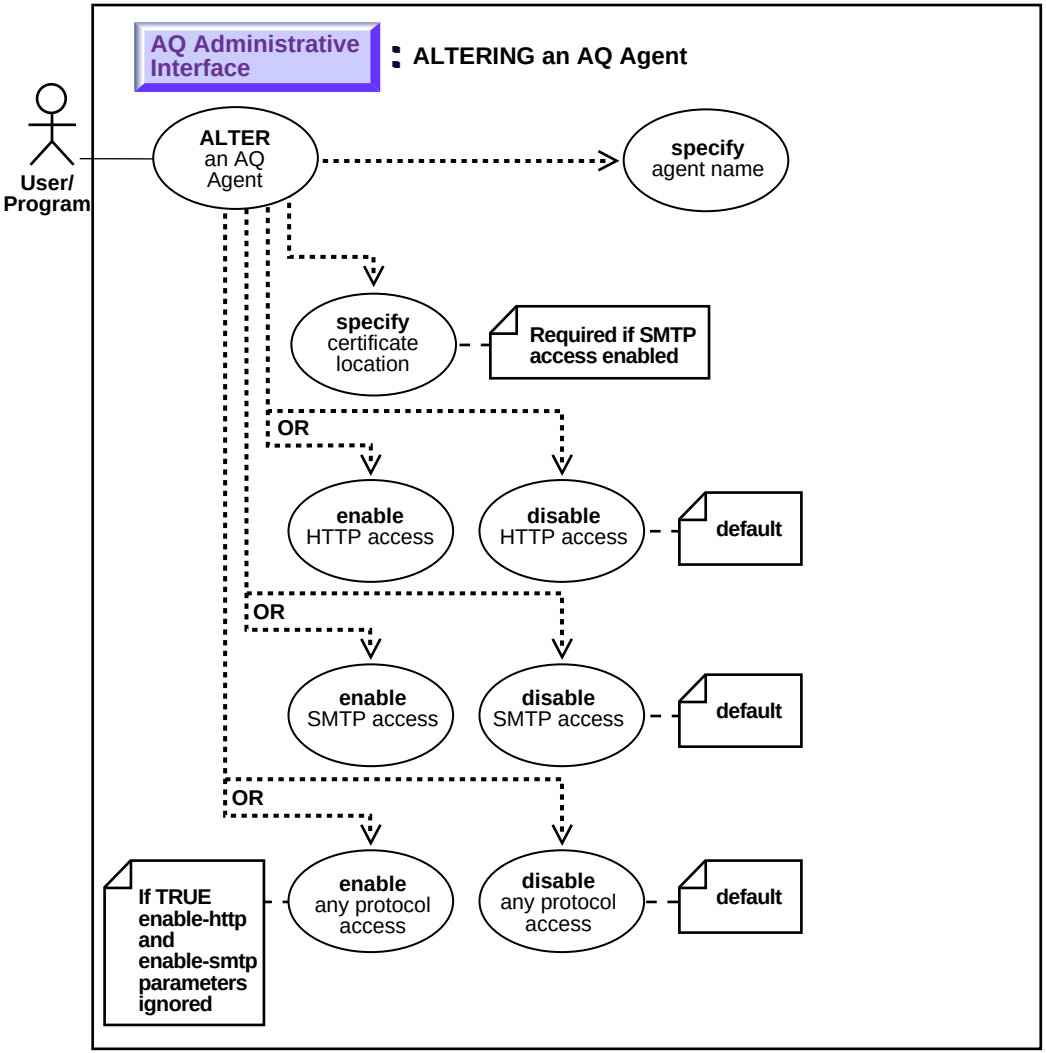
See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, CREATE_AQ_AGENT

- VB (OO4O): Example not provided.
- Java (JDBC): Example not provided.

Altering an AQ Agent

Figure 9–30 Use Case Diagram: Altering an AQ Agent



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Alters an agent registered for AQ Internet access.

Usage Notes

When an 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
- The user is registered for SMTP access

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.ALTER_AQ_AGENT Procedure

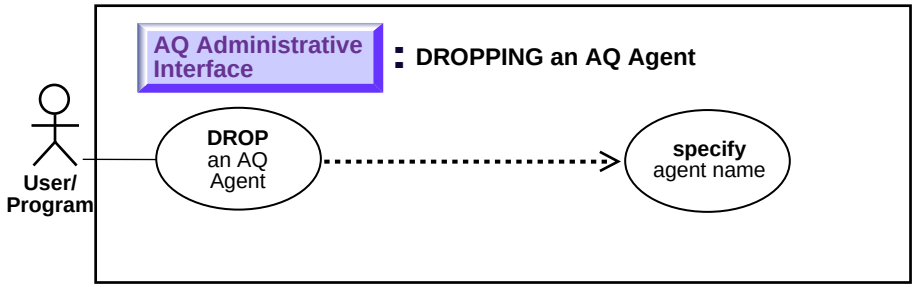
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, ALTER_AQ_AGENT

Dropping an AQ Agent

Figure 9–31 Use Case Diagram: Dropping an AQ Agent



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Drops an agent that was previously registered for AQ Internet access.

Usage Notes

When an 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
- The user is registered for SMTP access

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.DROP_AQ_AGENT Procedure

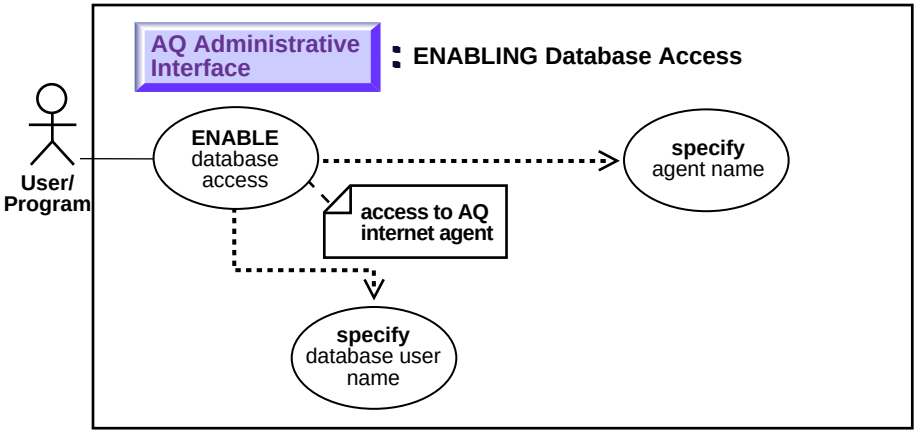
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, DROP_AQ_AGENT

Enabling Database Access

Figure 9–32 Use Case Diagram: Enabling Database Access



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Grants an AQ Internet agent the privileges of a specific database user. The AQ Internet agent should have been previously created using the `CREATE_AQ_AGENT` procedure.

Usage Notes

The SYS.AQ$INTERNET_USERS view has a list of all AQ Internet agents and the names of the database users whose privileges are granted to them.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.ENABLE_DB_ACCESS Procedure

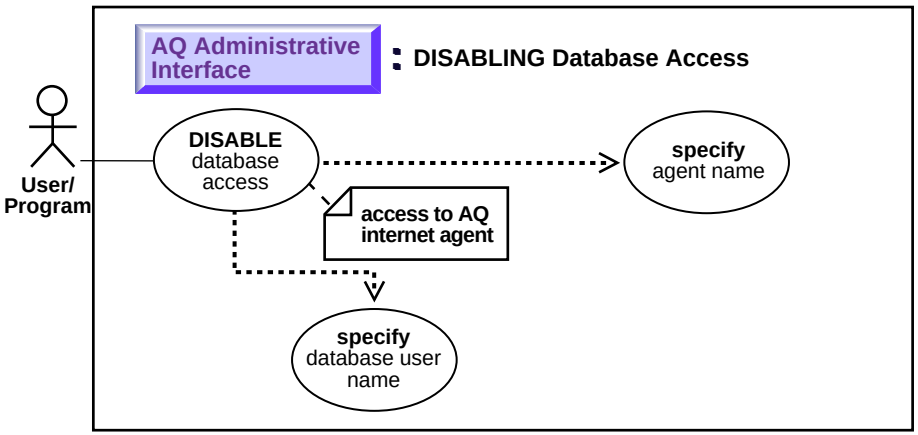
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, ENABLE_DB_ACCESS

Disabling Database Access

Figure 9–33 Use Case Diagram: Disabling Database Access



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

Revokes the privileges of a specific database user from an AQ Internet agent. The AQ Internet agent should have been previously granted those privileges using the ENABLE_DB_ACCESS procedure.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.DISABLE_DB_ACCESS Procedure

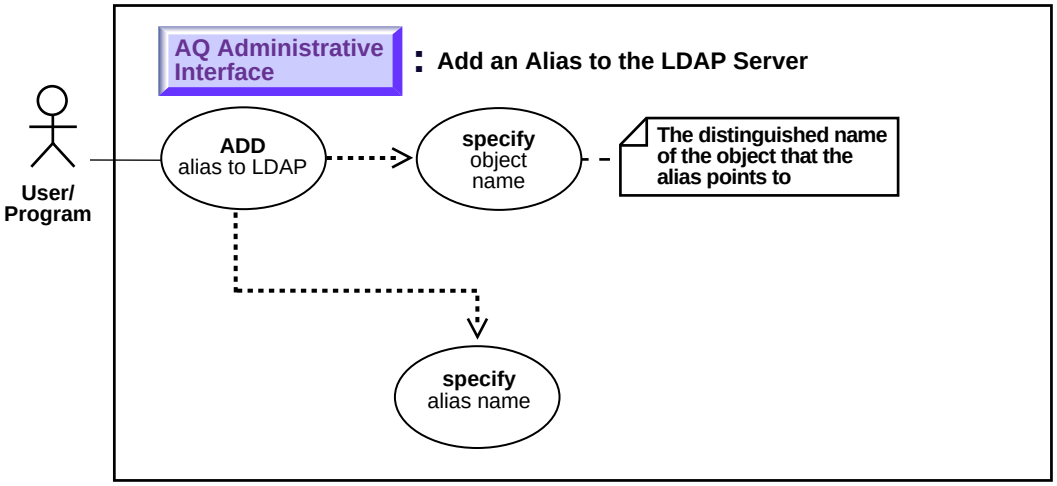
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, DISABLE_DB_ACCESS

Adding an Alias to the LDAP Server

Figure 9–34 Use Case Diagram: Add an Alias to the LDAP Server



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To add an alias to the LDAP server.

Usage Notes

This call takes the name of an alias and the distinguished name of an AQ object in LDAP, and creates the alias that points to the 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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.ADD_ALIAS_TO_LDAP Procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ

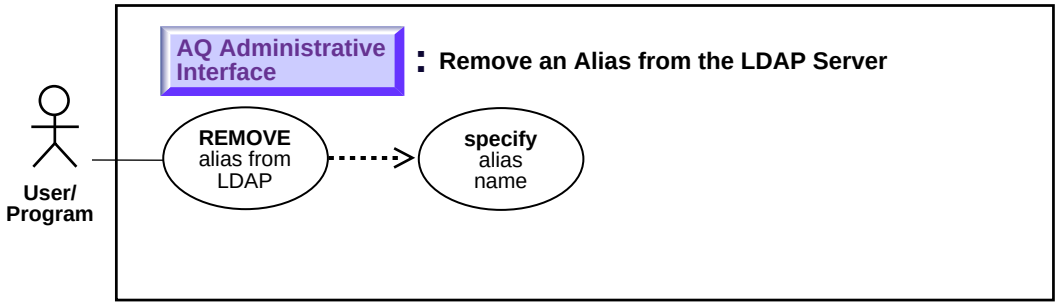
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, ADD_ALIAS_TO_LDAP
- VB (OO4O): Example not provided.
- Java (JDBC): Example not provided.

Removing an Alias from the LDAP Server

Figure 9–35 Use Case Diagram: Remove an Alias from the LDAP Server



To refer to the table of all basic operations having to do with the Administrative Interface see:

- "Use Case Model: Administrative Interface — Basic Operations" on page 9-2

Purpose

To remove an alias from the LDAP server.

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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQADM Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQADM.DEL_ALIAS_FROM_LDAP Procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case

- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.AQ

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_AQADM): See *Oracle9i Supplied PL/SQL Packages and Types Reference*, DEL_ALIAS_FROM_LDAP
- VB (OO4O): Example not provided.
- Java (JDBC): Example not provided.

10

Administrative Interface: Views

In this chapter we discuss each operation (such as "Selecting All Queue Tables in Database") in terms of a use case by that name. The use cases are listed in Table 10–1.

Figure 10–1 summarizes the use cases in a single drawing. If you are using the HTML version of this document, click on a use case title in Figure 10–1 to navigate to a particular use case.

Use cases are laid out as follows:

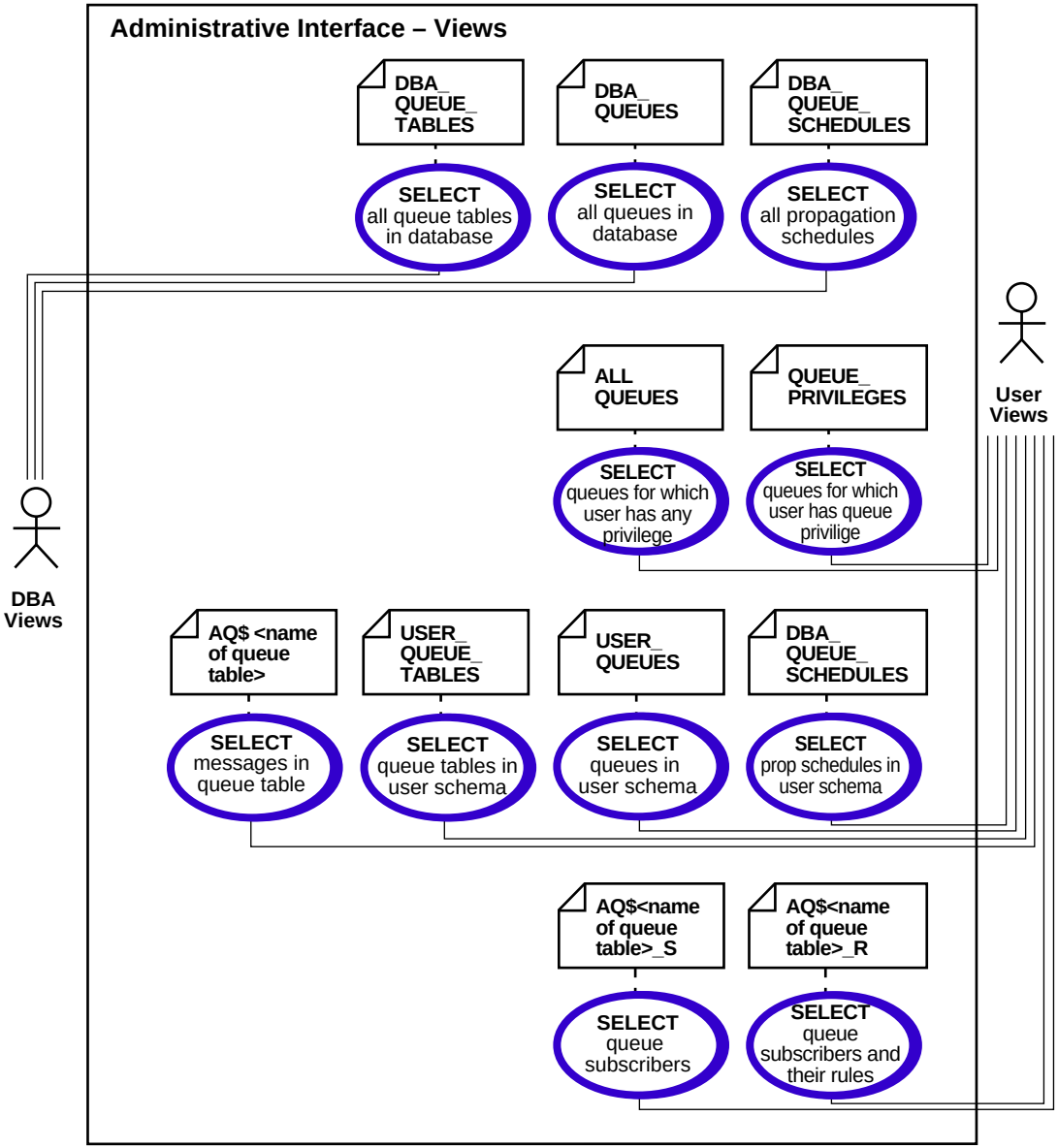
- *Use case figure*. We describe the administrative interface with respect to views in terms of a hybrid of use cases and state diagrams. That is, we describe each view as a use case in terms of the operations that represents it (such as "Selecting All Queue Tables in Database"). We describe each view as a state diagram in that each attribute of the view is represented as a possible state of the view, the implication being that any attribute (column) can be visible or invisible.
- *Syntax*. The syntax used to perform this activity.

Use Case Model: Administrative Interface — Views

Table 10–1 Use Case Model: Administrative Interface — Views

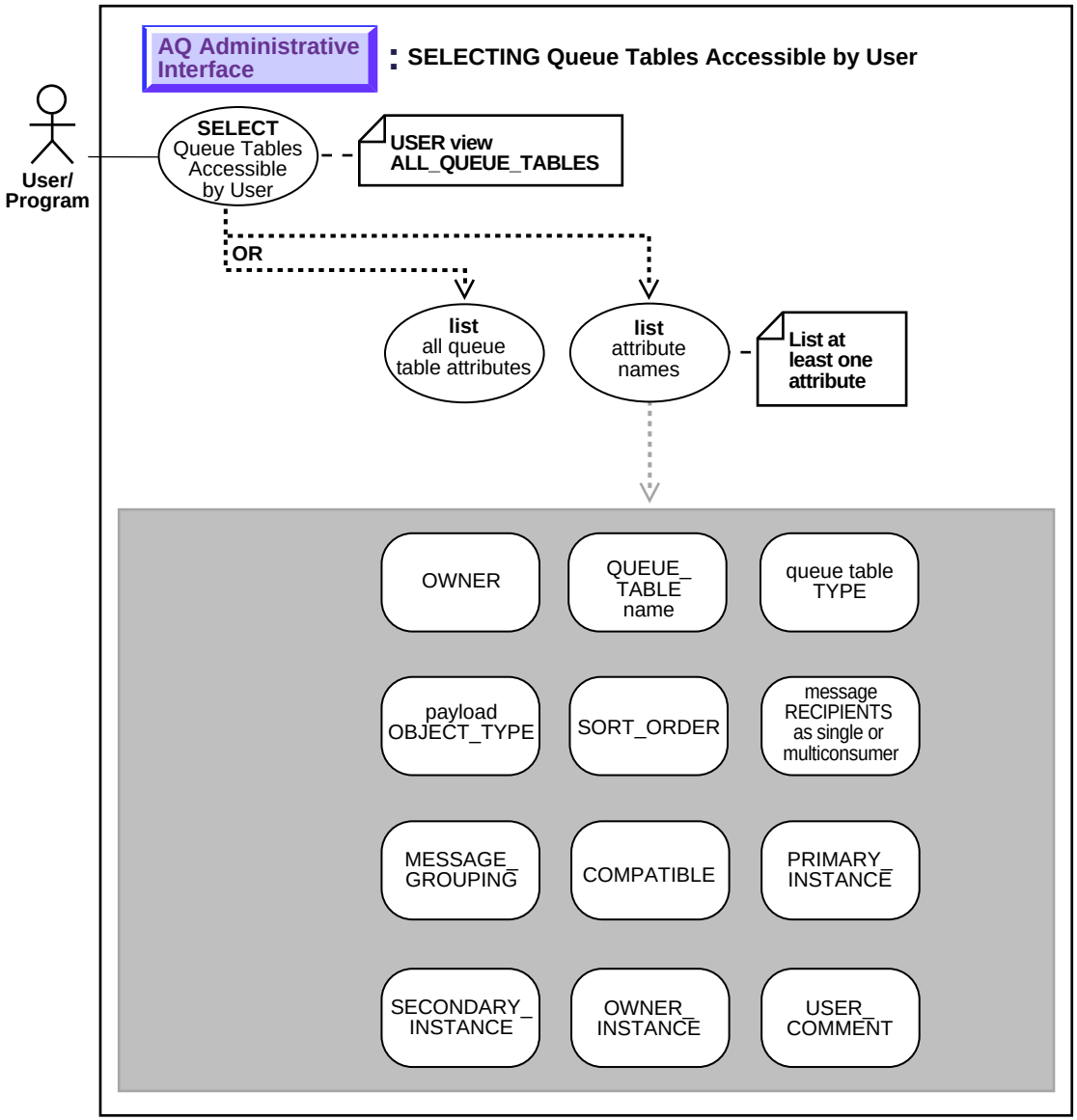
Use Case	Name of View
Selecting All Queue Tables in Database on page 10-4	DBA_QUEUE_TABLES
Selecting User Queue Tables on page 10-6	ALL_QUEUE_TABLES
Selecting All Queues in Database on page 10-8	DBA_QUEUES
Selecting All Propagation Schedules on page 10-10	DBA_QUEUE_SCHEDULES
Selecting Queues for Which User Has Any Privilege on page 10-14	ALL_QUEUES
Selecting Queues for Which User Has Queue Privilege on page 10-16	QUEUE_PRIVILEGES
Selecting Messages in Queue Table on page 10-18	AQ$<name of queue table>
Selecting Queue Tables in User Schema on page 10-22	USER_QUEUE_TABLES
Selecting Queues In User Schema on page 10-24	USER_QUEUES
Selecting Propagation Schedules in User Schema on page 10-26	USER_QUEUE_SCHEDULES
Selecting Queue Subscribers on page 10-30	AQ$<name of queue table>_S
Selecting Queue Subscribers and Their Rules on page 10-32	AQ$<name of queue table>_R
Selecting the Number of Messages in Different States for the Whole Database on page 10-34	GV$AQ
Selecting the Number of Messages in Different States for Specific Instances on page 10-36	V$AQ
Selecting the AQ Agents Registered for Internet Access on page 10-38	AQ$INTERNET_USERS
Selecting User Transformations on page 10-39	USER_TRANSFORMATIONS
Selecting User Transformation Functions on page 10-40	USER_ATTRIBUTE_TRANSFORMATIONS
Selecting All Transformations on page 10-40	DBA_TRANSFORMATIONS
Selecting All Transformation Functions on page 10-41	DBA_ATTRIBUTE_TRANSFORMATIONS

Figure 10–1 Use Case Model: Administrative Interface—Views



Selecting All Queue Tables in Database

Figure 10–2 Use Case Diagram: Select All Queue Tables in Database



Name of View

DBA_QUEUE_TABLES

Purpose

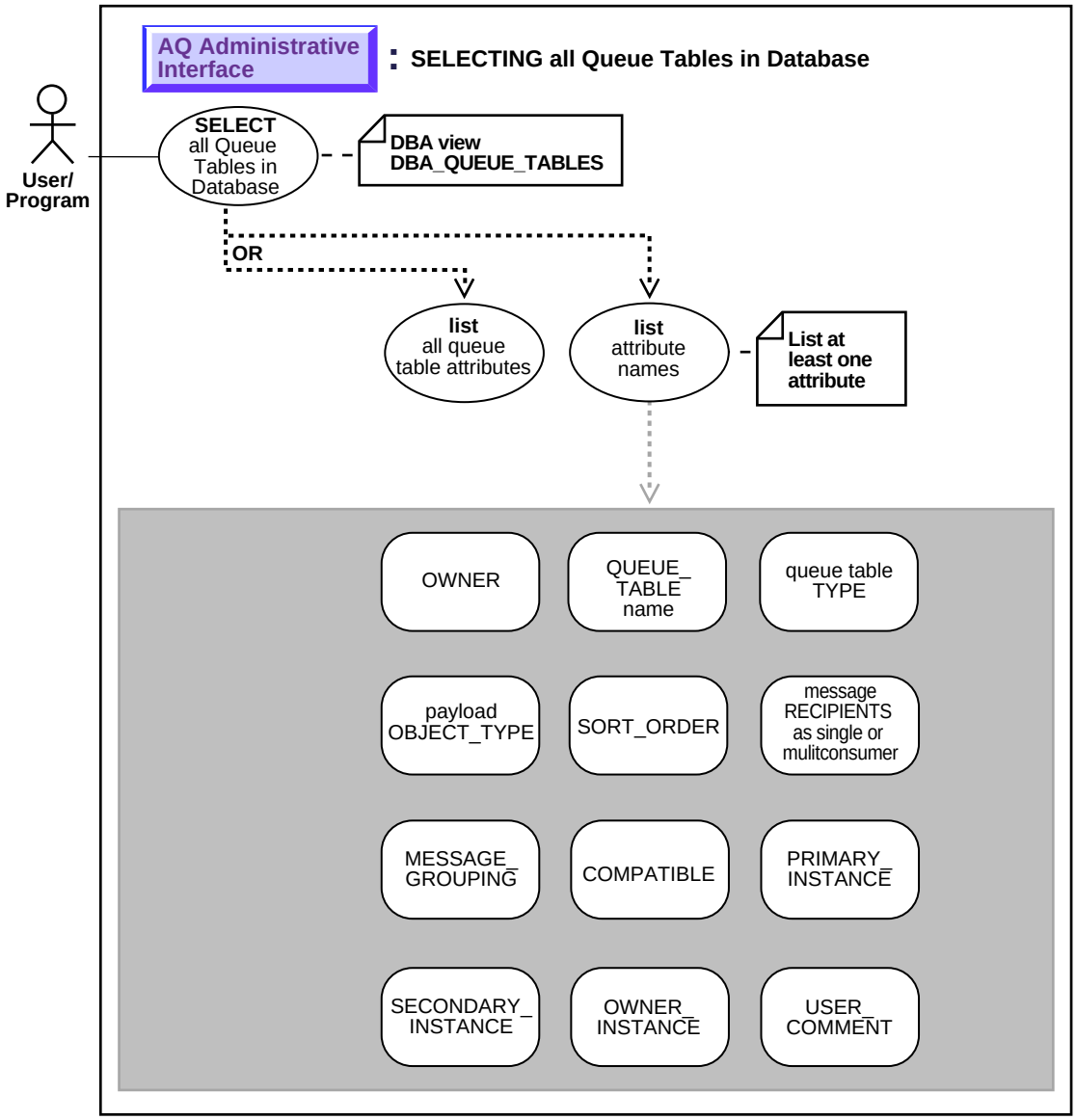
This view describes the names and types of all queue tables created in the database.

Table 10–2 DBA_QUEUE_TABLES

Column Name & Description	Null?	Type
OWNER—queue table schema		VARCHAR2(30)
QUEUE_TABLE—queue table name		VARCHAR2(30)
TYPE—payload type		VARCHAR2(7)
OBJECT_TYPE—name of object type, if any		VARCHAR2(61)
SORT_ORDER—user specified sort order		VARCHAR2(22)
RECIPIENTS—SINGLE or MULTIPLE		VARCHAR2(8)
MESSAGE_GROUPING—NONE or TRANSACTIONAL		VARCHAR2(13)
COMPATIBLE—indicates the lowest version with which the queue table is compatible		VARCHAR2(5)
PRIMARY_INSTANCE—indicates which instance is the primary owner of the queue table; a value of 0 indicates that there is no primary owner		NUMBER
SECONDARY_INSTANCE—indicates which owner 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		NUMBER
OWNER_INSTANCE—indicates which instance currently owns the queue table		NUMBER
USER_COMMENT—user comment for the queue table		VARCHAR2(50)

Selecting User Queue Tables

Figure 10–3 Use Case Diagram: Select User Queue Tables



Name of View

ALL_QUEUE_TABLES

Purpose

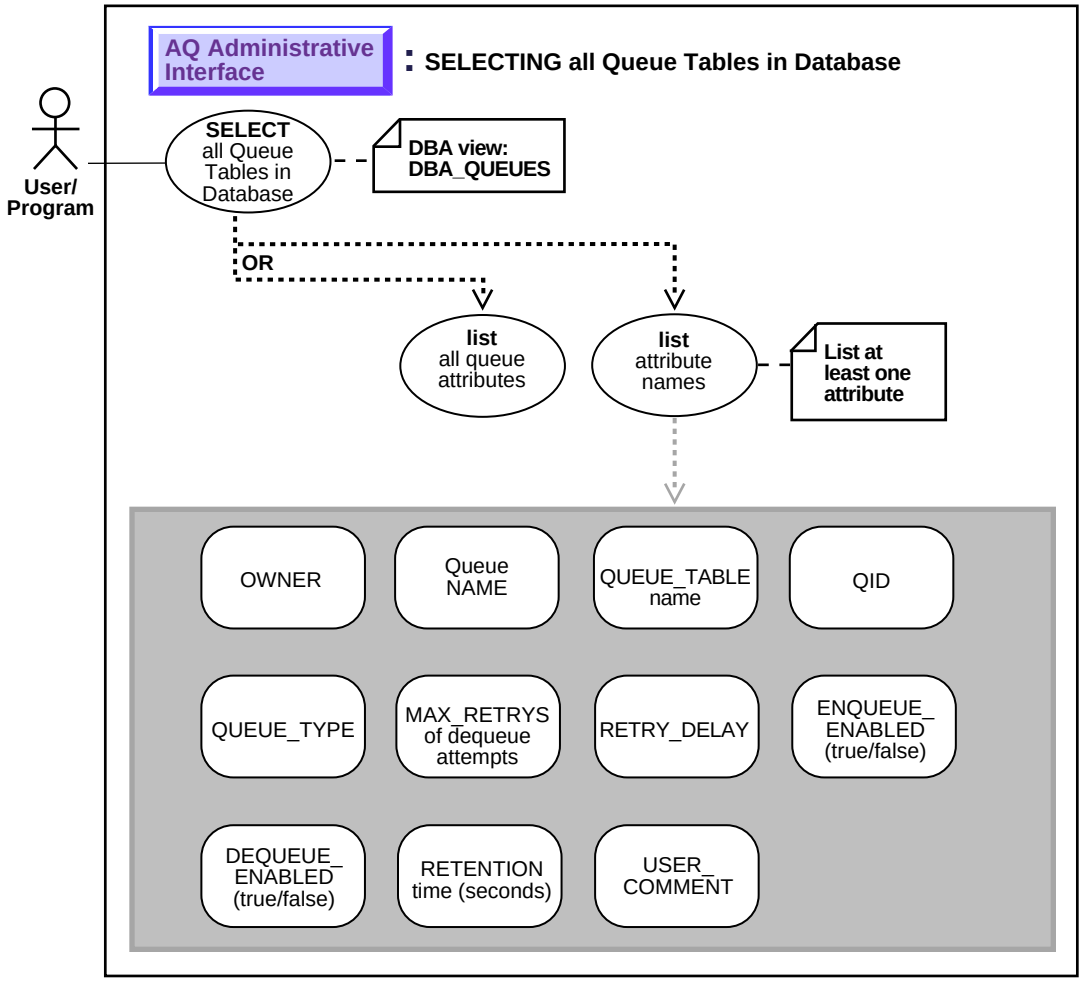
This view describes queue tables accessible to a user.

Table 10–3 DBA_QUEUE_TABLES

Column Name & Description	Null?	Type
OWNER—owner of the queue table		VARCHAR2(30)
QUEUE_TABLE—queue table name		VARCHAR2(30)
TYPE—payload type		VARCHAR2(7)
OBJECT_TYPE—object type, if any		VARCHAR2(61)
SORT_ORDER—user-specified sort order		VARCHAR2(22)
RECIPIENTS—SINGLE or MULTIPLE recipient queue		VARCHAR2(8)
MESSAGE_GROUPING—NONE or TRANSACTIONAL		VARCHAR2(13)
COMPATIBLE—indicates the lowest version with which the queue table is compatible		VARCHAR2(5)
PRIMARY_INSTANCE—indicates which instance is the primary owner of the queue table; a value of 0 indicates that there is no primary owner		NUMBER
SECONDARY_INSTANCE—indicates which owner 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		NUMBER
OWNER_INSTANCE—indicates which instance currently owns the queue table		NUMBER
USER_COMMENT—user comment for the queue table		VARCHAR2(50)

Selecting All Queues in Database

Figure 10–4 Use Case Diagram: Selecting All Queues in Database



Name of View

DBA_QUEUES

Purpose

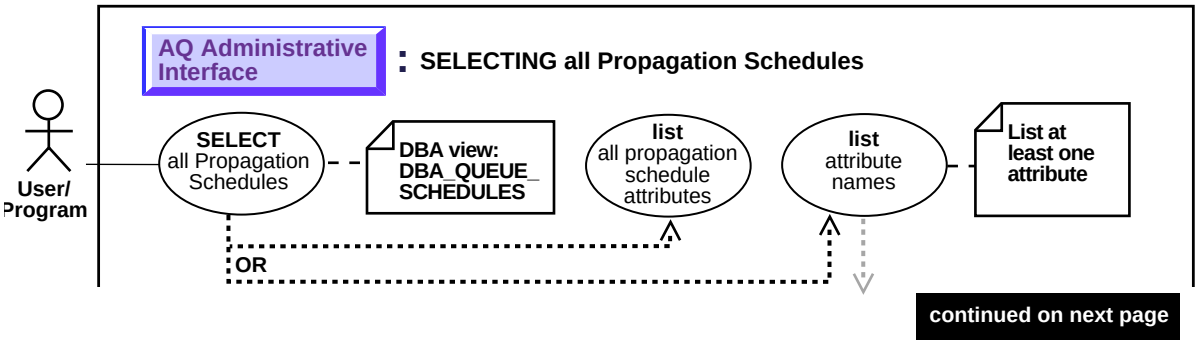
Users can specify operational characteristics for individual queues. DBA_QUEUES contains the view which contains relevant information for every queue in a database.

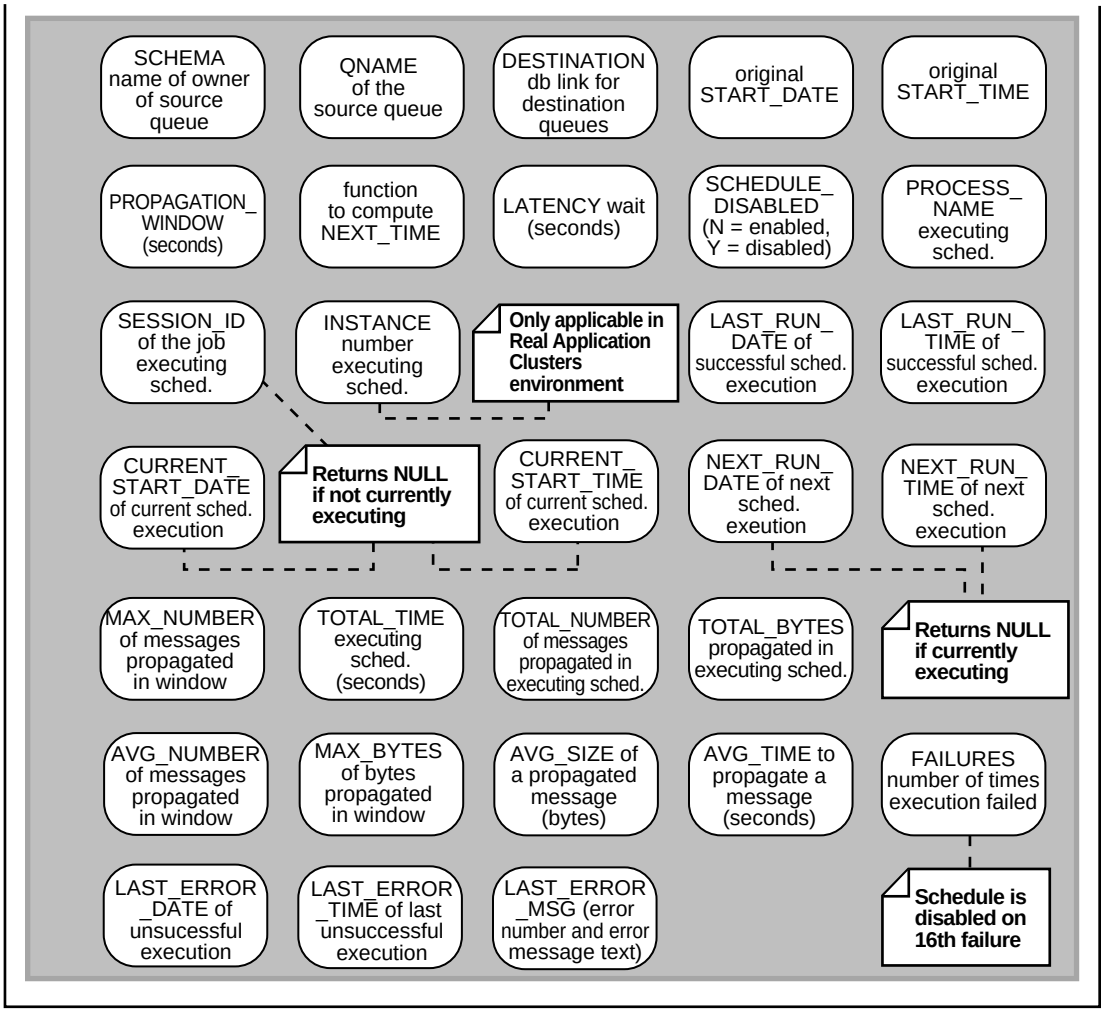
Table 10–4 DBA_QUEUES

Column Name & Description	Null?	Type
OWNER—queue schema name	NOT NULL	VARCHAR2(30)
NAME—queue name	NOT NULL	VARCHAR2(30)
QUEUE_TABLE—queue table where this queue resides	NOT NULL	VARCHAR2(30)
QID—unique queue identifier	NOT NULL	NUMBER
QUEUE_TYPE—queue type		VARCHAR2(15)
MAX_RETRIES—number of dequeue attempts allowed		NUMBER
RETRY_DELAY—number of seconds before retry can be attempted		NUMBER
ENQUEUE_ENABLED—YES / NO		VARCHAR2(7)
DEQUEUE_ENABLED—YES / NO		VARCHAR2(7)
RETENTION—number of seconds message is retained after dequeue		VARCHAR2(40)
USER_COMMENT—user comment for the queue		VARCHAR2(50)

Selecting All Propagation Schedules

Figure 10–5 Use Case Diagram: Selecting All Propagation Schedules





Name of View

DBA_QUEUE_SCHEDULES

Purpose

This view describes the current schedules for propagating messages.

Table 10–5 DBA_QUEUE_SCHEDULES

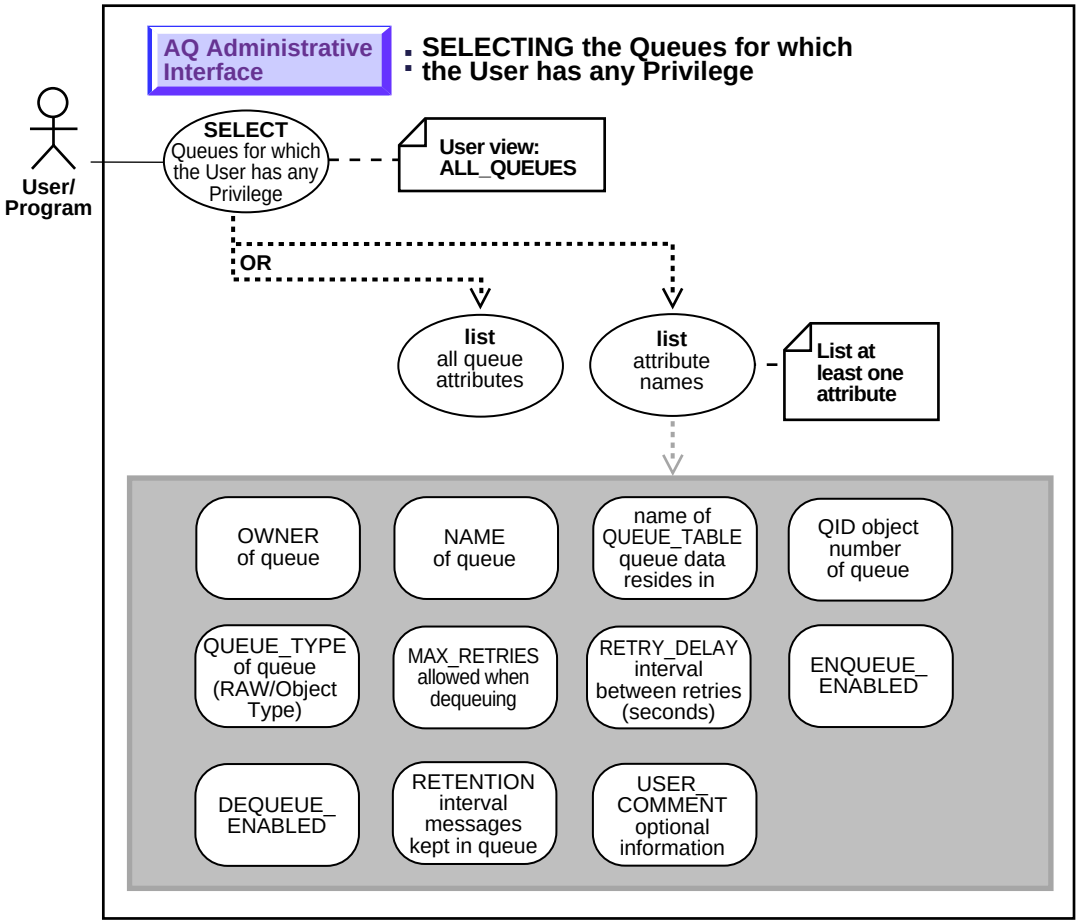
Column Name & Description	Null?	Type
SCHEMA—schema name for the source queue	NOT NULL	VARCHAR2(30)
QNAME—source queue name	NOT NULL	VARCHAR2(30)
DESTINATION—destination name, currently limited to be a DBLINK name	NOT NULL	VARCHAR2(128)
START_DATE—date to start propagation in the default date format		DATE
START_TIME—time of day at which to start propagation in HH:MI:SS format		VARCHAR2(8)
PROPAGATION_WINDOW—duration in seconds for the propagation window		NUMBER
NEXT_TIME—function to compute the start of the next propagation window		VARCHAR2(200)
LATENCY—maximum wait time to propagate a message during the propagation window.		NUMBER
SCHEDULE_DISABLED—N if enabled Y if disabled and schedule will not be executed		VARCHAR(1)
PROCESS_NAME—The name of the SNP background process executing this schedule. NULL if not currently executing		VARCHAR2(8)
SESSION_ID—The session ID (SID, SERIAL#) of the job executing this schedule. NULL if not currently executing		NUMBER
INSTANCE—The Real Application Clusters instance number executing this schedule		NUMBER
LAST_RUN_DATE—The date on the last successful execution		DATE
LAST_RUN_TIME—The time of the last successful execution in HH:MI:SS format		VARCHAR2(8)
CURRENT_START_DATE—Date at which the current window of this schedule was started		DATE
CURRENT_START_TIME—Time of day at which the current window of this schedule was started in HH:MI:SS format		VARCHAR2(8)

Table 10–5 (Cont.) DBA_QUEUE_SCHEDULES

Column Name & Description	Null?	Type
NEXT_RUN_DATE—Date at which the next window of this schedule will be started		DATE
NEXT_RUN_TIME—Time of day at which the next window of this schedule will be started in HH:MI:SS format		VARCHAR2(8)
TOTAL_TIME—Total time in seconds spent in propagating messages from the schedule		NUMBER
TOTAL_NUMBER—Total number of messages propagated in this schedule		NUMBER
TOTAL_BYTES—Total number of bytes propagated in this schedule		NUMBER
MAX_NUMBER—The maximum number of messages propagated in a propagation window		NUMBER
MAX_BYTES—The maximum number of bytes propagated in a propagation window		NUMBER
AVG_NUMBER—The average number of messages propagated in a propagation window		NUMBER
AVG_SIZE—The average size of a propagated message in bytes		NUMBER
AVG_TIME—The average time, in seconds, to propagate a message		NUMBER
FAILURES—The number of times the execution failed. If 16, the schedule will be disabled		NUMBER
LAST_ERROR_DATE—The date of the last unsuccessful execution		DATE
LAST_ERROR_TIME—The time of the last unsuccessful execution		VARCHAR2(8)
LAST_ERROR_MSG—The error number and error message text of the last unsuccessful execution		VARCHAR2(4000)

Selecting Queues for Which User Has Any Privilege

Figure 10–6 Use Case Diagram: Selecting Queues for which User has Any Privilege



Name of View

ALL_QUEUES

Purpose

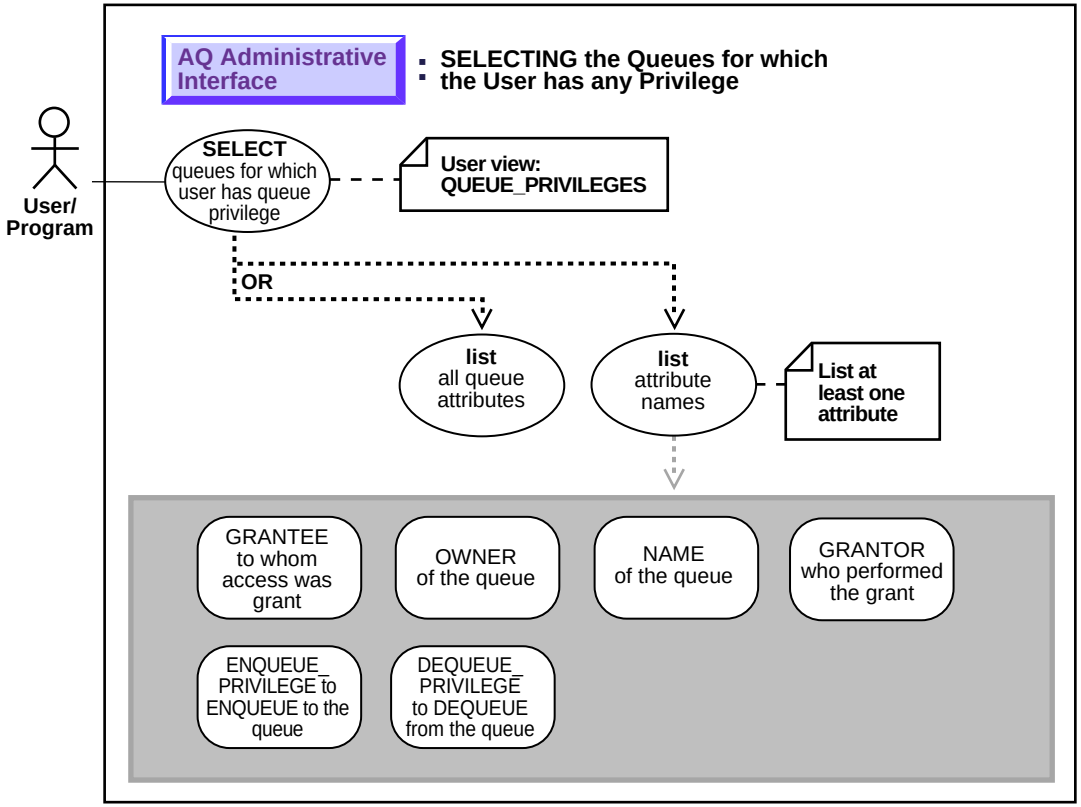
This view describes all queues accessible to the user.

Table 10–6 ALL_QUEUES

Column Name & Description	Null?	Type
OWNER—Owner of the queue	NOT NULL	VARCHAR2(30)
NAME—Name of the queue	NOT NULL	VARCHAR2(30)
QUEUE_TABLE—Name of the table the queue data resides in	NOT NULL	VARCHAR2(30)
QID—Object number of the queue	NOT NULL	NUMBER
QUEUE_TYPE—Type of the queue		VARCHAR2(15)
MAX_RETRIES—Maximum number of retries allowed when dequeuing from the queue		NUMBER
RETRY_DELAY—Time interval between retries		NUMBER
ENQUEUE_ENABLED—Queue is enabled for enqueue		VARCHAR2(7)
DEQUEUE_ENABLED—Queue is enabled for dequeue		VARCHAR2(7)
RETENTION—Time interval processed messages retained in the queue		VARCHAR2(40)
USER_COMMENT—User specified comment		VARCHAR2(50)

Selecting Queues for Which User Has Queue Privilege

Figure 10–7 Use Case Diagram: Selecting Queues for which User has Queue Privilege



Name of View

`QUEUE_PRIVILEGES`

Purpose

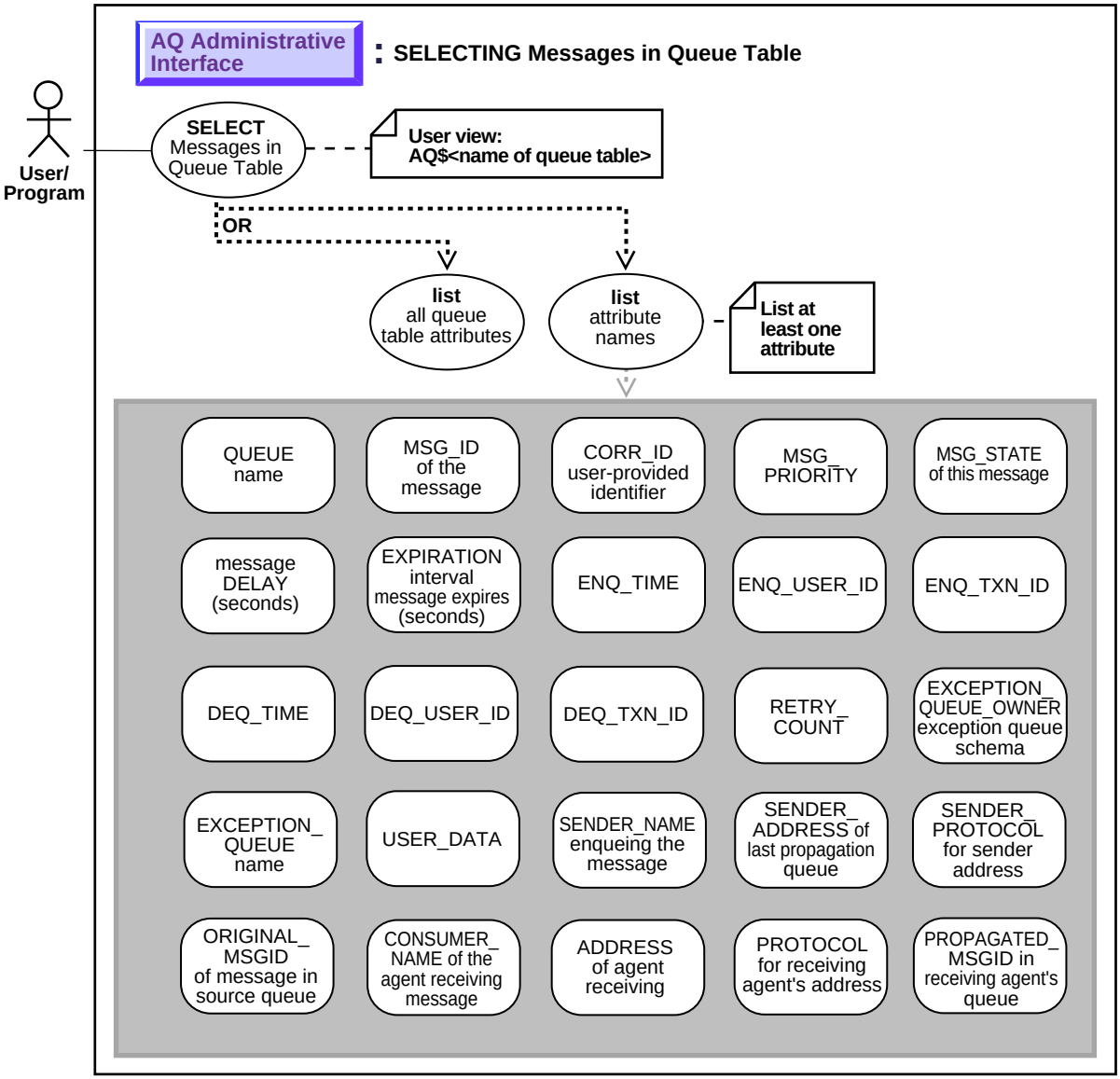
This view describes queues for which the user is the grantor, or grantee, or owner, or an enabled role or the queue is granted to `PUBLIC`.

Table 10–7 QUEUE_PRIVILEGES

Column Name & Description	Null?	Type
GRANTEE—Name of the user to whom access was granted	NOT NULL	VARCHAR2(30)
OWNER—Owner of the queue	NOT NULL	VARCHAR2(30)
NAME—Name of the queue	NOT NULL	VARCHAR2(30)
GRANTOR—Name of the user who performed the grant	NOT NULL	VARCHAR2(30)
ENQUEUE_PRIVILEGE—Permission to enqueue to the queue		NUMBER(1 if granted, 0 if not)
DEQUEUE_PRIVILEGE—Permission to dequeue to the queue		NUMBER(1 if granted, 0 if not)

Selecting Messages in Queue Table

Figure 10–8 Use Case Diagram: Selecting Messages in Queue Table



Name of View

Select messages in Queue Table.

Purpose

This view describes the queue table in which message data is stored. This view is automatically created with each queue table and should be used for querying the queue data. The dequeue history data (time, user identification and transaction identification) is only valid for single consumer queues.

Table 10–8 View for Selecting Messages in a Queue Table

Column Name & Description	Null?	Type
QUEUE—queue name		VARCHAR2(30)
MSG_ID—unique identifier of the message		RAW(16)
CORR_ID—user-provided correlation identifier		VARCHAR2(128)
MSG_PRIORIT—message priority		NUMBER
MSG_STATE—state of this message		VARCHAR2(9)
DELAY—number of seconds the message is delayed		DATE
EXPIRATION—number of seconds in which the message will expire after being READY		NUMBER
ENQ_TIME— enqueue time		DATE
ENQ_USER_ID—enqueue user id		NUMBER
ENQ_TXN_ID—enqueue transaction id	NOT NULL	VARCHAR2(30)
DEQ_TIME—dequeue time		DATE
DEQ_USER_ID—dequeue user id		NUMBER
DEQ_TXN_ID—dequeue transaction id		VARCHAR2(30)
RETRY_COUNT—number of retries		NUMBER
EXCEPTION_QUEUE_OWNER—exception queue schema		VARCHAR2(30)
EXCEPTION_QUEUE—exception queue name		VARCHAR2(30)
USER_DATA—user data		BLOB

Table 10–8 (Cont.) View for Selecting Messages in a Queue Table

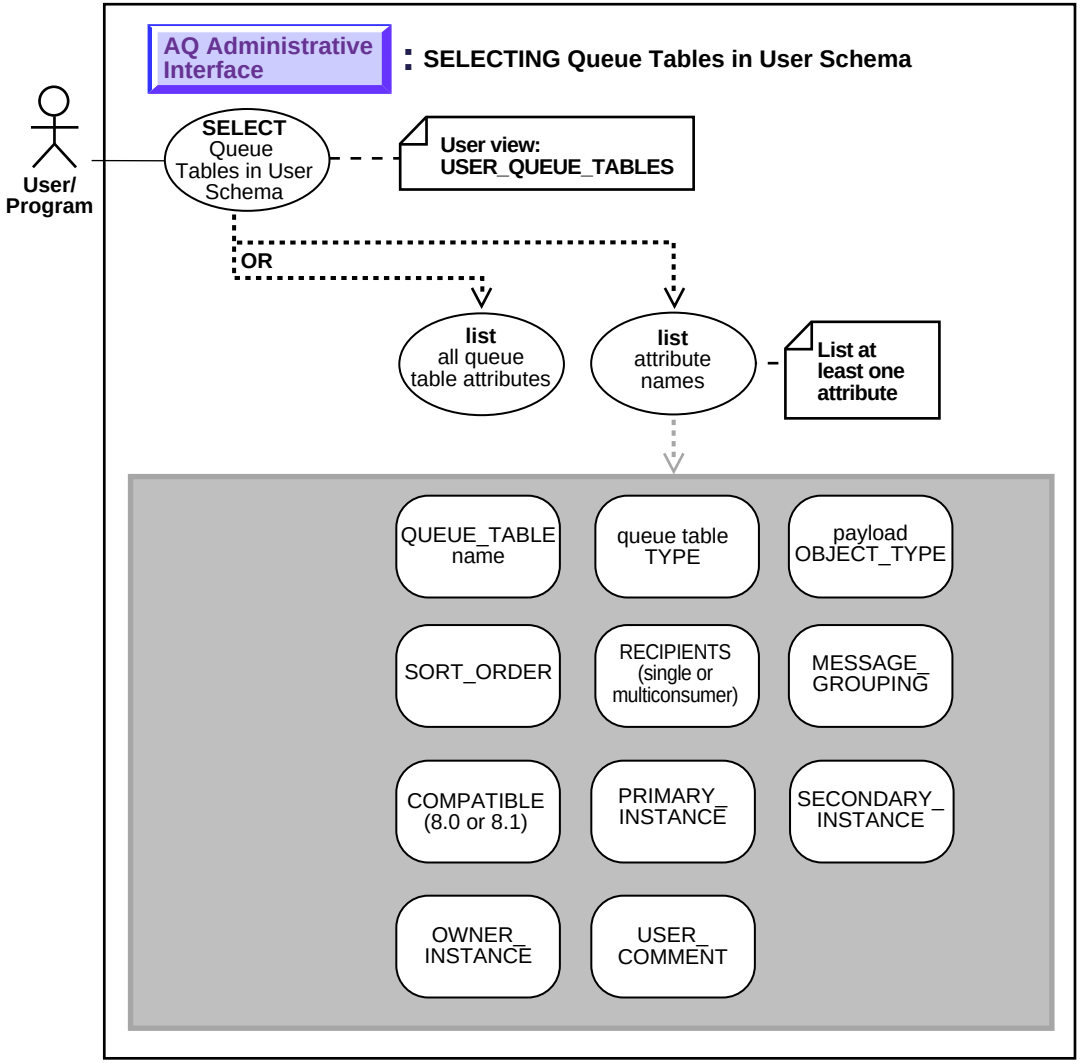
Column Name & Description	Null?	Type
SENDER_NAME—name of the Agent enqueuing the message (valid only for 8.1-compatible queue tables)		VARCHAR2(30)
SENDER_ADDRESS—queue name and database name of the source (last propagating) queue; the database name is not specified if the source queue is in the local database (valid only for 8.1-compatible queue tables)		VARCHAR2(1024)
SENDER_PROTOCOL—protocol for sender address, reserved for future use (valid only for 8.1-compatible queue tables)		NUMBER
ORIGINAL_MSGID—message id of the message in the source queue (valid only for 8.1-compatible queue tables)		RAW(16)
CONSUMER_NAME—name of the Agent receiving the message (valid ONLY for 8.1-compatible MULTICONSUMER queue tables)		VARCHAR2(30)
ADDRESS—address (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)		VARCHAR2(1024)
PROTOCOL—protocol for receiving agent's address (valid only for 8.1-compatible queue tables)		NUMBER
PROPAGATED_MSGID—message id of the message in the receiving agent's queue (valid only for 8.1-compatible queue tables)	NULL	RAW(16)
ORIGINAL_QUEUE_NAME—name of the queue the message came from		
ORIGINAL_QUEUE_OWNER—owner of the queue the message came from		

Table 10–8 (Cont.) View for Selecting Messages in a Queue Table

Column Name & Description	Null?	Type
EXPIRATION_REASON—the reason the message came into the exception queue. Possible values are TIME_EXPIRATION (message expired after the specified expired time), MAX_RETRY_EXCEEDED (max. retry count was exceeded), and PROPAGATION_FAILURE (message became undeliverable during propagation)		

Selecting Queue Tables in User Schema

Figure 10–9 Use Case Diagram: Selecting Queue Tables in User Schema



Name of View

USER_QUEUE_TABLES

Syntax

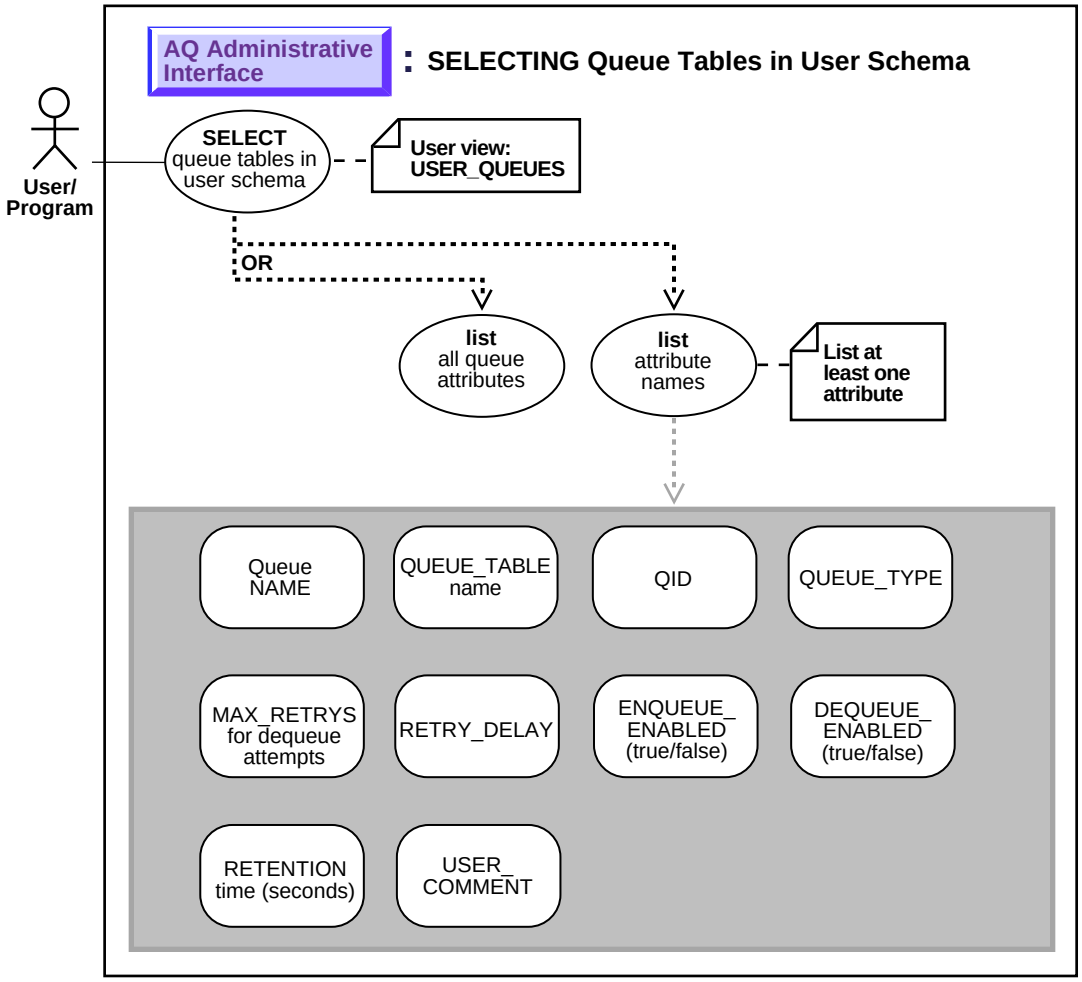
This view is the same as DBA_QUEUE_TABLES with the exception that it only shows queue tables in the user's schema. It does not contain a column for OWNER.

Table 10–9 USER_QUEUE_TABLES

Column Name & Description	Null?	Type
QUEUE_TABLE–queue table name		VARCHAR2(30)
TYPE—payload type		VARCHAR2(7)
OBJECT_TYPE—name of object type, if any		VARCHAR2(61)
SORT_ORDER—user specified sort order		VARCHAR2(22)
RECIPIENTS—SINGLE or MULTIPLE		VARCHAR2(8)
MESSAGE_GROUPING—NONE or TRANSACTIONAL		VARCHAR2(13)
COMPATIBLE—indicates the lowest version with which the queue table is compatible		VARCHAR2(5)
PRIMARY_INSTANCE—indicates which instance is the primary owner of the queue table; a value of 0 indicates that there is no primary owner		NUMBER
SECONDARY_INSTANCE—indicates which owner 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		NUMBER
OWNER_INSTANCE—indicates which instance currently owns the queue table		NUMBER
USER_COMMENT—user comment for the queue table		VARCHAR2(50)

Selecting Queues In User Schema

Figure 10–10 Use Case Diagram: Selecting Queues in User Schema



Name of View

USER_QUEUES

Purpose

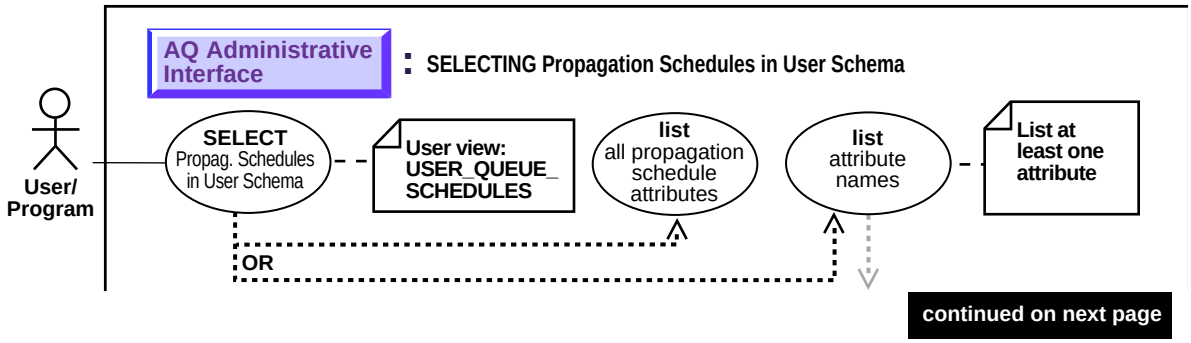
This view is the same as DBA_QUEUES with the exception that it only shows queues in the user's schema.

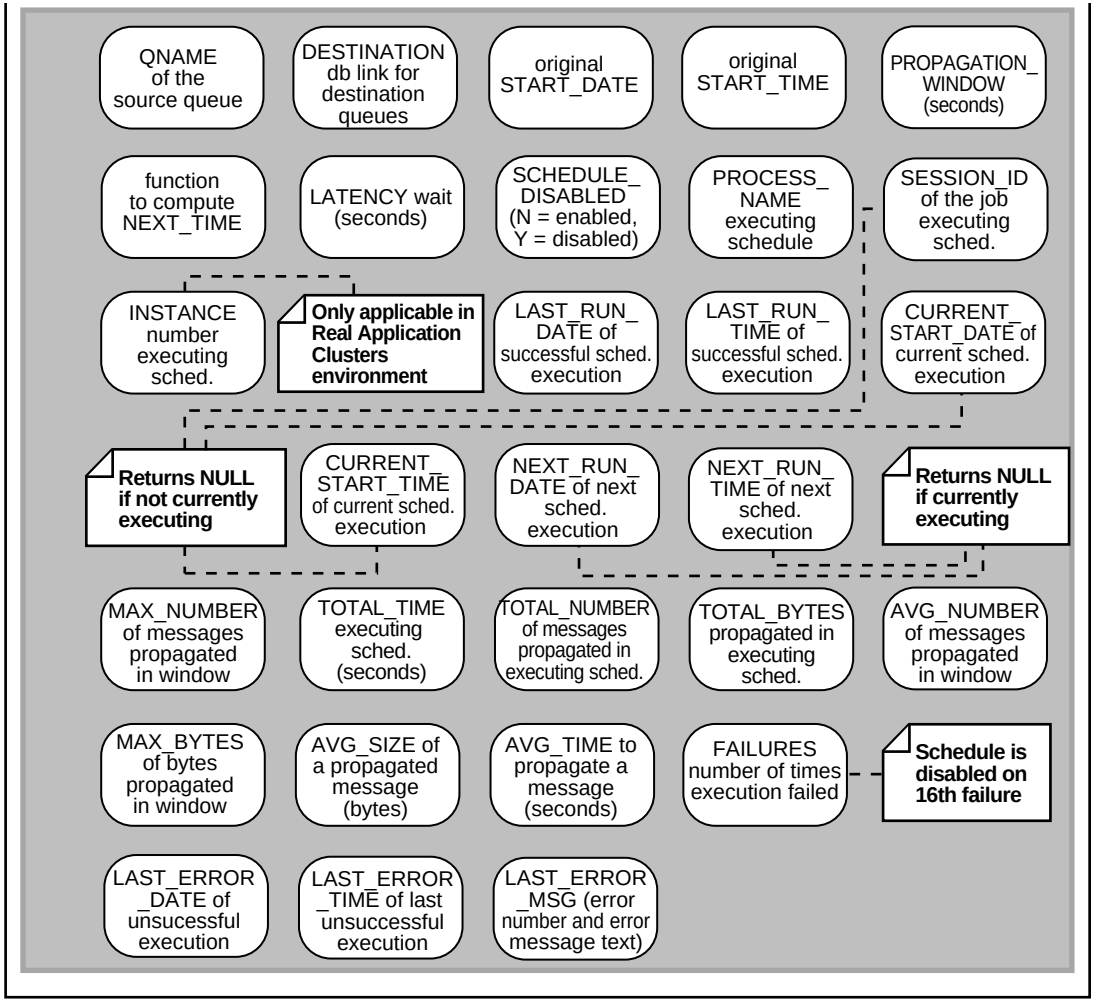
Table 10–10 USER_QUEUES

Column Name & Description	Null?	Type
NAME—queue name	NOT NULL	VARCHAR2(30)
QUEUE_TABLE—queue table where this queue resides	NOT NULL	VARCHAR2(30)
QID—unique queue identifier	NOT NULL	NUMBER
QUEUE_TYPE—queue type		VARCHAR2(15)
MAX_RETRIES—number of dequeue attempts allowed		NUMBER
RETRY_DELAY—number of seconds before retry can be attempted		NUMBER
ENQUEUE_ENABLED—YES / NO		VARCHAR2(7)
DEQUEUE_ENABLED—YES / NO		VARCHAR2(7)
RETENTION—number of seconds message is retained after dequeue		VARCHAR2(40)
USER_COMMENT—user comment for the queue		VARCHAR2(50)

Selecting Propagation Schedules in User Schema

Figure 10–11 Use Case Diagram: Select Propagation Schedules in User Schema





Name of View

USER_QUEUE_SCHEDULES

Purpose

Table 10–11 USER_QUEUE_SCHEDULES

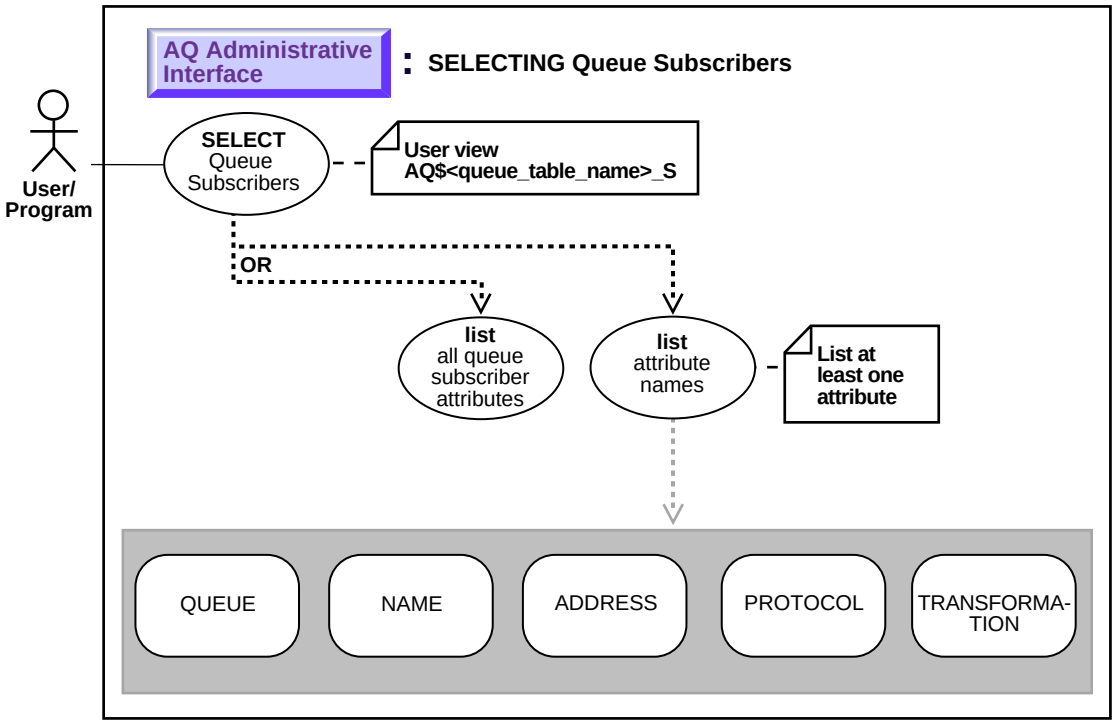
Column Name & Description	Null?	Type
QNAME—source queue name	NOT NULL	VARCHAR2(30)
DESTINATION—destination name, currently limited to be a DBLINK name	NOT NULL	VARCHAR2(128)
START_DATE—date to start propagation in the default date format		DATE
START_TIME—time of day at which to start propagation in HH:MI:SS format		VARCHAR2(8)
PROPAGATION_WINDOW—duration in seconds for the propagation window		NUMBER
NEXT_TIME—function to compute the start of the next propagation window		VARCHAR2(200)
LATENCY—maximum wait time to propagate a message during the propagation window.		NUMBER
SCHEDULE_DISABLED—N if enabled Y if disabled and schedule will not be executed		VARCHAR(1)
PROCESS_NAME—The name of the SNP background process executing this schedule. NULL if not currently executing		VARCHAR2(8)
SESSION_ID—The session ID (SID, SERIAL#) of the job executing this schedule. NULL if not currently executing		VARCHAR2(82)
INSTANCE—The Real Application Clusters instance number executing this schedule		NUMBER
LAST_RUN_DATE—The date on the last successful execution		DATE
LAST_RUN_TIME—The time of the last successful execution in HH:MI:SS format		VARCHAR2(8)
CURRENT_START_DATE—Date at which the current window of this schedule was started		DATE
CURRENT_START_TIME—Time of day at which the current window of this schedule was started in HH:MI:SS format		VARCHAR2(8)

Table 10–11 (Cont.) USER_QUEUE_SCHEDULES

Column Name & Description	Null?	Type
NEXT_RUN_DATE—Date at which the next window of this schedule will be started		DATE
NEXT_RUN_TIME—Time of day at which the next window of this schedule will be started in HH:MI:SS format		VARCHAR2(8)
TOTAL_TIME—Total time in seconds spent in propagating messages from the schedule		NUMBER
TOTAL_NUMBER—Total number of messages propagated in this schedule		NUMBER
TOTAL_BYTES—Total number of bytes propagated in this schedule		NUMBER
MAX_NUMBER—The maximum number of messages propagated in a propagation window		NUMBER
MAX_BYTES—The maximum number of bytes propagated in a propagation window		NUMBER
AVG_NUMBER—The average number of messages propagated in a propagation window		NUMBER
AVG_SIZE—The average size of a propagated message in bytes		NUMBER
AVG_TIME—The average time, in seconds, to propagate a message		NUMBER
FAILURES—The number of times the execution failed. If 16, the schedule will be disabled		NUMBER
LAST_ERROR_DATE—The date of the last unsuccessful execution		DATE
LAST_ERROR_TIME—The time of the last unsuccessful execution		VARCHAR2(8)
LAST_ERROR_MSG—The error number and error message text of the last unsuccessful execution		VARCHAR2(4000)

Selecting Queue Subscribers

Figure 10–12 Use Case Diagram: Selecting Queue Subscribers



Name of View

`AQ$<queue_table_name>_S`

Purpose

This is a view of all the subscribers for all the queues in any given queue table. This view is generated when the queue table is created and is called `aq$<queue_table_name>_s`. This view is used to query subscribers for any or all the queues in this queue table. Note that this view is only created for 8.1-compatible queue tables. This view also displays the transformation for the subscriber if it was created with one.

Table 10–12 AQ$<queue_table_name>_S

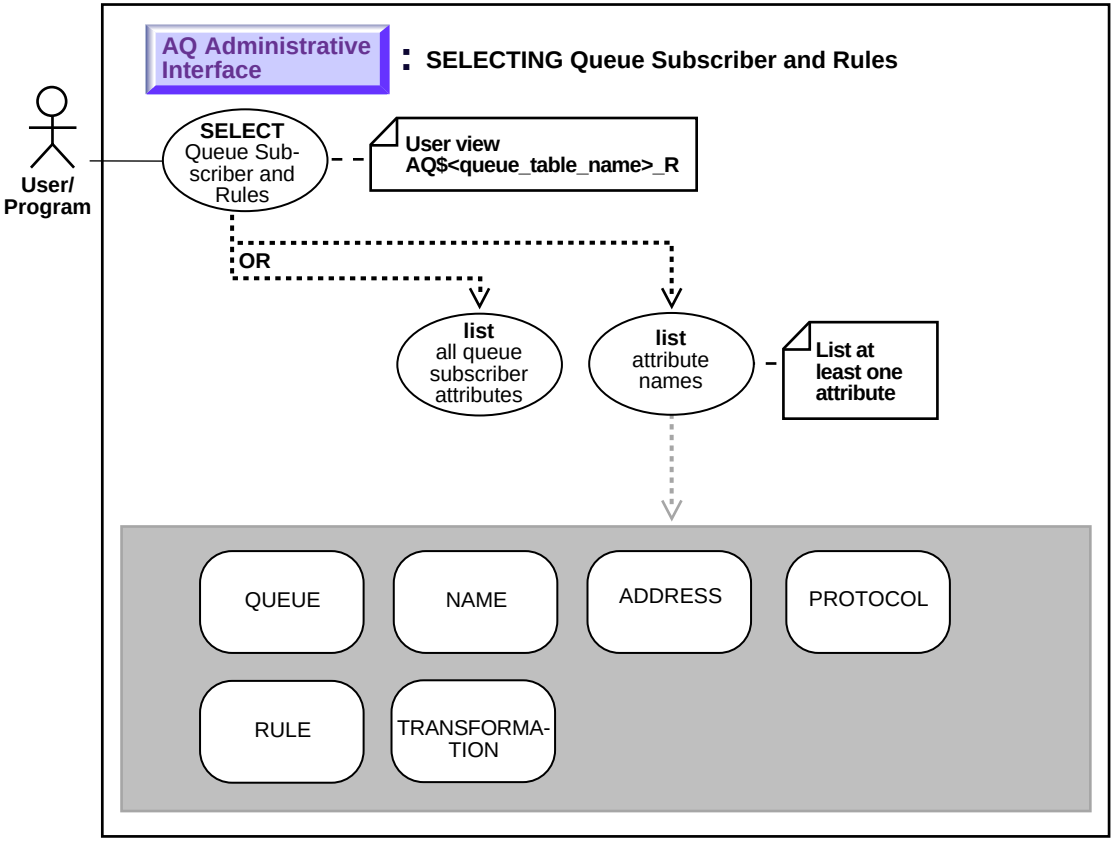
Column Name & Description	Null?	Type
QUEUE—name of Queue for which subscriber is defined	NOT NULL	VARCHAR2(30)
NAME—name of Agent		VARCHAR2(30)
ADDRESS—address of Agent		VARCHAR2(1024)
PROTOCOL—protocol of Agent		NUMBER
TRANSFORMATION—the name of the transformation can be null		VARCHAR2(61)

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, it is recommended that the view be used instead of this procedure to view queue subscribers.

Selecting Queue Subscribers and Their Rules

Figure 10–13 Use Case Diagram: Selecting Queue Subscribers and their Rules



Name of View

`AQ$<queue_table_name>_R`

Purpose

This view displays only the rule based subscribers for all queues in a given queue table including the text of the rule defined by each subscriber. This is a view of subscribers with rules defined on any queues of a given queue table. This view is generated when the queue table is created and is called aq$<queue_table_name>_r.

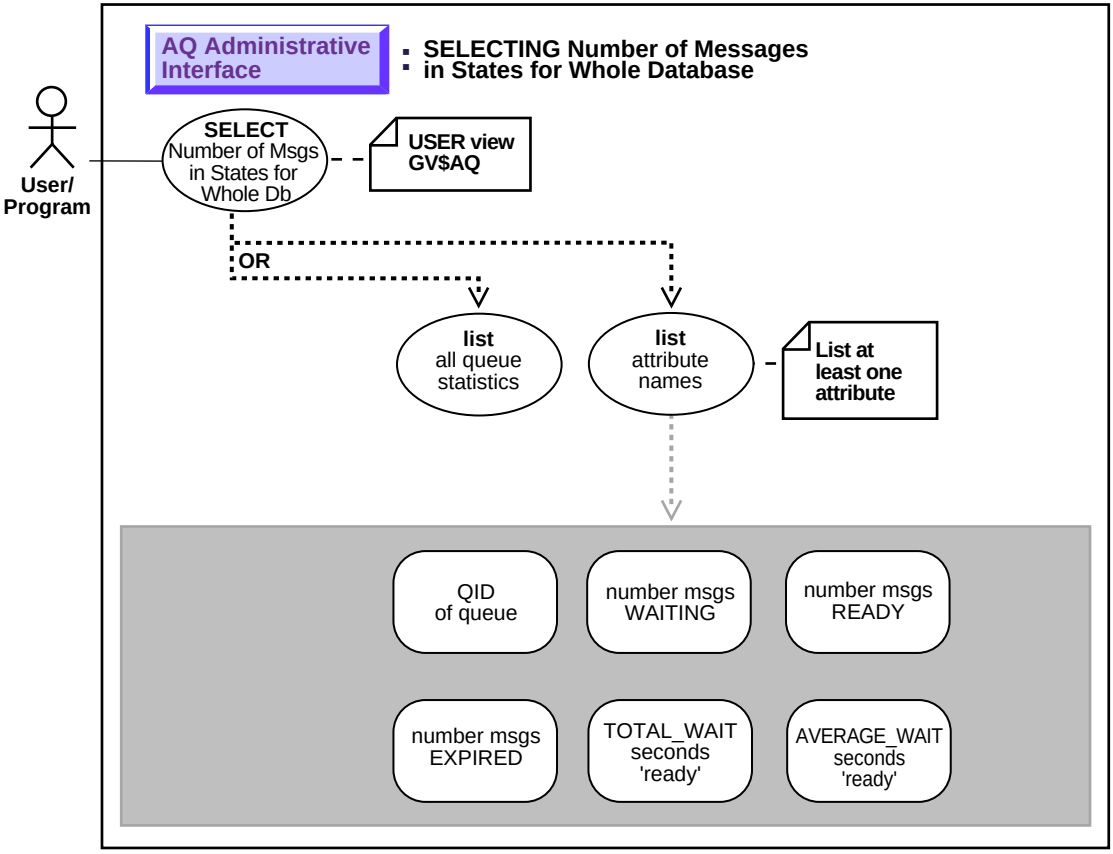
It is used to query subscribers for any or all the queues in this queue table. Note that this view is only created for 8.1-compatible queue tables. The view will also display the transformation for the subscriber if one was specified.

Table 10–13 AQ$<queue_table_name>_R

Column Name & Description	Null?	Type
QUEUE—name of Queue for which subscriber is defined	NOT NULL	VARCHAR2(30)
NAME—name of Agent		VARCHAR2(30)
ADDRESS—address of Agent		VARCHAR2(1024)
PROTOCOL—protocol of Agent		NUMBER
RULE—text of defined rule		VARCHAR2(30)
TRANSFORMATION—name of transformation specified, can be null		VARCHAR2(61)

Selecting the Number of Messages in Different States for the Whole Database

Figure 10–14 Selecting the Number of Messages in Different States for the Whole Database



Name of View

GV$AQ

Purpose

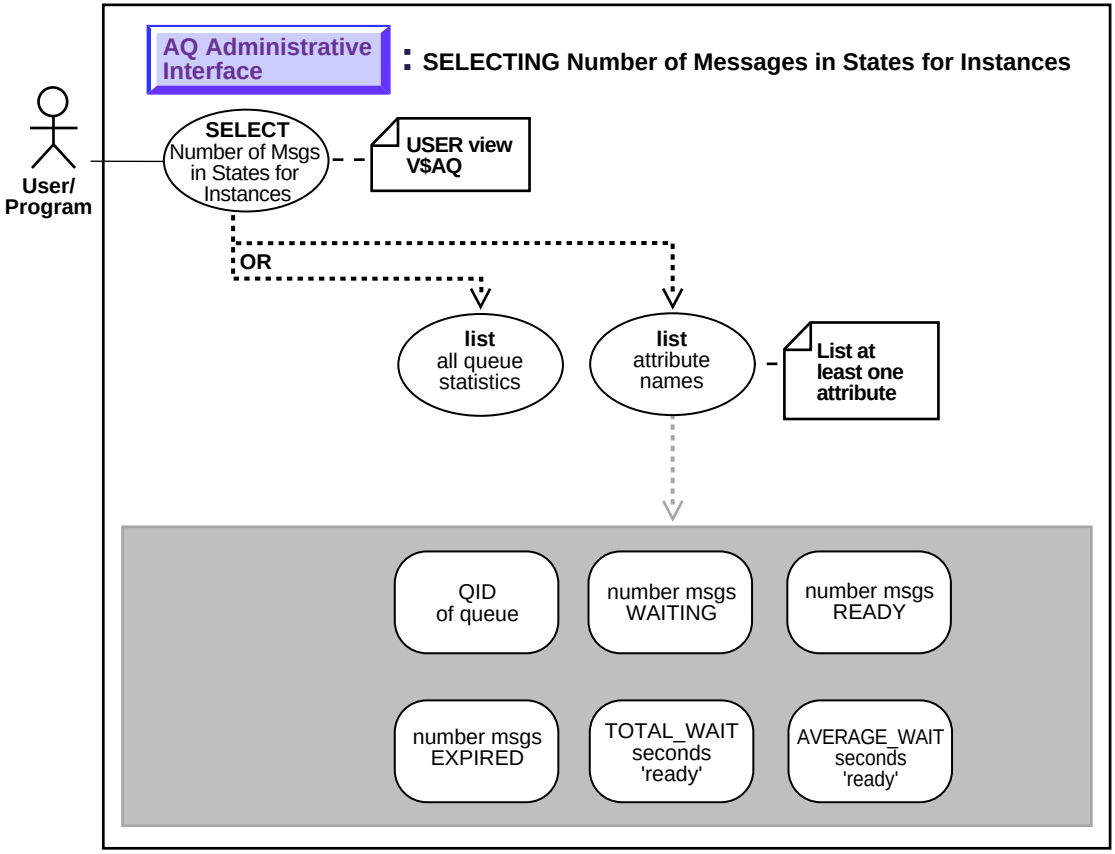
Provides information about the number of messages in different states for the whole database.

Table 10–14 AQ$<queue_table_name>_R

Column Name & Description	Null?	Type
QID—the identity of the queue. This is the same as the qid in user_queues and dba_queues.		NUMBER
WAITING—the number of messages in the state 'WAITING'.		NUMBER
READY—the number of messages in state 'READY'.		NUMBER
EXPIRED—the number of messages in state 'EXPIRED'.		NUMBER
TOTAL_WAIT—the number of seconds for which messages in the queue have been waiting in state 'READY'		NUMBER
AVERAGE_WAIT—the average number of seconds a message in state 'READY' has been waiting to be dequeued.		NUMBER

Selecting the Number of Messages in Different States for Specific Instances

Figure 10–15 Select the Number of Messages in Different States for Specific Instances



Name of View

V$AQ

Purpose

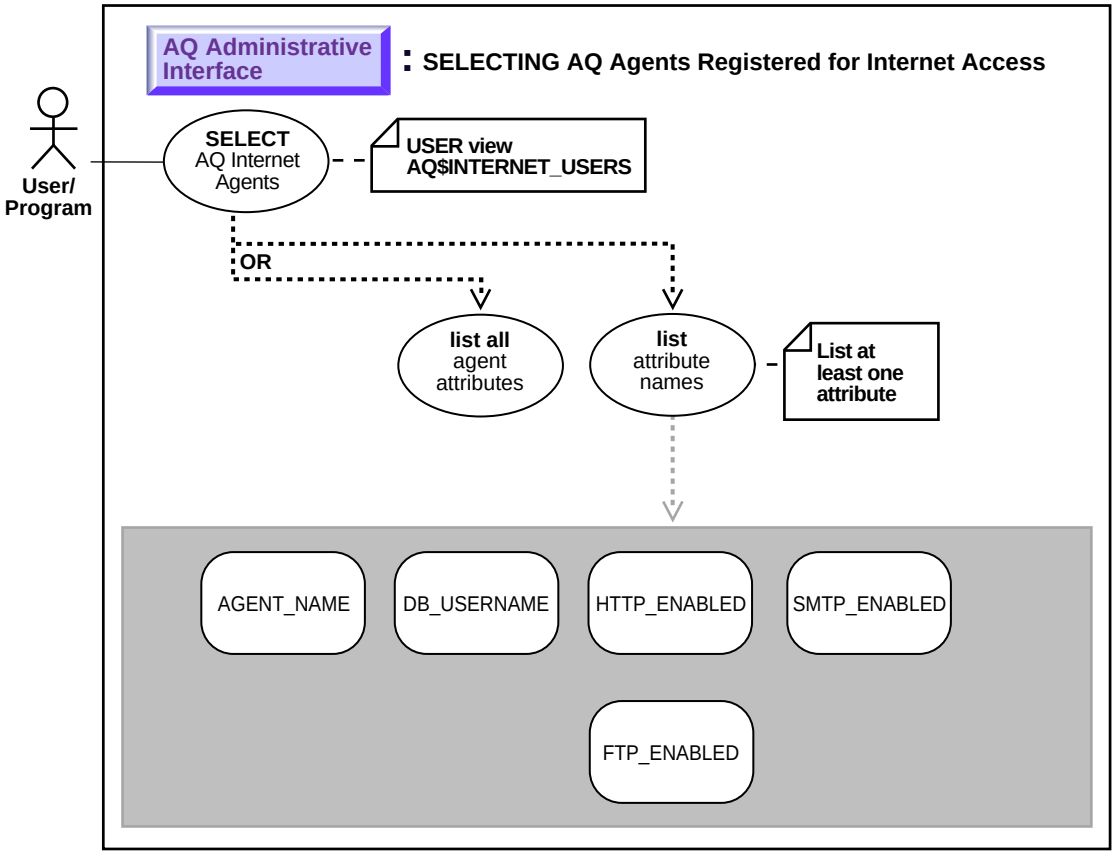
Provides information about the number of messages in different states for specific instances.

Table 10–15 AQ$<queue_table_name>_R

Column Name & Description	Null?	Type
QID — the identity of the queue. This is the same as the qid in user_queues and dba_queues.		NUMBER
WAITING — the number of messages in the state 'WAITING'.		NUMBER
READY — the number of messages in state 'READY'.		NUMBER
EXPIRED — the number of messages in state 'EXPIRED'.		NUMBER
TOTAL_WAIT — the number of seconds for which messages in the queue have been waiting in state 'READY'		NUMBER
AVERAGE_WAIT — the average number of seconds a message in state 'READY' has been waiting to be dequeued.		NUMBER

Selecting the AQ Agents Registered for Internet Access

Figure 10–16 Selecting the AQ Agents Registered for Internet Access



Name of View

`AQ$INTERNET_USERS`

Purpose

Provides information about the agents registered for Internet access to AQ. It also provides the list of database users that each Internet agent maps to.

Table 10–16 AQ$INTERNET_USERS

Column Name & Description	Null?	Type
AGENT_NAME—the name of the AQ Internet agent	NOT NULL	VARCHAR2(30)
DB_USERNAME—the name of the database user that this Internet agent maps to	NOT NULL	VARCHAR2(30)
HTTP_ENABLED—indicates whether this agent is allowed to access AQ via HTTP. Has a value of YES or NO		VARCHAR2(4)
SMTP_ENABLED—indicates whether this agent is allowed to access AQ via SMTP. Has a value of YES or NO		VARCHAR2(4)
FTP_ENABLED—indicates whether this agent is allowed to access AQ via FTP. Always has a value of NO in current release		VARCHAR2(4)

Selecting User Transformations

Name of View

USER_TRANSFORMATIONS

Purpose

This view displays all the transformations owned by the user. To view the transformation definition, query USER_ATTRIBUTE_TRANSFORMATIONS.

Table 10–17 USER_TRANSFORMATIONS

Column Name & Description	Null?	Type
TRANSFORMATION_ID—unique id for the transformation		NUMBER
NAME—transformation name		VARCHAR2(30)
FROM_TYPE—source type name		VARCHAR2(61)
TO_TYPE—target type name		VARCHAR2(61)

Selecting User Transformation Functions

Name of View

USER_ATTRIBUTE_TRANSFORMATIONS

Purpose

This view displays the transformation functions for all the transformations of the user.

Table 10–18 USER_ATTRIBUTE_TRANSFORMATIONS

Column Name & Description	Null?	Type
TRANSFORMATION_ID—unique id of the transformation		NUMBER
NAME—transformation name		VARCHAR2(30)
FROM_TYPE—source type name		VARCHAR2(61)
TO_TYPE— target type name		VARCHAR2(61)
ATTRIBUTE—target type attribute number		NUMBER
ATRIBUTE_ TRANSFORMATION—transformation function for the attribute		VARCHAR2(4000)

Selecting All Transformations

Name of View

DBA_TRANSFORMATIONS

Purpose

This view displays all the transformations in the database. These transformations can be specified with Advanced Queue operations like enqueue, dequeue and subscribe to automatically integrate transformations in AQ messaging. This view is accessible only to users having DBA privileges.

Table 10–19 DBA_TRANSFORMATIONS

Column Name & Description	Null?	Type
TRANSFORMATION_ID—unique identifier for the transformation		NUMBER
OWNER—owning user of the transformation		VARCHAR2(30)
NAME— transformation name		VARCHAR2(30)
FROM_TYPE—source type name		VARCHAR2(61)
TO_TYPE—target type name		VARCHAR2(61)
Namespace—one for transformations created by the Oracle transformation engine. Transformations from third party-transformation engines are in different namespaces.		
From_type_schema—owning user of the source type		
From_type_name—source type of the transformation		
To_type_Schema—owning user of the destination type		
To_type_name—destination type of the transformation. The transformation takes an object of the source type and returns an object of the destination type.		
Transformation_type—type of the transformation. Values: SQL and XSL		
Attribute_Name—attribute name of the destination type for which the transformation is being specified.		
Transformation_Expression—can be a SQL expression, P/LSQL function, or an XSL document		
Comment—user-specified comment.		

Selecting All Transformation Functions

Name of View

`DBA_ATTRIBUTE_TRANSFORMATIONS`

Purpose

This view displays the transformation functions for all the transformations in the database.

Table 10–20 DBA_ATTRIBUTE_TRANSFORMATIONS

Column Name & Description	Null?	Type
TRANSFORMATION_ID—unique id of the transformation		NUMBER
OWNER— transformation owner		VARCHAR2(30)
NAME—transformation name		VARCHAR2(30)
FROM_TYPE— source type name		VARCHAR2(61)
TO_TYPE— target type name		VARCHAR2(61)
ATTRIBUTE— target type attribute number		NUMBER
ATRIBUTE_ TRANSFORMATION—transformation function for the attribute		VARCHAR2(4000)

11

Operational Interface: Basic Operations

In this chapter we describe the operational interface to Oracle Advanced Queuing in terms of use cases. That is, we discuss each operation (such as "Enqueue a Message") as a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: Operational Interface — Basic Operations" on page 11-2).

A summary figure, "Use Case Diagram: Operational Interface — Basic Operations", locates all the use cases in a single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case in which you are interested by clicking on the relevant use case title.

Each use case is laid out as follows:

- *Use case figure*. A figure that depicts the use case.
- *Purpose*. The purpose of this use case.
- *Usage Notes*. Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment which illustrate the use case.

Use Case Model: Operational Interface — Basic Operations

Table 11–1, "Use Case Model: Operational Interface" indicates with a + where examples are provided for specific use cases and in which programmatic environment.

The table refers to programmatic environments with the following abbreviations:

- **P** — PL/SQL using the DBMS_AQADM and DBMS_AQ packages
- **O** — C using OCI (Oracle Call Interface)
- **V** — Visual Basic using OO4O (Oracle Objects for OLE)
- **J** — Java (native AQ) using JDBC (Java Database Connectivity)
- **JM** — Java (JMS standard) using JDBC (Java Database Connectivity)

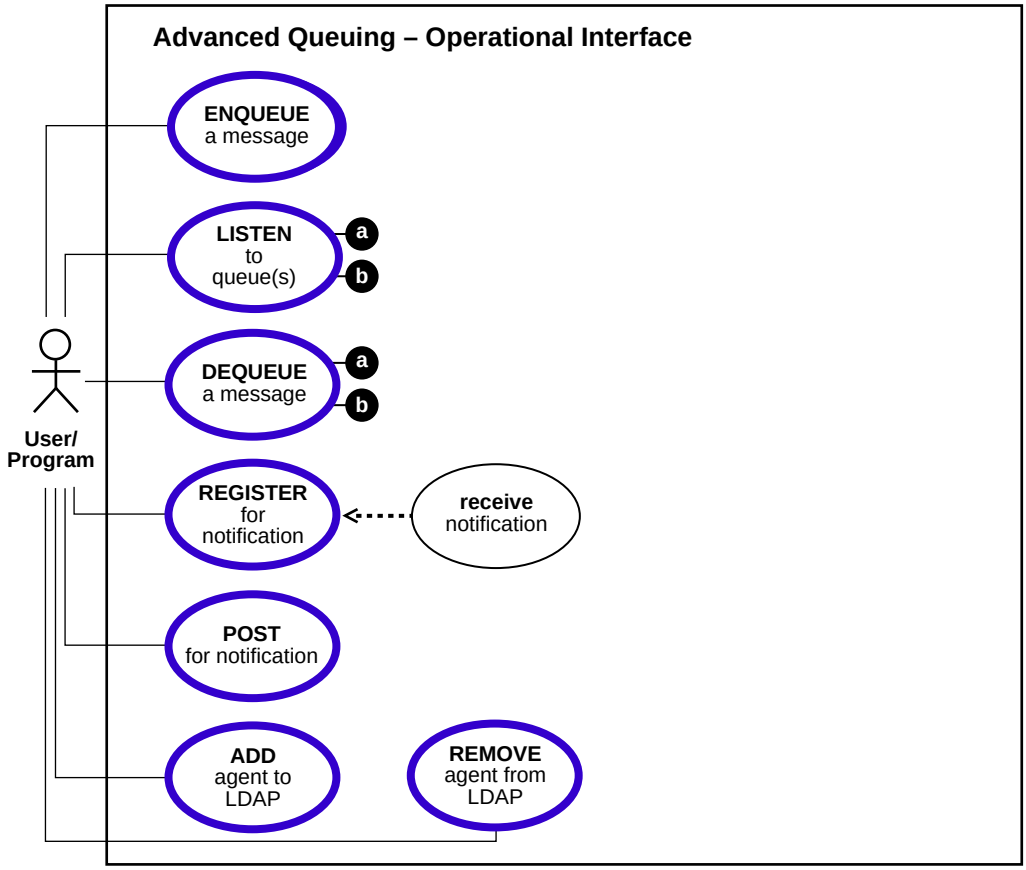
Table 11–1 Use Case Model: Operational Interface

Use Case	Programmatic Environment Examples				
	P	O	V	J	JM
Enqueuing a Message on page 11-5					
Enqueuing a Message [Specify Options] on page 11-7	+		+		+
Enqueuing a Message [Specify Message Properties] on page 11-10	+		+		+
Enqueuing a Message [Add Payload] on page 11-15	+		+		+
Listening to One (Many) Queue(s) on page 11-24					
Listening to One (Many) Single-Consumer Queues on page 11-26	+	+	+		
Listening to One (Many) Multi-Consumer Queue(s) on page 11-38	+	+	+		
Dequeuing a Message on page 11-47					
Dequeuing a Message from a Single-Consumer Queue [SpecifyOptions] on page 11-50	+		+		+
Dequeuing a Message from a Multi-Consumer Queue [Specify Options] on page 11-55	+		+		+
Registering for Notification on page 11-58					
Registering for Notification [Specifying Subscription Name — Single-Consumer Queue] on page 11-61		+			

Table 11–1 Use Case Model: Operational Interface (Cont.)

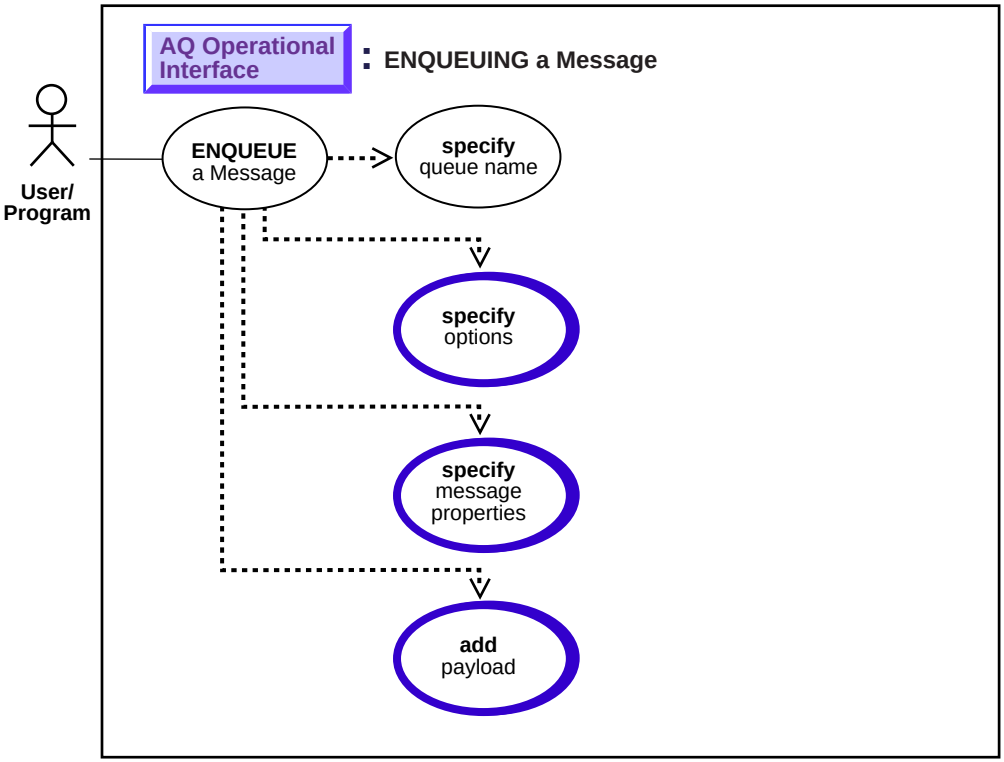
Use Case		Programmatic Environment Examples
Registering for Notification [Specifying Subscription Name — Multi-Consumer Queue] on page 11-62		+
Posting for Subscriber Notification on page 11-69	+	+
Adding an Agent to the LDAP Server on page 11-73		
Removing an Agent from the LDAP Server on page 11-75		

Figure 11–1 Use Case Model Diagram: Operational Interface



Enqueuing a Message

Figure 11–2 Use Case Diagram: Enqueuing a Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose:

Adds a message to the specified queue.

Usage Notes

- If a message is enqueued to a multi-consumer queue with no recipient and the queue has no subscribers (or rule-based subscribers that match this message), then the Oracle error ORA 24033 is raised. This is a warning that the message will be discarded since there are no recipients or subscribers to whom it can be delivered.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, ENQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQOracleQueue.enque

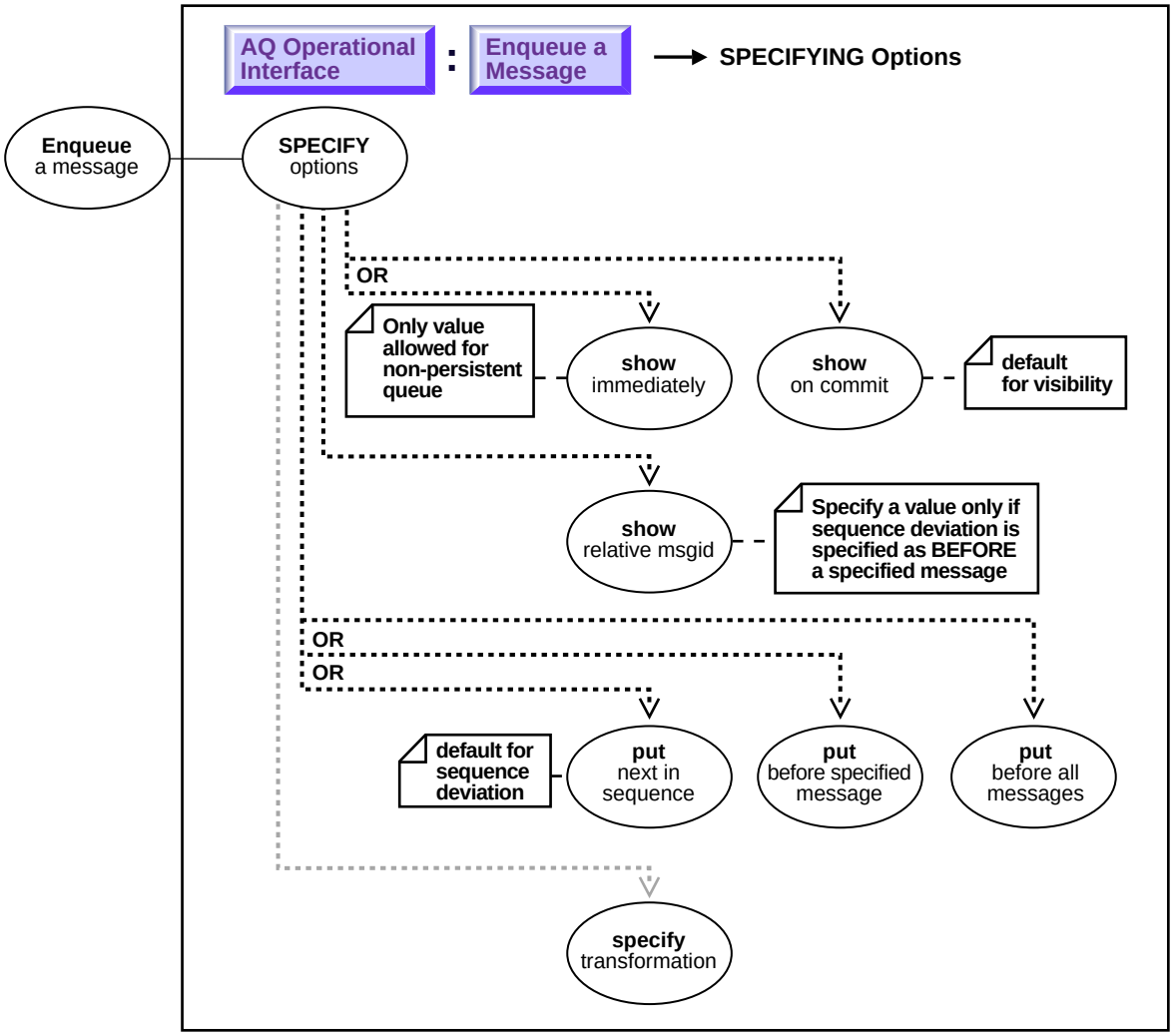
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages on page 11-17
- Java (JDBC): Enqueue a message (add payload) on page 11-19
- Visual Basic (OO4O): Enqueue a message on page 11-22

Enqueuing a Message [Specify Options]

Figure 11–3 Use Case Diagram: Enqueuing a Message [Specify Options]



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To specify the options available for the enqueue operation.

Usage Notes

Do not use the `immediate` option when you want to use LOB locators since LOB locators are valid only for the duration of the transaction. As the `immediate` option automatically commits the transaction, your locator will not be valid.

- 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 has to be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message has to 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 non-persistent queues.
- Only local recipients are supported are supported for non-persistent queues.
- If a transformation is specified, it will be applied to the message before enqueuing it to the queue. The transformation must map the message into an object whose type is the ADT type of the queue.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, ENQUEUE Procedure

- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQ Enqueue Option

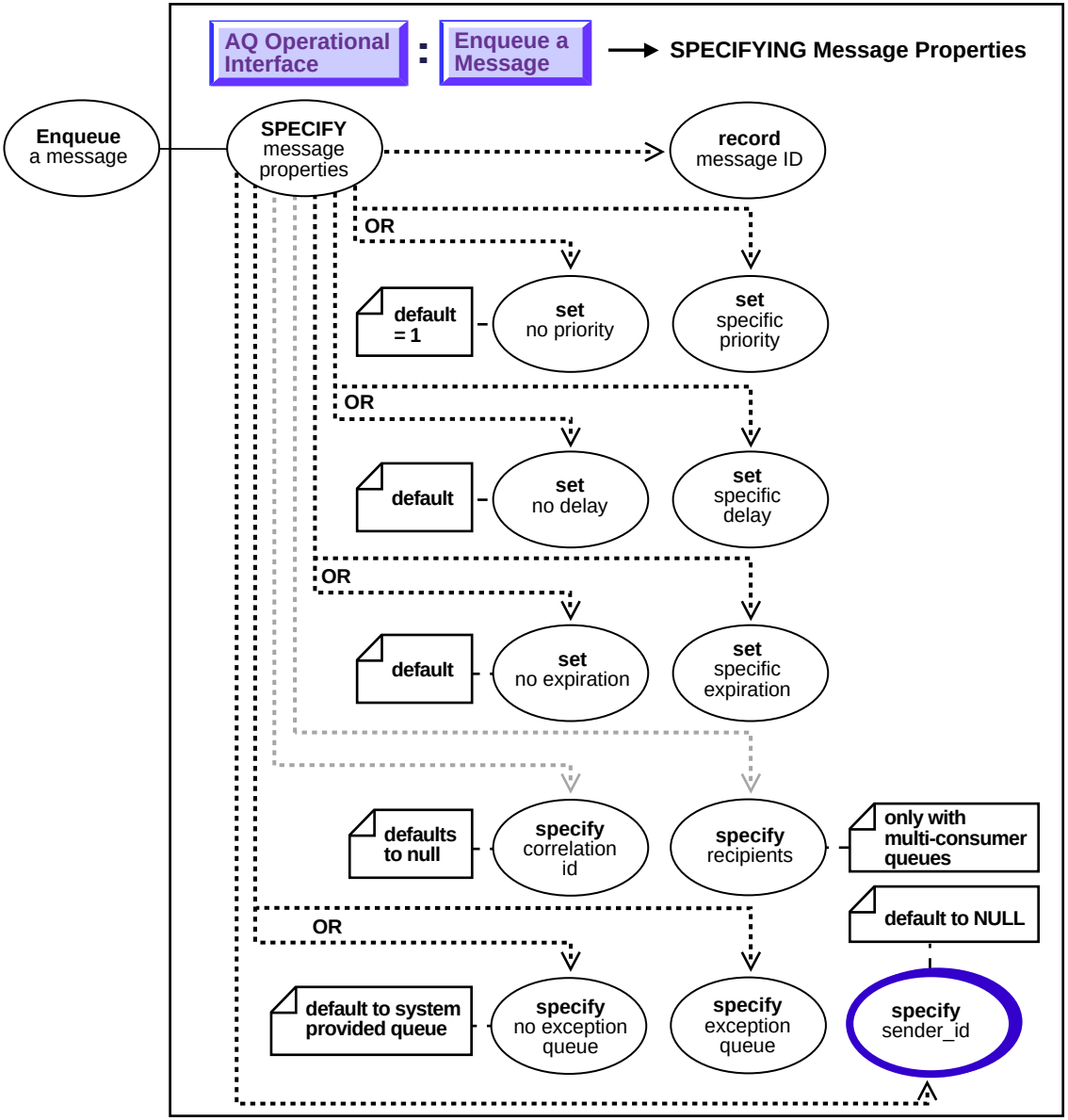
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages on page 11-17
- Java (JDBC): Enqueue a message (add payload) on page 11-19
- Visual Basic (OO4O): Enqueue a message on page 11-22

Enqueuing a Message [Specify Message Properties]

Figure 11–4 Use Case Diagram: Enqueuing a Message [Specify Message Properties]



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

The *Message Properties* describe the information that is used by AQ to manage individual messages. These are set at enqueue time and their values are returned at dequeue time.

Usage Notes

- 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 should remember to start the QMN processes for the database if you intend to use the delay and expiration features of AQ.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, ENQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQMessageProperty

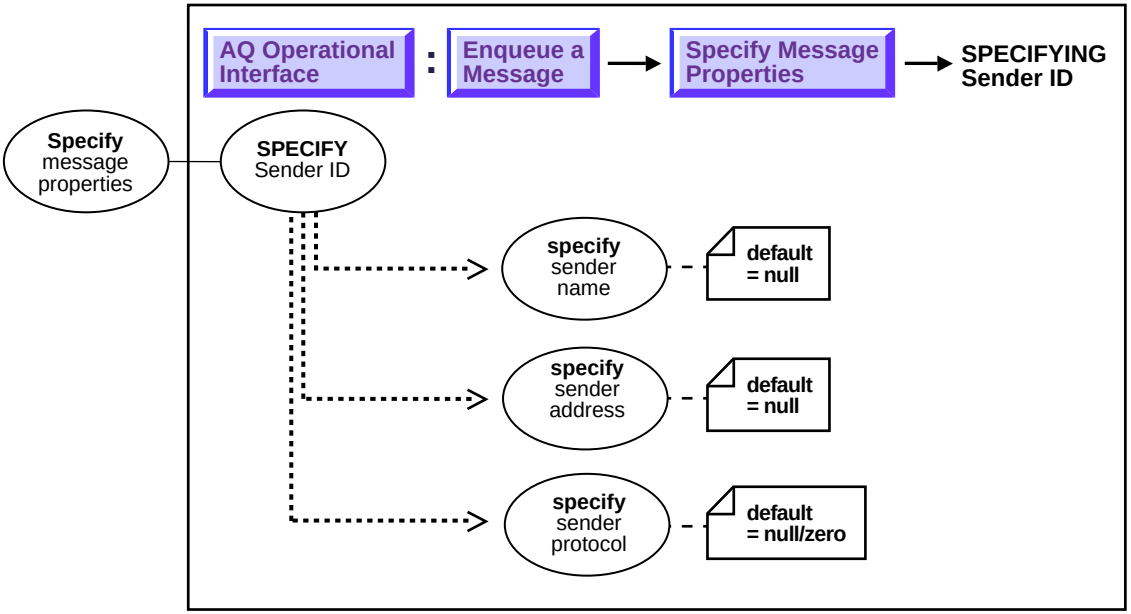
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages on page 11-17
- Java (JDBC): Enqueue a message (add payload) on page 11-19
- Visual Basic (OO4O): Enqueue a message on page 11-22

Enqueuing a Message [Specify Message Properties [Specify Sender ID]]

Figure 11–5 Use Case Diagram: Enqueuing a Message [Specify Message Properties [Specify Sender ID]]



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To identify the sender (producer) of a message.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, ENQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQMessageProperty.setsender

For more information about Agent see:

- "Agent Type (aq$_agent)" on page 2-3

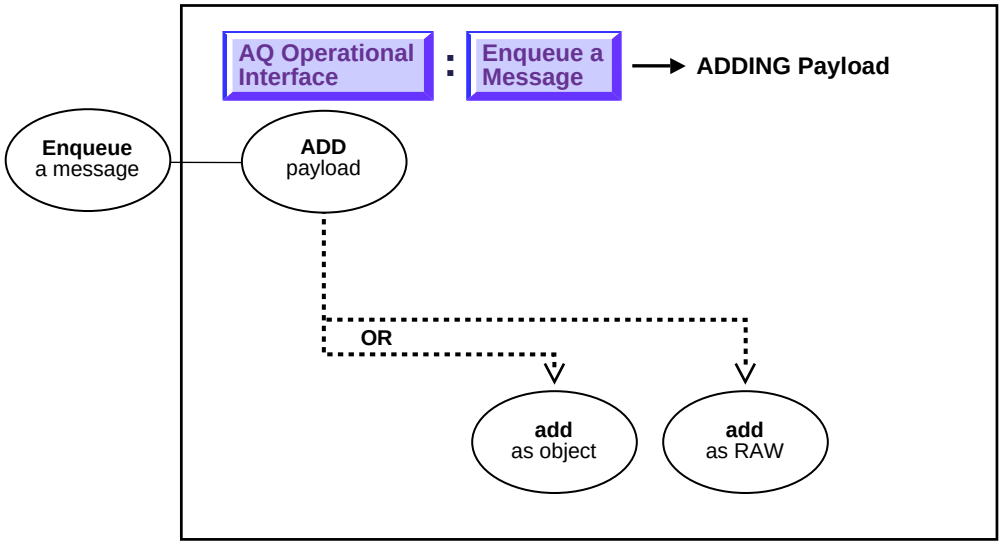
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages on page 11-17
- Java (JDBC): Enqueue a message (add payload) on page 11-19
- Visual Basic (OO4O): Enqueue a message on page 11-22

Enqueuing a Message [Add Payload]

Figure 11–6 Use Case Diagram: Enqueuing a Message [Add Payload]



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

Usage Notes

To store a payload of type RAW, AQ will create 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 AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, ENQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQOracleQueue.enque

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages on page 11-17
- Java (JDBC): Enqueue a message (add payload) on page 11-19
- Visual Basic (OO4O): Enqueue a message on page 11-22

PL/SQL (DBMS_AQ Package): Enqueue of Object Type Messages

Note: You may need to 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;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   Queue_table            =>  'aq.objmsgs_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);
```

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;
```

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;
```

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;
```

Java (JDBC): Enqueue a message (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();
}
```

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"

Set OraMsg.Value = OraObj
Msgid = Q.Enqueue
```

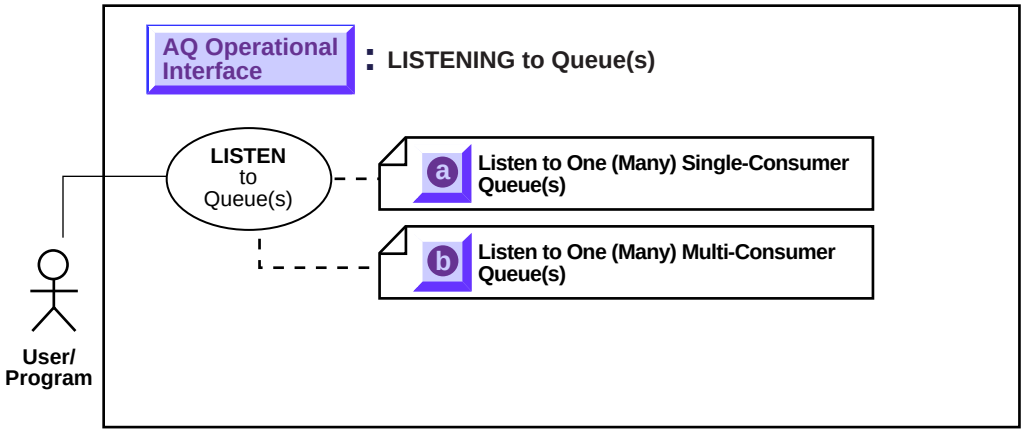
Enqueuing messages of type RAW

```
'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 non-default 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
```

Listening to One (Many) Queue(s)

Figure 11–7 Use Case Diagram: Listening to One(Many) Queue(s)



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To monitor one or more queues on behalf of a list of agents.

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.

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, only the first agent listed is returned. If there are no messages found when the wait time expires, 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 the specified queues. The interested agent must still dequeue the relevant message.

Note that you cannot call `listen` on nonpersistent queues.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, LISTEN procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ Object > Monitoring Messages
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, AQSession.listen

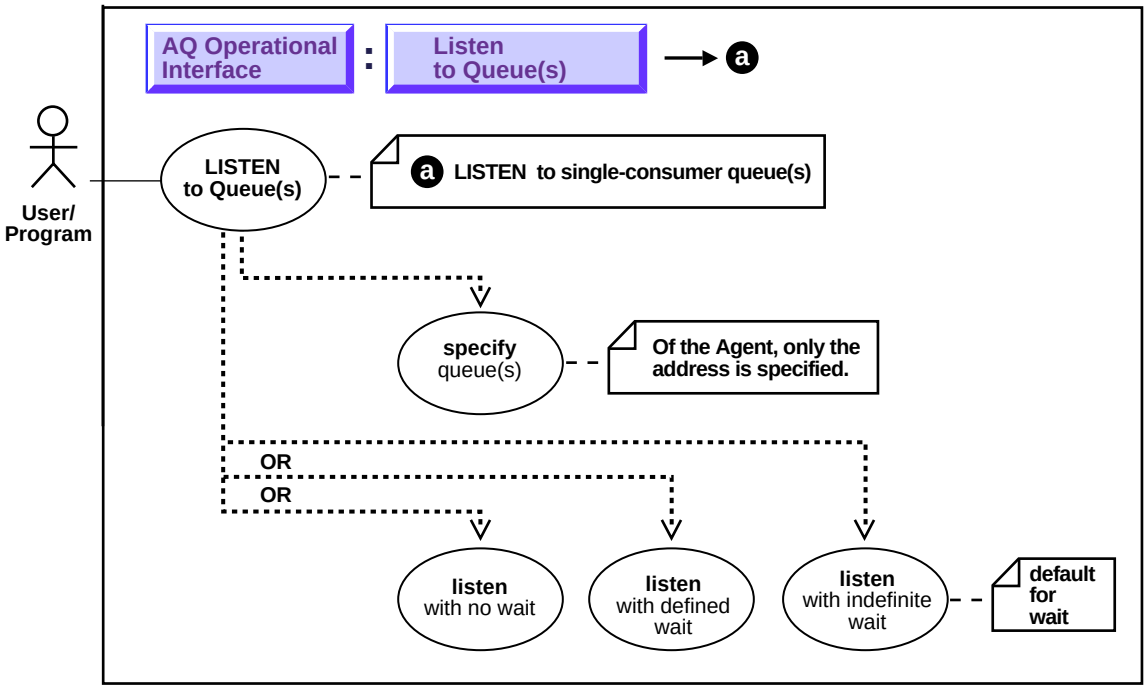
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Listen to Queues on page 11-27
- C (OCI): Listen to Single-Consumer Queue(s) on page 11-29

Listening to One (Many) Single-Consumer Queues

Figure 11–8 Use Case Diagram: Listening to One (Many) Single-Consumer Queues



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, LISTEN procedure
- C (OCI): *Oracle Call Interface Programmer's Guide* Relational Functions, OCIAQListen
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ Object > Monitoring Messages

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Listen to Queues on page 11-27
- Java (JDBC): Listen to Queues
- C (OCI): Listen to Single-Consumer Queue(s) on page 11-29

PL/SQL (DBMS_AQ Package): Listen to Queues

```
/* The listen call allows you to monitor a list of queues for messages for
   specific agents. You need to have dequeue privileges for all the queues
   you wish to monitor. */
```

Listen to Single-Consumer Queue (Timeout of Zero).

```
DECLARE
   Agent_w_msg       aq$_agent;
   My_agent_list     dbms_aq.agent_list_t;

BEGIN
   /* NOTE:  MCQ1, MCQ2, MCQ3 are multi-consumer 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 time-out 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;
```

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);
    }

}
```

C (OCI): Listen to Single-Consumer Queue(s)

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");

}
```

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 )
```

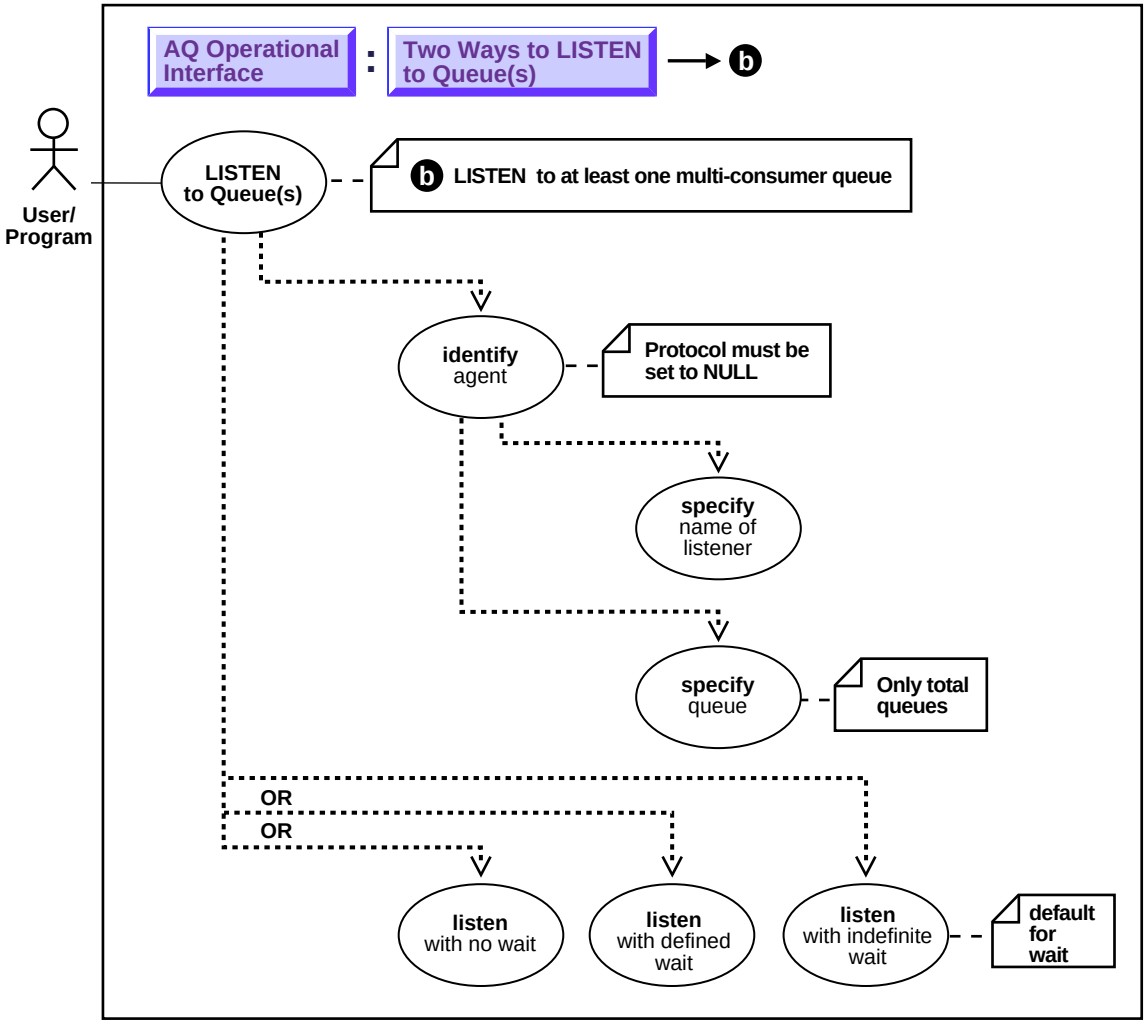
```
  {
    printf("agent was NULL \n");
    return;
  }
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&appname, &appsz, OCI_ATTR_AGENT_NAME, errhp));
  checkerr(errhp, OCIAttrGet(agent, OCI_DTYPE_AQAGENT,
     (dvoid *)&queue, &queuesz, OCI_ATTR_AGENT_ADDRESS, errhp));
  if (!appsz)
     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");

}
```

Listening to One (Many) Multi-Consumer Queue(s)

Figure 11–9 Use Case Diagram: Listening to One(Many) Multi-Consumer Queue(s)



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, LISTEN procedure
- C (OCI): *Oracle Call Interface Programmer's Guide* Relational Functions, OCIAQListen
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ Object > Monitoring Messages
- Feature not available through the Java API

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Listen to Queue(s) on page 11-40
- C (OCI): Listen to Multi-Consumer Queue(s) on page 11-41

PL/SQL (DBMS_AQ Package): Listen to Queue(s)

```
/* The listen call allows you to monitor a list of queues for messages for
   specific agents. You need to have dequeue privileges for all the queues
   you wish to monitor. */
```

Listen to Multi-Consumer Queue (Timeout of Zero).

```
DECLARE
   Agent_w_msg      aq$_agent;
   My_agent_list    dbms_aq.agent_list_t;

BEGIN
   /* NOTE:  MCQ1, MCQ2, MCQ3 are multi-consumer queues  in SCOTT's schema
   *        SCQ1, SCQ2, SCQ3 are single consumer queues in SCOTT's schema
   */
    Qlist(1):= aq$_agent('agent1', 'MCQ1',  NULL);
    Qlist(2):= aq$_agent('agent2', 'scott.MCQ2', NULL);
    Qlist(3):= aq$_agent('agent3', 'scott.MCQ3', NULL);

   /* Listen with a time-out 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;
   /
```

Listen to Mixture of Multi-Consumer Queues (Timeout 100 Seconds).

```
DECLARE
   Agent_w_msg      aq$_agent;
   My_agent_list    dbms_aq.agent_list_t;

BEGIN
   /* NOTE:  MCQ1, MCQ2, MCQ3 are multi-consumer queues  in SCOTT's schema
   *        SCQ1, SCQ2, SCQ3 are single consumer queues in SCOTT's schema
   */
   Qlist(1):= aq$_agent('agent1', 'MCQ1',  NULL);
   Qlist(2):= aq$_agent(NULL, 'scott.SQ1', NULL);
   Qlist(3):= aq$_agent('agent3', 'scott.MCQ3', NULL);
   /* Listen with a time-out of 100 seconds */
   DBMS_AQ.LISTEN(
```

```
      Agent_list   =>   My_agent_list,
      Wait         =>   100,
      Agent        =>    agent_w_msg);
      DBMS_OUTPUT.PUT_LINE('Message in Queue :- ' ||  agent_w_msg.address
                           || 'for agent' || agent_w_msg.name);
      DBMS_OUTPUT.PUT_LINE('');
   END;
   /
```

C (OCI): Listen to Multi-Consumer Queue(s)

Listening to Multi-consumer Queues with a Zero Timeout, a Timeout of 120 Seconds, and a Timeout of 100 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;
     }
}

void SetAgent(OCIAQAgent *agent,
         text       *appname,
         text       *queue,
         OCIError   *errhp,
         OCIEnv     *envhp);

void GetAgent(OCIAQAgent *agent,
         OCIError   *errhp);

/*-----------------------------------------------------------------*/
/* OCI Listen examples for multi-consumers                         */
/*                                                                 */
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);
}

/* main from AQ Listen to Multi-Consumer Queue(s) */

/*  int main() */
int main(char *argv, int argc)
{
    OCIEnv     *envhp;
    OCIServer  *srvhp;
    OCIError   *errhp;
    OCISvcCtx  *svchp;
    OCISession *usrhp;
    OCIAQAgent *agent_list[3];
```

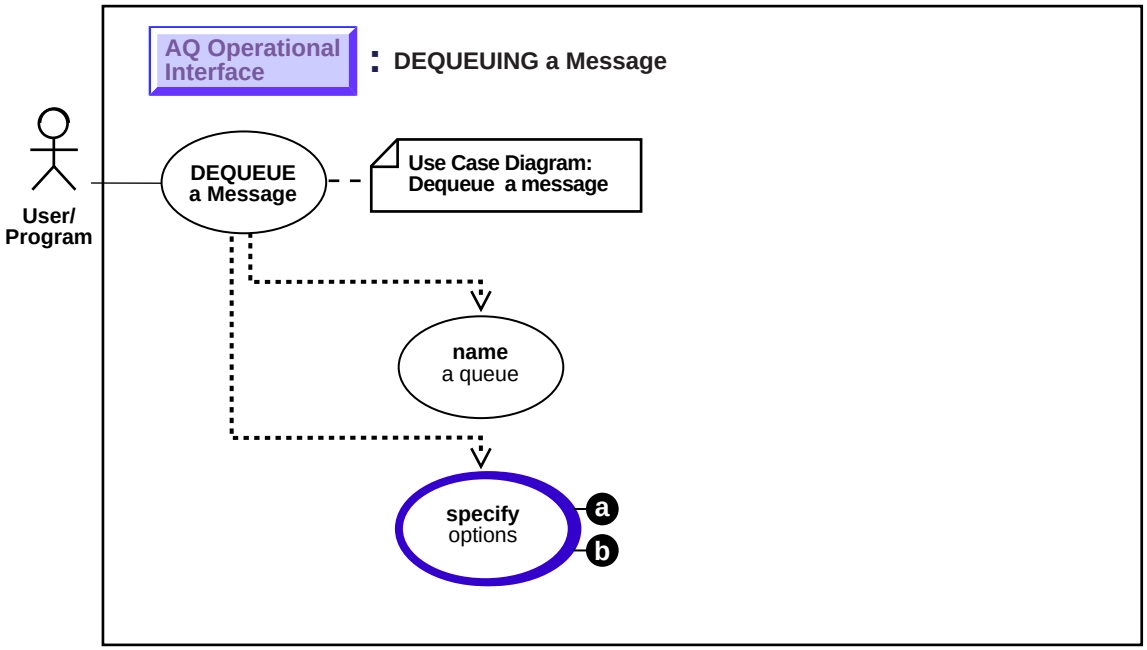
```
  OCIAQAgent *agent;
  int         i;

/* 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++)
{
   OCIDescriptorAlloc(envhp, (dvoid **)&agent_list[i],
       OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
}

/*
 *  MCQ1, MCQ2, MCQ3 are multi-consumer queues in SCOTT's schema
 */
/* Listening to Multi-consumer Queues with Zero Timeout */

SetAgent(agent_list[0], "app1", "MCQ1", errhp);
SetAgent(agent_list[1], "app2", "MCQ2", errhp);
SetAgent(agent_list[2], "app3", "MCQ3", errhp);

checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 0, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");


/* Listening to Multi-consumer Queues with Timeout of 120 Seconds */

SetAgent(agent_list[0], "app1", "SCOTT.MCQ1", errhp);
SetAgent(agent_list[1], "app2", "SCOTT.MCQ2", errhp);
SetAgent(agent_list[2], "app3", "SCOTT.MCQ3", errhp);

checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 120, &agent, 0));

printf("MESSAGE for :- \n");
GetAgent(agent, errhp);
printf("\n");


/* Listening to a Mixture of Single and Multi-consumer Queues
```

```
   * with a Timeout of 100 Seconds
   */

  SetAgent(agent_list[0], "app1", "SCOTT.MCQ1", errhp);
  SetAgent(agent_list[1], "app2", "SCOTT.MCQ2", errhp);
  SetAgent(agent_list[2], (text *)0, "SCOTT.SCQ3", errhp);

  checkerr(errhp, OCIAQListen(svchp, errhp, agent_list, 3, 100, &agent, 0));

  printf("MESSAGE for :- \n");
  GetAgent(agent, errhp);
  printf("\n");

}
```

Dequeuing a Message

Figure 11–10 Use Case Diagram: Dequeuing a Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

Dequeues a message from the specified queue.

Usage Notes

Search criteria and dequeue order for messages:

- The search criteria for messages to be dequeued is determined by the *consumer name*, *msgid* and *correlation* parameters in the dequeue options. Msgid uniquely identifies the message to be dequeued. Correlation identifiers are application-defined identifiers that are not interpreted by AQ.
- Only messages in the `READY` state are dequeued unless a msgid is specified.
- The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the msgid 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.

Navigating through a queue

The default `NAVIGATION` parameter during dequeue is `NEXT MESSAGE`. This means that subsequent dequeues will 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 will be 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 needs to 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 may 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, AQ will have to continually generate the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST MESSAGE` option is specified, AQ will use a new snapshot for every dequeue command.

Dequeue by Message Grouping

- Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, this will effectively form 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 will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic 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 will time-out after the specified `WAIT` period.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, DEQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQOracleQueue.dequeue

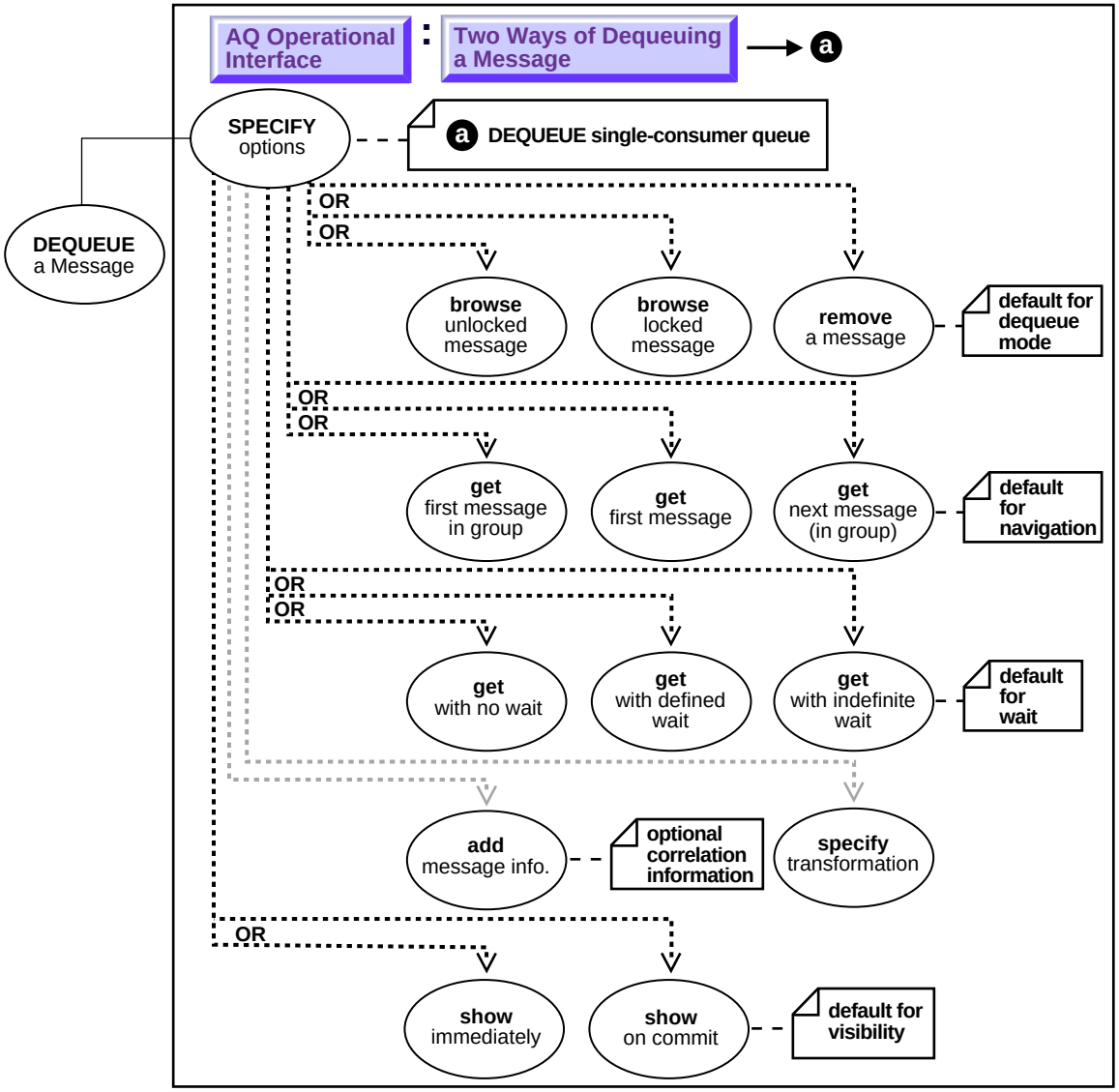
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Dequeue of Object Type Messages on page 11-52
- Java (JDBC): Dequeue a message from a single consumer queue (specify options) on page 11-52
- Visual Basic (OO4O): Dequeue a message on page 11-53

Dequeuing a Message from a Single-Consumer Queue [SpecifyOptions]

Figure 11–11 Use Case Diagram: Dequeuing a Message from a Single-Consumer Queue



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To specify 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`s.

The transformation, if specified, is applied before returning the message to the caller. The transformation should be defined to map the queue ADT type to the return type desired by the caller.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, DEQUEUE procedure
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQDequeueOption

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_AQ Package): Dequeue of Object Type Messages on page 11-52
- Java (JDBC): Dequeue a message from a single consumer queue (specify options) on page 11-52
- Visual Basic (OO4O): Dequeue a message on page 11-53

PL/SQL (DBMS_AQ Package): 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;
```

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();
}
```

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
```

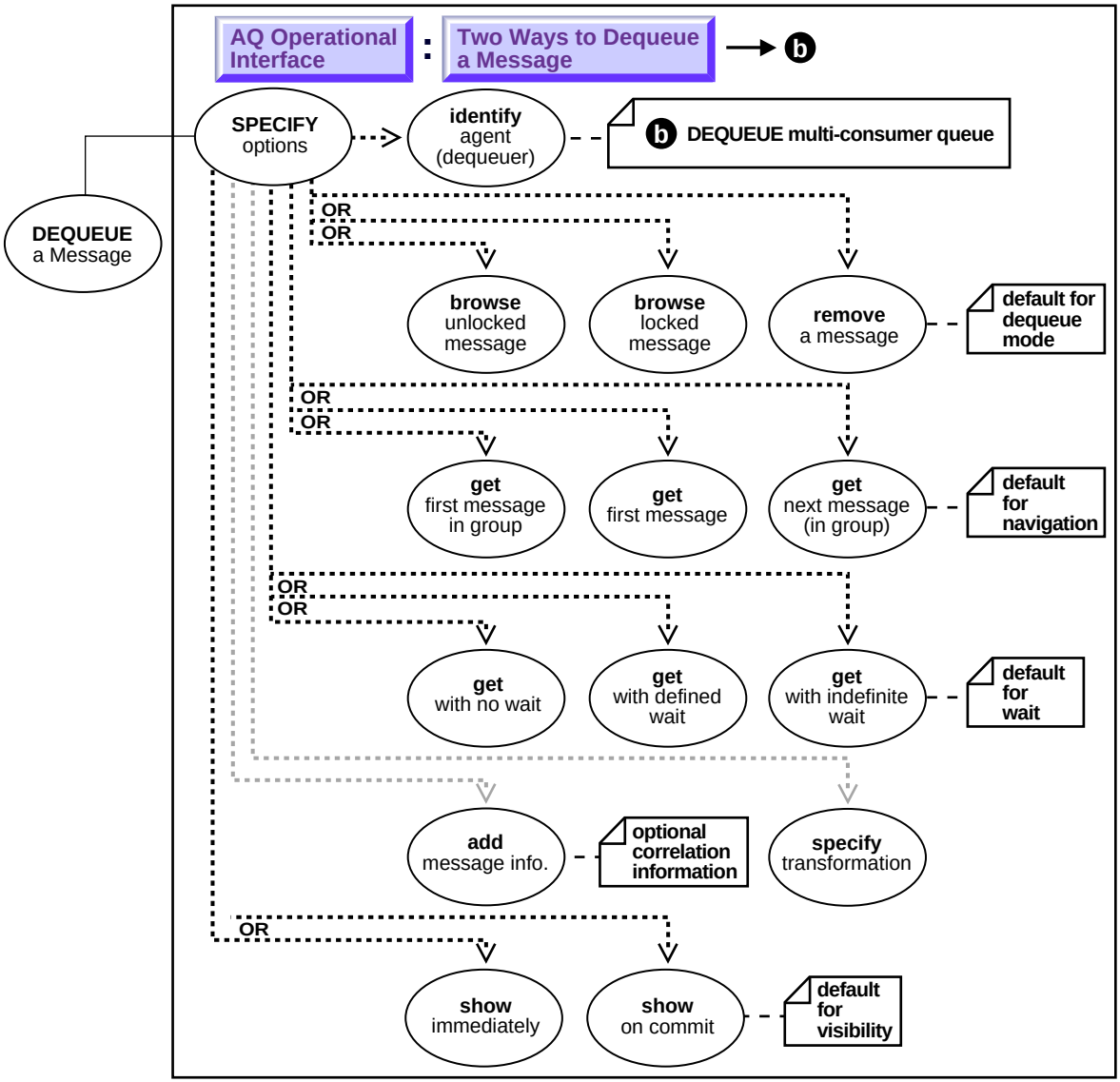
```
' of "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_NEXT_MSG
Q.Dequeue()

'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 Multi-Consumer Queue [Specify Options]

Figure 11–12 Use Case Diagram: Dequeuing a Message from a Multi-Consumer Queue



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose:

To specify the options available for the dequeue operation.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* DBMS_AQ, DEQUEUE procedure
- Visual Basic (OO4O): There is no applicable syntax reference for this use case
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQDequeueOption

Examples

Examples in the following programmatic environments are provided:

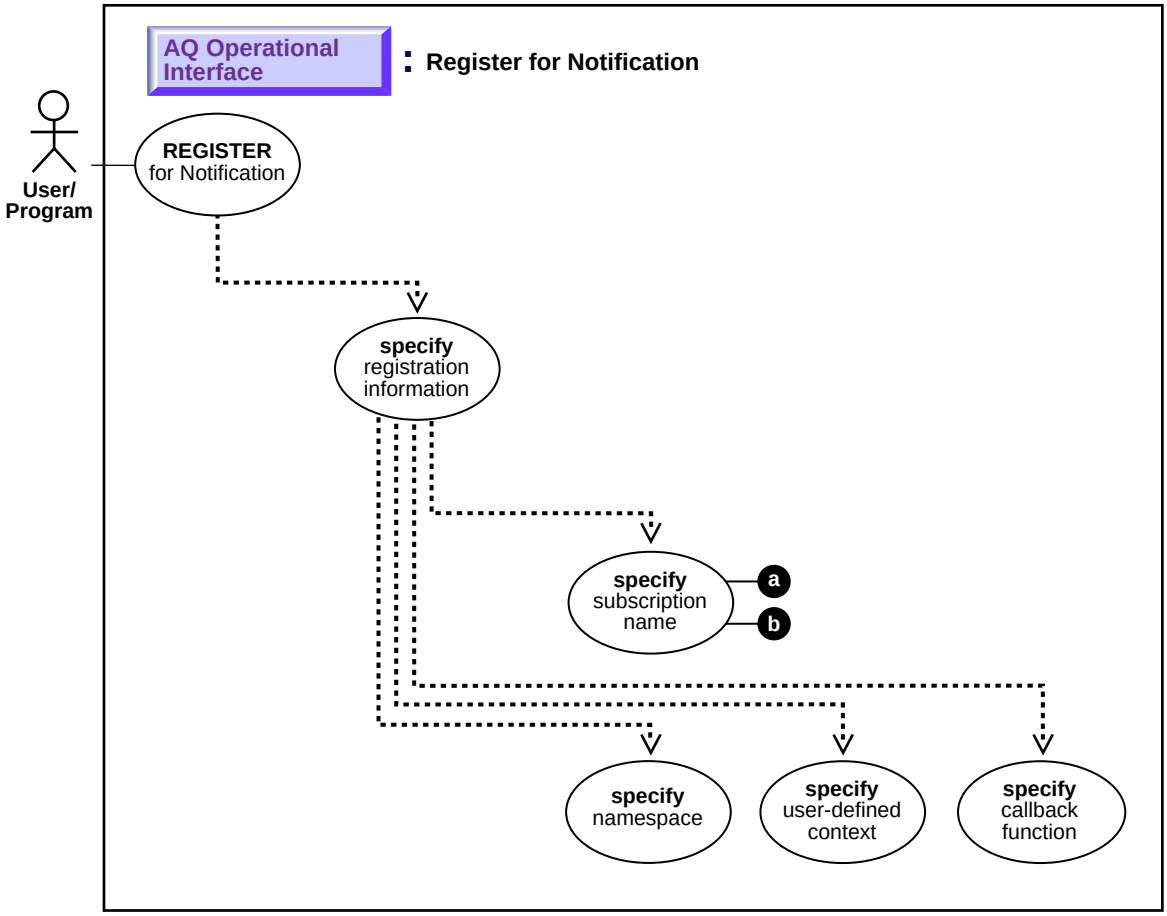
- Java (JDBC): Dequeue a message from a multi consumer queue (specify options) on page 11-57

Java (JDBC): Dequeue a message from a multi consumer 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();
}
```

Registering for Notification

Figure 11–13 Use Case Diagram: Registering for Notification



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To register a callback for message notification.

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 issues a registration call which specifies a callback. When messages are received that match the subscription criteria, the callback is invoked. The callback may then issue 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`()

For more information about the OCI operation `Register for Notification` see:

- *Oracle Call Interface Programmer's Guide*

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): Not available.
- C (OCI): *Oracle Call Interface Programmer's Guide* OCI Programming Advanced Topics, Publish-Subscribe Notification
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ Object > Monitoring Messages
- Java (JDBC): Not available.

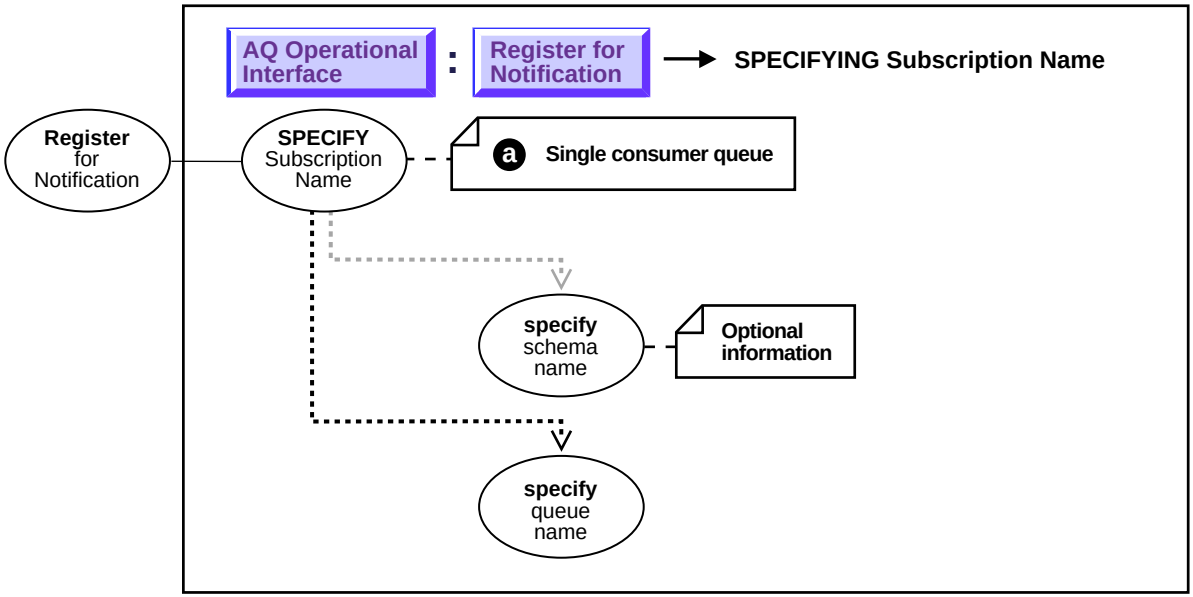
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Examples in the following programmatic environments are provided:

- C (OCI): Register for Notifications For Single-Consumer and Multi-Consumer Queries on page 11-63

Registering for Notification [Specifying Subscription Name — Single-Consumer Queue]

Figure 11–14 Use Case Diagram: Specifying Subscription Name - Single Consumer Queue

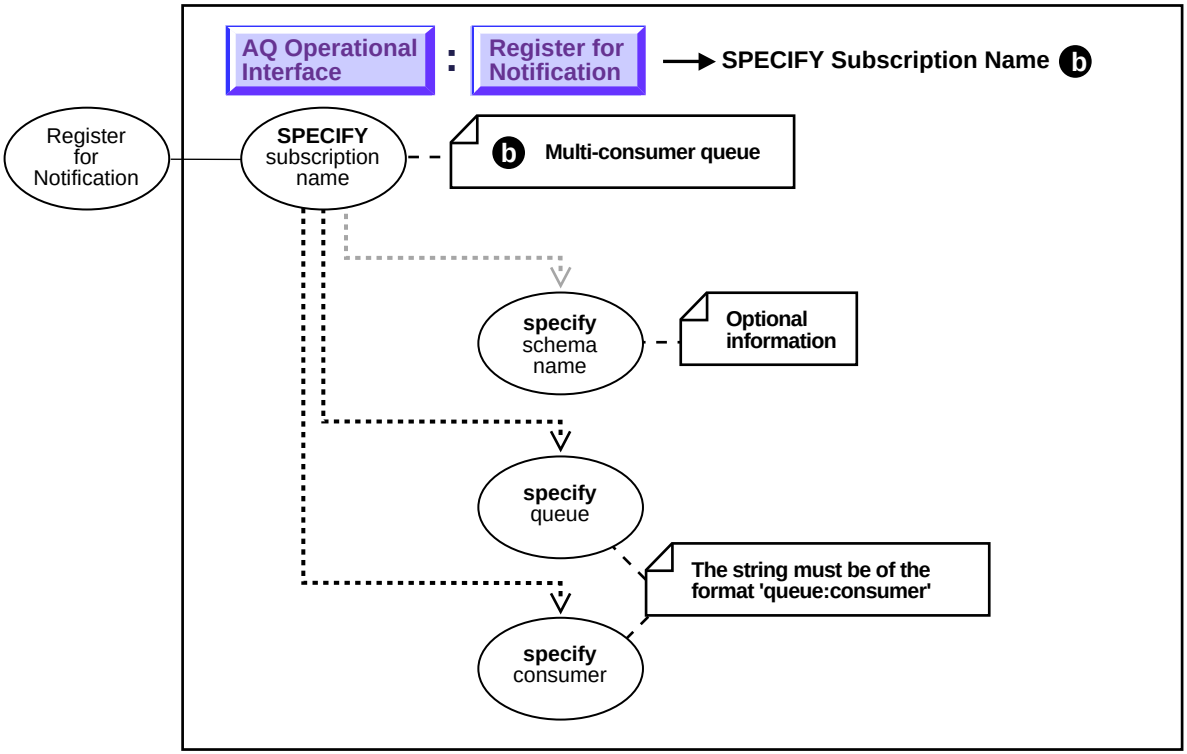


To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Registering for Notification [Specifying Subscription Name — Multi-Consumer Queue]

Figure 11–15 Use Case Diagram: Specifying Subscription Name - Multi-Consumer Queue



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): Not available.
- C (OCI): *Oracle Call Interface Programmer's Guide* OCI Programming Advanced Topics, Publish-Subscribe Notification
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraAQ Object > Monitoring Messages
- Java (JDBC): Not available.

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

C (OCI): Register for Notifications For Single-Consumer and Multi-Consumer Queries

```
/* OCIRegister can be used by the client to register to receive notifications
   when messages are enqueued into non-persistent and normal 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);
```

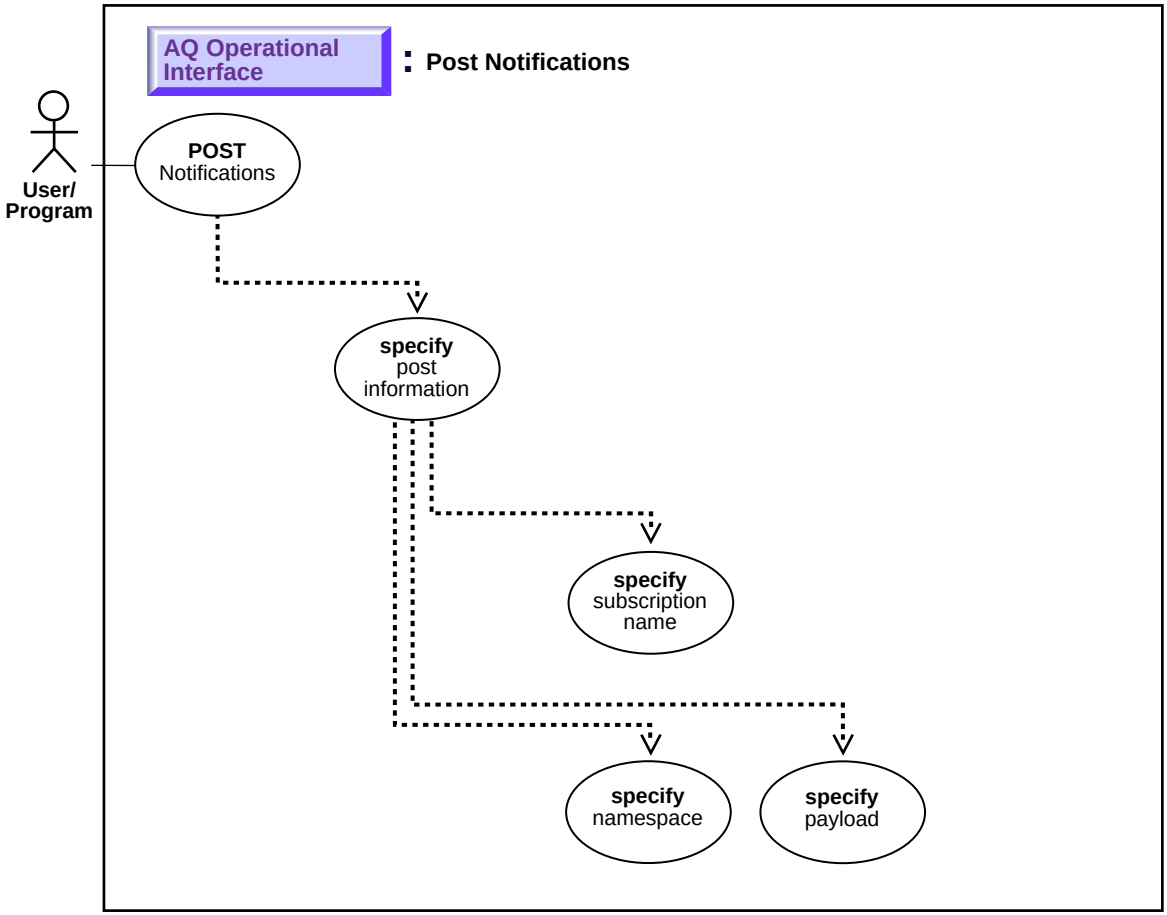
```
 (void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) "scott", (ub4) strlen("scott"),
                (ub4) OCI_ATTR_USERNAME, errhp);

 (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 multi-consumer
```

```
  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 non-persistent
  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 non-persistent 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

Figure 11–16 Use Case Diagram: Posting for Subscriber Notification



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To post to a list of anonymous subscriptions so clients registered for the subscription get notifications.

Usage Notes

Several subscriptions can be posted to at one time. Posting to a subscription involves identifying the subscription name and the payload, if desired. 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, it can use AQ functionality by enqueuing to a queue.

When using OCI, the user 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`.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): `POST` procedure.
- C (OCI): *Oracle Call Interface Programmer's Guide* OCI Programming Advanced Topics, Publish-Subscribe Notification

- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): Not supported.
- Java (JDBC): Not supported.

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

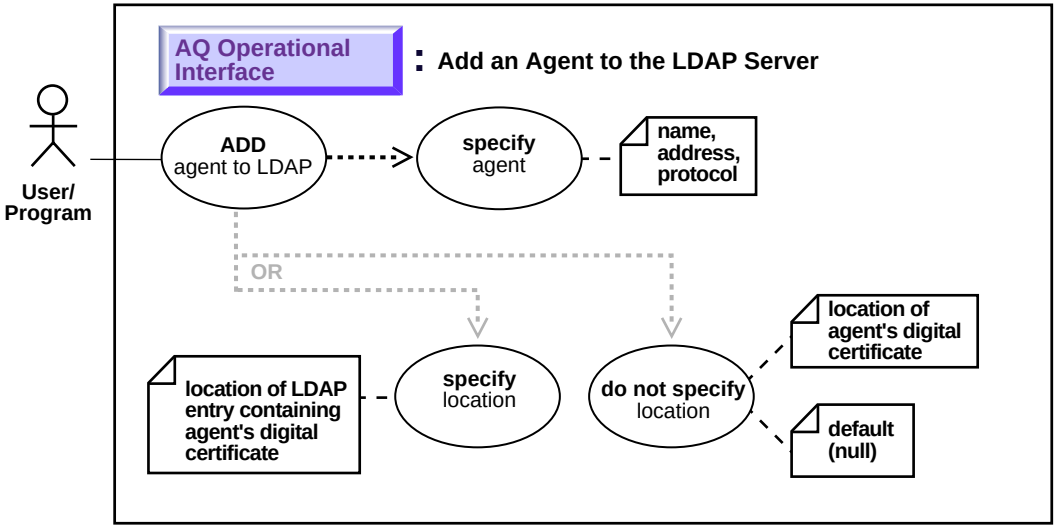
PL/SQL (DBMS_AQ Package): Post of Object-Type Messages

```
-- 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 will be 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

Figure 11–17 Use Case Diagram: Add Agent to LDAP



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To add an agent to the LDAP server.

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 will use. If the agent does not have a digital certificate, this parameter will be defaulted to null.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

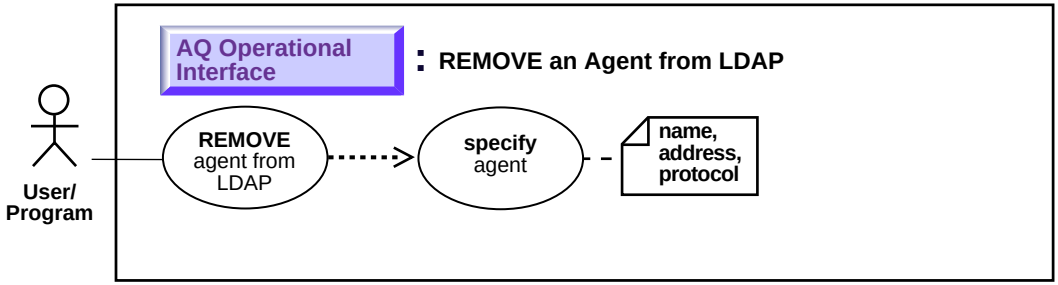
- PL/SQL (DBMS_AQ Package): `BIND_AGENT` procedure.
- C (OCI): *Oracle Call Interface Programmer's Guide* OCI Programming Advanced Topics
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): Not supported.
- Java (JDBC): Not supported.

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

Removing an Agent from the LDAP Server

Figure 11–18 Use Case Diagram: Remove Agent from LDAP



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 11-2

Purpose

To remove an agent from the LDAP server.

Usage notes

This call takes an agent as the argument, and removes the corresponding agent entry in the LDAP server.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_AQ Package): `UNBIND_AGENT` procedure.
- C (OCI): *Oracle Call Interface Programmer's Guide* OCI Programming Advanced Topics

- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): Not supported.
- Java (JDBC): Not supported.

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

12

Creating Applications Using JMS

In Chapter 1, "Introduction to Oracle Advanced Queuing" we described a messaging system for an imaginary company, `BooksOnLine`. In this chapter we consider the features of the Oracle JMS interface to AQ in the context of a sample application based on that scenario. This chapter contains these topics:

- A Sample Application Using JMS
- General Features of JMS
- JMS Point-to-Point Model Features
- JMS Publish-Subscribe Model Features
- JMS Message Producer Features
- JMS Message Consumer Features
- JMS Propagation
- Message Transformation with JMS

A Sample Application Using JMS

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 entire sale process. The front end of the system is an order entry application where new orders are entered. These incoming orders are processed by an order processing application that validates and records the order. Shipping departments located at regional warehouses are then responsible for ensuring that these orders are shipped in a timely fashion. There are three regional warehouses: one serving the East Region, one serving the West Region, and a third warehouse for shipping International orders. Once an order has been shipped, the order information is routed to a central billing department that handles payment processing. The customer service department, located at its own site, is responsible for maintaining order status and handling inquiries about orders.

In Chapter 1 we outlined a messaging system for an imaginary company, `BooksOnLine`. In this chapter we consider the features of the JMS interface to AQ in the context of a sample application based on that scenario. This sample application has been devised for the sole purpose of demonstrating the features of Oracle AQ. Our aim in creating this integrated scenario is to make it easier to grasp the possibilities of this technology by locating our explanations within a single context. However, it is not possible within the scope of a single relatively small code sample to demonstrate every possible application of AQ.

General Features of JMS

- JMS Connection and Session
- JMS Destinations - Queue and Topic
- 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
- Structured Payload / Message Types in JMS

JMS Connection and Session

Connection Factory

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, Oracle8*i* is the JMS Provider.

There are two types of `ConnectionFactory` objects

- `QueueConnectionFactory`
- `TopicConnectionFactory`

ConnectionFactory objects can be obtained in one of the following ways

1. Static methods in AQjmsFactory
2. Java Naming and Directory Interface (JNDI) Lookup from a LDAP directory server

Using AQjmsFactory to Obtain ConnectionFactory Objects

The `AQjmsFactory` class can be used to obtain a handle to `Queue/Topic ConnectionFactory` objects.

- To obtain a `QueueConnectionFactory`, use the `AQjmsFactory.getQueueConnectionFactory()` method

  The queue connection factory can be created using hostname, port number, SID driver or by using JDBC URL and properties.

- To obtain a `TopicConnectionFactory`, use the `AQjmsFactory.getTopicConnectionFactory()` method

  The topic connection factory can be created using hostname, port number, SID driver or by using JDBC URL and properties.

Example

```
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

ConnectionFactory objects can be registered in an LDAP server by a JMS administrator.

The following setup is required to enable JNDI lookup in JMS:

1. When the Oracle9i server is installed, the database must be registered with the LDAP server. This can be done using the Database Configuration Assistant (DBCA).

   AQ entries in the LDAP server have the following structure:

```
                <cn=acme, cn=com>      (administrative context)
                        |
               <cn=OracleContext>     (root of Oracle RDBMS schema)
                        |
                    <cn=db1>            (database)
                        |
 ┌──────────────────────────────────────────────────────────────┐
 │                                │                             │
<cn=OracleDBConnections>          <cn=OracleDBQueue>      <cn=...>

(Connection Factories)             (Queues/Topics)    (Other db objects)
```

   Connection Factory information is stored under <cn=OracleDBConnections>, while topics and queues are stored under <cn=OracleDBQueues>

2. The `GLOBAL_TOPIC_ENABLED` system parameter for the database must be set to `TRUE`. This ensures that all queues and topics created in AQ are automatically registered with the LDAP server.

   This parameter can be set by using

   ```
   ALTER SYSTEM SET GLOBAL_TOPICS_ENABLED = TRUE
   ```

3. After the database has been setup to use an LDAP server, the JMS administrator can register QueueConnectionFactory and TopicConnectionFactory objects in LDAP by using the `AQjmsFactory.registerConnectionFactory()` method.

   The registration can be done 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 Queue/Topic ConnectionFactory, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

- Connect to LDAP via the database server - the user can log on to the Oracle9i 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 directly to LDAP via the database server, the parameters for the registerConnectionFactory method include a JDBC connection (to a user having AQ_ADMINISTRATOR_ROLE), the name of the Queue/Topic ConnectionFactory, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

After the Connection Factory objects have been registered in LDAP by a JMS administrator, they can be looked up by using JNDI

Example

Lets say the JMS administrator wants to register a 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 info
    // 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 info
    // 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");
}
```

Connection

A JMS Connection is a client's active connection to its JMS provider. A 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 `createQueueConnection() or createTopicConnection()` and passing the parameters username and password on the `QueueConnectionFactory` and `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 per 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's `start()` method is called and messages begin arriving at the Connection's consumers. This setup convention minimizes any client confusion that may 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()` - start, or restart, a Connection's delivery of incoming messages.
- `stop()` - Used to temporarily stop a Connection's delivery of incoming messages. When stopped, delivery to all the Connection's message consumers is inhibited. Also, synchronous receive's block and messages are not delivered to message listener
- `close()` - close the JMS session and release all associated resources
- `createQueueSession(true, 0)` - create a queue session. Currently only transacted sessions are allowed.
- `createTopicSession (true, 0)` - create a topic session. Currently only transacted sessions are allowed.
- `setExceptionListener (ExceptionListener)` - set 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()` - get the ExceptionListener for this connection.

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 may allocate provider resources outside the Java virtual machine, it is considered a light-weight JMS object.

A Session serves several purposes:

- Constitutes a factory for its `MessageProducer`s and `MessageConsumers`.
- Provides a way to get a handle to a destination objects (queues / topics)
- Supplies provider-optimized message factories
- Supports a single series of transactions that combines work spanning this session's Producers and Consumers, organizing these into atomic units.
- Defines a serial order for the messages it consumes and the messages it produces.
- Serializes execution of `MessageListener`s registered with it.

Only one session can be created from one connection. Since a provider may 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 `MessageConsumer`s created by a session.

Methods on the Session object include:

- `commit()` - commits all messages performed in this transaction and releases locks currently held
- `rollback()` - rollsback any messages done in the transaction and release locks currently held
- `close()` - closes the session
- `getDBConnection()` - 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 Oracle JMS specific.

The following are some of the extensions to JMS made by Oracle. The `Session` object has to be cast to `AQjmsSession` to use any of the extensions.

- `QueueTables` and Queues, Topics can be created from the Session object
- `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 following code illustrates how some of the above mentioned calls are used.

Example Code

```
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 queue session */
   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 Destinations - Queue and Topic

A `Destination` is an object a client uses to specify the destination where it sends messages, and the source from which it receives messages.

There are two types of destination objects - `Queue` and `Topic`. In AQ, these would map to a `<schema>.<queue>` at a specific database. `Queue` maps to Single consumer queue in AQ and `Topic` maps to multiconsumer queue in AQ.

Destination objects can be obtained in one of the following ways:

1. Using domain specific methods in the JMS `Session`

2. Java Naming and Directory Interface (JNDI) Lookup from a LDAP directory server

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)`- this method can be used to get a handle to a JMS queue
- `AQjmsSession.getTopic(topic_owner, topic_name)`- this method can be used to get a handle to a JMS topic

Example Code

In the `BooksOnline` application, new orders are to be sent to the `neworders_queue` in OE schema. After creating a JMS connection and session, we can get a handle to the queue as follows

```
public Queue get_queue_example(QueueSession jms_session)
{
   QueueSender   sender;
   Queue         queue = null;

   try
   {
       /* get a handle to the OE.oe_new_orders queue */
       queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
   }
   catch (JMSException ex){
      System.out.println("Exception: " + ex); }
   return queue;
}
```

Using JNDI to Look Up Destination Objects

As described in "Connection Factory" on page 12-3, 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 via JNDI using the queue/topic name or one of their aliases.

Example Code

Lets say we have a new orders queue OE.OE_neworders_que stored 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 info
    // 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");

}
```

Methods on the Destination Object include:

- `alter()` - alters a queue or topic
- `schedulePropagation()` - schedules propagation from a source to a destination
- `unschedulePropagation()` - unschedules a previously scheduled propagation
- `enablePropagationSchedule ()` - enable a propagation schedule

- `disablePropagationSchedule ()` - disable a propagation schedule
- `start()` - starts a queue or a topic.The queue can be started for enqueue and / or dequeue. The topic can be started for published and / or subscribe.
- `stop()` - stops a queue or a topic. The queue is stopped for enqueue and / or dequeue. The topic is stopped for publish and / or subscribe.
- `drop()` - drops a queue or a topic

Example Code

```
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).scheduledPropagation(t_sess, "dba", null,
                  null, null, null);
    /*
       some processing done here
    */
    /* Unschedule propagation */
    ((AQjmsDestination)topic).unscheduledPropagation(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* supports system-level access control for all queueing operations. This feature allows an application designer or DBA to create users as queue administrators. A queue/topic administrator can invoke all JMS interface (both administration and operation) on any queue in the database. This simplifies the administrative work since all administrative scripts for the queues in a database can be managed under one schema. For more information, see "Oracle Enterprise Manager Support" in on page 4-8.

Example Scenario and Code

In the BooksOnLine (BOL) 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 any queues in the database.If you decide to create PL/SQL packages in the BOLADM schema that can be used by any applications to enqueue or dequeue, then you should also grant `BOLADM` the `ENQUEUE_ANY` and `DEQUEUE_ANY` system privilege.

```
CREATE USER BOLADM IDENTIFIED BY BOLADM; GRANT CONNECT, RESOURCE, aq_
administrator_role TO BOLADM;
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "BOLADM", false);
((AQjmsSession)t_sess).grantSystemPrivilege("DEQUEUE_ANY", "BOLADM", false)
;where t_sess is the session object.
```

In the application, AQ propagators populate messages from the OE (Order Entry) schema to WS (Western Sales), ES (Eastern Sales) and OS (Worldwide Sales) schemas. The WS, ES and OS schemas in turn populate messages to CB (Customer

Billing) and CS (Customer Service) schemas. Hence the OE, WS, ES and OS 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 need to grant schemas of the source queues enqueue privileges to the destination queues.

To simplify administration, all schemas that host a source queue in the BooksOnLine application are granted the ENQUEUE_ANY system privilege.

```
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "OE", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "WS", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "ES", false);
((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "OS", false);
where t_sess is the session object
```

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 do not need 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* supports queue/topic level access control for enqueue and dequeue operations. This feature allows the application designer to protect queues/topics created in one schema from applications running in other schemas. You need to grant only minimal access privileges to the applications that run outside the queue/topic's schema. The supported access privileges on a queue/topic are `ENQUEUE`, `DEQUEUE` and `ALL`. For more information see "Oracle Enterprise Manager Support" in Chapter 4, "Managing AQ".

Example Scenario and Code

The BooksOnLine application processes customer billings in its `CB` and `CBADM` schemas. CB (Customer Billing) 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_topic`, the shipped order topic. It processes the messages, and then enqueues new messages into `CBADM_billedorders_topic`, the billed order topic.

To protect the queues from other illegal operations from the application, the following two grant calls are made:

```
/* 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. */

((AQjmsDestination)cbadm_shippedorders_topic).grantTopicPrivilege(t_sess,
"DEQUEUE", "CB", false);
where t_sess is the session

/* 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. */

((AQjmsDestination)cbadm_billedorders_topic).grantTopicPrivilege(t_sess,
"ENQUEUE", "CB", false);
```

Retention and Message History in JMS

AQ allows users retain messages in the queue-table. This means that SQL can then be used to query these message 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, the two are related. As the application designer, you may want to keep track of such relationships. Along with retention and message identifiers, 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 Scenario and Code

Let us suppose that the shipping application needs to determine the average processing times of orders. This includes the time the order has to 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 will only analyze 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 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 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, the queue monitor changes ownership to a suitable instance such as the secondary instance.

AQ propagation is able to make use of Real Application Clusters, although it is completely 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 will be handling the propagation from queues stored in that queue table thereby minimizing pinging. Additional discussion on this topic can be found under AQ propagation scheduling (see "Scheduling a Queue Propagation"in Chapter 9, "Administrative Interface"and *Oracle9i Real Application Clusters Installation and Configuration*.)

Example Scenario and Code

In the BooksOnLine example, operations on the `OE_neworders_que` and `booked_order_topic` at the order entry (OE) site can be made faster if the two topics are associated with different instances. This is done by creating the topics in different queue tables and specifying different affinities for the queue tables in the `CreateQueueTable()` command.

In the example, the queue table `OE_orders_sqtab` stores queue `OE_neworders_que` and the primary and secondary are instances 1 and 2 respectively. For queue

table `OE_orders_mqtab` stores queue `booked_order_topic` and the primary and secondary are instances 2 and 1 respectively. The objective is to let instances 1 & 2 manage the two queues in parallel. By default, only one instance is available. In this case the owner instances of both queue tables will be set to instance1. However, if Real Application Clusters are set up correctly and both instances 1 and 2 are available, then queue table OE_orders_sqtab will be owned by instance 1 and the other queue table will be 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 example below. Information about the primary, secondary and owner instance of a queue table can be obtained by querying the view USER_QUEUE_TABLES. See "Selecting Queue Tables in User Schema" in Chapter 10, "Administrative Interface: Views".

```
/* Create queue tables, topics for OE */

/* createing a queue table to hold queues */
qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
qt_prop.setPrimaryInstance(1);
qt_prop.setSecondaryInstance(2);
q_table = createQueueTable("OE", "OE_orders_sqtab", qt_prop);

/* creating a queue table to hold topics */
qt1_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
qt1_prop.setMultiConsumer(TRUE);
qt1_prop.setPrimaryInstance(2);
qt1_prop.setSecondaryInstance(1);
q_table1 = createQueueTable("OE", "OE_orders_mqtab", qt1_prop);

dest_prop = new AQjmsDestinationProperty();
queue = ((AQjmsSession)q_sess).createQueue(q_table. "OE_neworders_que",
                                           dest_prop);

dest_prop1 = new AQjmsDestinationProperty();
topic = ((AQjmsSession)q_sess).createTopic(q_table1, "OE_bookedorders_topic",
                                           dest_prop1);


/* 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 */
q_table.alter("OE_orders_sqtab", 2, 1);
q_table1.alter("OE_orders_mqtabl", 1, 2);
```

Supporting Statistics Views in JMS

Each instance keeps its own 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 AQ statistics information to the instance issuing the query.

Example Scenario and Code

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 Chapter , "Selecting the Number of Messages in Different States for the Whole Database" in Chapter 10, "Administrative Interface: Views".

```
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';
```

Structured Payload/Message Types in JMS

JMS Messages are composed of the following parts:

- Header - All messages support the same set of header fields. Header fields contain values used by both clients and providers to identify and route messages
- Properties - In addition to the standard header fields, there is a facility for adding optional header fields to a message
  - Standard properties - JMS defines some standard properties that are in effect, optional header fields.
  - Provider specific properties - every JMS provider may add certain provider- specific properties to a message
  - Application-specific properties - this provides a mechanism for adding application specific header fields to a message
- Body - this is the message payload. JMS defines various types of message payloads.

Message Header

The message header contains the following fields:

- `JMSDestination` - this field contains the destination to which the message is sent. In AQ this would correspond to the destination queue/topic.
- `JMSDeliveryMode` - JMS supports two modes of message delivery - PERSISTENT (where messages are logged to stable storage) and NON_ PERSISTENT (messages not logged). Oracle AQ support persistent message delivery.
- `JMSMessageID` - this value uniquely identifies a message in a provider. All message ids must begin with ID:.
- `JMSTimeStamp` - contains the time the message was handed over to the provider to be sent. This maps to AQ message enqueue time.
- `JMSCorrelationID` - This field can be used by a client to link one message with another.
- `JMSReplyTo` - this field contains a Destination supplied by a client when a message is sent. Its the destination where a reply to the message should be sent. Clients must use an `AQjmsAgent` to specify the `ReplyTo` Destination.
- `JMSType` - this field contains a message type identifier supplied by a client at send time. For portability it is recommended that the `JMSType` be symbolic values.
- `JMSExpiration` - when a message is sent, its expiration time is calculated as the sum of the time-to-live value specified on the send method and the current GMT. If the time-to-live is specified as zero, expiration is set to zero, which is to say, the message does not expire.
- `JMSPriority` - This field contain's the message's priority. The permitted values for priority are 0, 1, 2, 3..., 0 being the highest priority.JMS permits an administrator to configure JMS to override the client specified values for `JMSDeliveryMode`, `JMSExpiration` and `JMSPriority`.

Message Properties

Properties are a mechanism to add optional header fields to a message. Properties allow a client, via 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 begin with JMSX.

- `JMSXUserID` - The identity of the user sending the message.
- `JMSXAppID` - this is the identity of the application sending the message.
- `JMSXDeliveryCount` - the number of message delivery attempts.
- `JMSXGroupid` - this field is set by the client refers to the identity of the message group, this message is a part of.
- `JMSXGroupSeq` - the sequence number of a message within a group.
- `JMSXRcvTimeStamp` - the time the message was delivered to the consumer (dequeue time)
- `JMSXState` - message state set by provider. Message can be `WAITING`, `READY`, `EXPIRED` or `RETAINED`

Oracle JMS specific properties begin with `JMS_Oracle`. The following properties are Oracle-specific:

- `JMS_OracleExcpQ` - 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` - time in seconds to delay the delivery of the message. This may affect the order if message delivery
- `JMS_OracleOriginalMessageId` - if the messages are propagated from one destination to another, this property is set to the message id of the message in the source. If the message is not propagated, 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 send and receive operations.

Table 12–1

Table 12–2 Message Header Fields

Message Header Field	Type	Set by	Use
`JMSDestination`	`Destination`	Set by JMS after `Send` Method has completed	The destination to which the message is sent
`JMSDeliveryMode`	`int`	Set by JMS after `Send` Method has completed	The delivery mode -`PERSISTENT`
`JMSExpiration`	`long`	Set by JMS after Send Method has completed	The expiration time can be specified for a Message Producer or can be explicitly specified during each send or publish
`JMSPriority`	`int`	Set by JMS after `Send` Method has completed	Message's priority can be specified for a Message Producer or can be explicitly specified during each send or publish
`JMSMessageID`	`String`	Set by JMS after `Send` Method has completed	A value that uniquely identifies each message sent by the provider
`JMSTimeStamp`	`long`	Set by JMS after `Send` Method has completed	The time a message is handed to a provider to be sent
`JMSCorrelationID`	`String`	Set by JMS client	A field that can be used to link one message with another
`JMSReplyTo`	`Destination`	Set by JMS client	A destination set by the client, where a reply to the message should be sent. Should be specified as `AQjsAgent` type
`JMSType`	`String`	Set by JMS client	Message type identifier
`JMSRedelivered`	`boolean`	Set by JMS provider	The message probably was delivered earlier but the client did not acknowledge it at that time

Table 12–3 JMS Defined Message Properties

JMS Defined Message Property	Type	Set by	Use
`JMSXUserID`	`String`	Set by JMS after `Send` Method has completed	The identity of the user sending the message
`JMSAppID`	`String`	Set by JMS after `Send` Method has completed	The identity of the application sending the message
`JMSDeliveryCount`	`int`	Set by JMS after `Receive` Method has completed	The number of message delivery attempts; the first is 1, second is 2,...
`JMSXGroupID`	`String`	Set by JMS client	The identity of the message group the message is a part of
`JMSXGroupSeq`	`int`	Set by JMS client	The sequence number of the message within the group first message is 1, second message is 2...
`JMSXRcvTimeStamp`	`String`	Set by JMS after `Receive` Method has completed	The time that JMS delivered the message to the consumer
`JMSXState`	`int`	Set by JMS Provider	Message state set by provider

Table 12–4 Oracle Defined Message Properties

Header Field/Property	Type	Set by	Use
`JMS_OracleExcpQ`	`String`	Set by JMS Client	Specifies the name of the exception queue
`JMS_OracleDelay`	`int`	Set by JMS Client	Specifies the time (seconds) after which the message should become available to the consumers
`JMS_OracleOrigi-nalMessageID`	`String`	Set by JMS Provider	Specifies the message id of the message in source when the messages are propagated from one destination to another

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 uninterpeted 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 ADT type object (AdtMessage type has been added in Oracle JMS).

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 may also be read or written generically as objects. To use Stream Messages, the user has to create the Queue table with payload type `SYS.AQ$_JMS_STREAM_MESSAGE`.

Stream messages support the following conversion table. A value written as the row type can be read as the column type.

Table 12–5 Stream Message Conversion

	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	

Table 12–5 Stream Message Conversion

	boolean	byte	short	char	int	long	float	double	String	byte[]
double								X	X	
String	X	X	X	X	X	X	X	X	X	
byte[]										X

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 may also be written generically as objects. To use Bytes Messages, the user has to create the Queue table with payloadtype `SYS.AQ$_JMS_BYTES_MESSAGE`.

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 may also be read or written generically as objects.

To use Bytes Messages, the user has to create the Queue table with payloadtype `SYS.AQ$_JMS_MAP_MESSAGE`. Map messages support the following conversion table. A value written as the row type can be read as the column type.

Table 12–6 Map Message Conversion

	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

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, the user has to create the Queue table with payload type `SYS.AQ$_JMS_TEXT_MESSAGE`.

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, one

of the collection classes provided in JDK 1.2 can be used. The objects can be read or written using the methods `getObject()` and `setObject(...)`.To use Object Messages, the user has to create the Queue table with payloadtype `SYS.AQ$_JMS_OBJECT_MESSAGE`.

Example Code

```
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);
    }

}
```

Adt Message

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 adds a body containing a Java object that implements the `CustomDatum` interface. For more information on the `CustomDatum` interface refer to the JDBC Developers guide.

To use Adt Messages, the user has to 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.

Using Message Properties with Different Message Types

- JMS Properties that can be set by client using the `setProperty` call:

- On `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, `MapMessage` -

  `JMSXAppID`

  `JMSXGroupID`

  `JMSXGroupSeq`

  `JMS_OracleExcpQ`

  `JMS_OracleDelay`

- On AdtMessage

  `JMS_OracleExcpQ`

  `JMS_OracleDelay`

- JMS Properties that can be obtained by client using the `getProperty` call
  - On `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, `MapMessage`

    `JMSXuserID`

    `JMSXAppID`

    `JMSXDeliveryCount`

    `JMSXGroupID`

    `JMSXGroupSeq`

    `JMSXRecvTimeStamp`

    `JMSXState`

    `JMS_OracleExcpQ`

    `JMS_OracleDelay`

    `JMS_OracleOriginalMessageID`

  - On AdtMessage

    `JMSXDeliveryCount`

    `JMSXRecvTimeStamp`

    `JMSXState`

    `JMS_OracleExcpQ`

JMS_OracleDelay

- JMS Properties / 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 ADT payloads, use AQ rule syntax for any SQL92 where clause of string that contains

    * corrid
    * priority
    * tab.user_data.<adt_field_name>

Payload Used by JMS Examples

```
/*
 *  BooksOrder - payload for BooksOnline example
 *
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolOrder implements Serializable
```

```
{

  int             orderno;
  String          status;
  String          type;
  String          region;
  BolCustomer     customer;
  String          paymentmethod;
  BolOrderItem[]  itemlist;
  String          ccnumber;
  Date            orderdate;

  public BolOrder(int orderno, BolCustomer customer)
  {
    this.customer   = customer;
    this.orderno    = orderno;
  }

  public int getOrderNo()
  {
    return orderno;
  }

  public String getStatus()
  {
    return status;
  }


  public void setStatus(String new_status)
  {
    status = new_status;
  }


  public String getRegion()
  {
    return region;
  }


  public void setRegion(String region)
  {
    this.region = region;
  }
```

```
public BolCustomer getCustomer()
{
  return customer;
}


public String getPaymentmethod()
{
  return paymentmethod;
}

public void setPaymentmethod(String paymentmethod)
{
  this.paymentmethod = paymentmethod;
}

public BolOrderItem[] getItemList()
{
  return itemlist;
}

public void setItemList(BolOrderItem[] itemlist)
{
  this.itemlist = itemlist;
}

public String getCCnumber()
{
  return ccnumber;
}

public void setCCnumber(String ccnumber)
{
  this.ccnumber = ccnumber;
}

public Date getOrderDate()
{
  return orderdate;
}

public void setOrderDate(Date orderdate)
{
  this.orderdate = orderdate;
```

```
  }

}

/*
 *  BolOrderItem - order item type  for BooksOnline example
 *
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolOrderItem implements Serializable
{

  BolBook     item;
  int         quantity;

  public BolOrderItem(BolBook book, int quantity)
  {
    item          = book;
    this.quantity = quantity;

  }

  public BolBook getItem()
  {
    return item;
  }

  public int getQuantity()
  {
    return quantity;
  }
}


/*
 *  BolBook - book type  for BooksOnline example
 *
 */

import java.lang.*;
import java.io.*;
```

```
import java.util.*;

public class BolBook implements Serializable
{

  String    title;
  String    authors;
  String    isbn;
  float     price;

  public BolBook(String title)
  {
    this.title   = title;
  }

  public BolBook(String title, String authors, String isbn, float price)
  {
    this.title   = title;
    this.authors = authors;
    this.isbn    = isbn;
    this.price   = price;

  }

  public String getISBN()
  {
    return isbn;
  }

  public String getTitle()
  {
    return title;
  }

  public String getAuthors()
  {
    return authors;
  }

  public float getPrice()
  {
    return price;
  }

}
```

```
/*
 *  BolCustomer - customer type  for BooksOnline example
 *
 */

import java.lang.*;
import java.io.*;
import java.util.*;

public class BolCustomer implements Serializable
{

  int          custno;
  String       custid;
  String       name;
  String       street;
  String       city;
  String       state;
  int          zip;
  String       country;

  public BolCustomer(int custno, String name)
  {

    this.custno  = custno;
    this.name    = name;
  }

  public BolCustomer(int custno, String custid, String name, String  street,
           String city, String state, int zip, String country)
  {

    this.custno  = custno;
    this.custid  = custid;
    this.name    = name;
    this.street  = street;
    this.city    = city;
    this.state   = state;
    this.zip     = zip;
    this.country = country;

  }

  public int getCustomerNo()
  {
```

```
    return custno;
  }

  public String getCustomerId()
  {
    return custid;
  }

  public String getName()
  {
    return name;
  }

  public String getStreet()
  {
    return street;
  }

  public String getCity()
  {
    return city;
  }

  public String getState()
  {
    return state;
  }

  public int getZipcode()
  {
    return zip;
  }

  public String getCountry()
  {
    return country;
  }
}
```

JMS Point-to-Point Model Features

- Queues
- Queue Sender

- Queue Receiver
- Queue Browser

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 may 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? Since a locked message cannot be dequeued by a process other than the one that has created the lock, each process will dequeue the first unlocked message at the head of the queue.

Before using a queue, the queue needs to be enabled for enqueue/dequeue using 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.

Queue Sender

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 time-to-live for all messages sent by the `QueueSender`. Alternatively, the client can define these options on a per message basis.

Example Code

In the `BooksOnline` application, new orders are to be sent to the `new_orders_queue`. After creating a JMS connection and session, we create a sender:

```
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);
       sender.send(obj_message);
       jms_session.commit();
    }
    catch (JMSException ex)
    {
      System.out.println("Exception: " + ex);
    }
}
```

Queue Receiver

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 Scenario and Code

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);
    }
}
```

Queue Browser

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` may 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
  ```

Example Scenario and Code

In the BooksOnline application, new orders are put into the new_orders_queue. A client can then browse selected messages.

```
public void browse_rush_orders(QueueSession jms_session)
{
   QueueBrowser    browser;
   Queue           queue;
```

```
    ObjectMessage   obj_message;
    BolOrder        new_order;
    Enumeration     messages;
    String          customer_name;

    try
    {
      /* get a handle to the new_orders queue */
      queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

      /* create a Browser to look at RUSH orders in USA */
      browser = jms_session.createBrowser(queue,
                            "JMSCorrelationID = 'RUSH' and country = 'USA' ");

      for (messages = browser.getEnumeration() ; messages.hasMoreElements() ;)
      {
        obj_message = (ObjectMessage)messages.nextElement();

        new_order = (BolOrder)obj_message.getObject();

        customer_name = new_order.getCustomer().getName();
        System.out.println("Customer " +  customer_name +
            " has placed a RUSH order");
      }

      browser.close();
    }
    catch (Exception ex)
    {
      System.out.println("Exception " + ex);
    }
}
```

JMS Publish-Subscribe Model Features

- Topic
- Durable Subscriber
- Topic Publisher
- Recipient Lists
- TopicReceiver
- Topic Browser

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 do not have 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 and/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 may 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:

- Enable enqueue/dequeue on the Topic using the start call in AQjmsDestination.
- 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.
- Create a set of **Durable Subscribers**. Each subscriber may 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
- Subscribers may 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. For details on propagation, see: Chapter 9, "Administrative Interface".

- Create `TopicPublishers` using the session's `createPublisher` method Messages are published using the `publish` call. Messages may be published to all subscribers to the topic or to a specified subset of recipients on the topic
- Subscribers may receive messages on the topic by using the `receive` method
- Subscribers may also receive messages asynchronously by using **Message Listeners**. The concepts of **Remote Subscribers** and **Propagation** are Oracle extensions to JMS.

Example Scenario

In the BooksOnline application all booked orders are published to the `OE_bookedorders_topic`. Orders for customers in the eastern region are routed to the `ES.ES_bookedorders_topic` and those for the western region are routed to the `WS.WS_bookedorders_topic`. There is also another application that subscribes to the `OE_bookedorders_topic` to track messages for some important customers. Refer to the code examples in the following sections.

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 the *Oracle9i Supplied Java Packages Reference* (`createDurableSubscriber` use case).

Remote subscribers are defined using the `createRemoteSubscriber` call.The remote subscriber may 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`.

In the BooksOnline application there is one local subscriber `SUBS1` and two remote subscribers -

- `West_Shipping` at the remote topic `WS.WS_bookedorders_topic`
- `East_Shipping` at `ES.ES_booked_orders_topic`

Example Code

```
 public void create_booked_orders_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);
    }

  }
```

Topic Publisher

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 time-to-live for all messages sent by the `TopicPublisher`. It can also specify these options on a per message basis.

Example Scenario and Code

In the BooksOnline application, booked orders are published to the `OE.OE_bookedorders_topic`

```
public void book_new_order(TopicSession jms_session, ObjectMessage obj_message)
{
   TopicPublisher  publisher;
   Topic           topic;

   try
   {
     /* get a handle to the booked_orders topic */
     topic = ((AQjmsSession) jms_session).getTopic("OE",
                    "OE_bookedorders_topic");

     publisher = jms_session.createPublisher(topic);

     publisher.publish(topic, obj_message);

     jms_session.commit();
   }
   catch (JMSException ex)
   {
     System.out.println("Exception: " + ex);
   }

}
```

In the BooksOnline application, each shipping region receives messages from the corresponding booked orders topic (`WS_bookedorder_topic` or `ES_bookedorder_topic`). The local subscriber `SUBS1` receives messages from the `OE_booked_orders_topic`.

```
public void get_martins_orders(TopicSession jms_session)
```

```
{
   Topic            topic;
   TopicSubscriber  tsubs;
   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("OE",
                  "OE_bookedorders_topic");

     /* Create local subscriber - to track messages for some customers */
     tsubs = jms_session.createDurableSubscriber(topic, "SUBS1",
                "JMSPriority < 3 AND Customer = 'MARTIN'",
                   false);

     /* process 10 messages */
     for(i=0; i<10; i++)
     {
       /* wait for a message to show up in the topic */
       obj_message = (ObjectMessage)tsubs.receive(10);

       new_order = (BolOrder)obj_message.getObject();

       customer = new_order.getCustomer();
       state    = customer.getState();

       System.out.println("Order: " + i + " for customer " +
           customer.getName());
       jms_session.commit();
     }
   }
   catch (Exception ex)
   {
     System.out.println("Exception " + ex);
   }
}
```

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 Scenario and Code

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 booked_orders 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 QueueReceiver.

Example Scenario and Code

```
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_booked_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 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, clients have to use durable subscribers at the remote site to receive messages.

Topic Browser

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 may 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 topic browsers.

TopicBrowsers also support a purge feature. This allows a client using a topic browser 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 the purpose of a purge, a message is considered seen if it has been returned to the client via a call to the `nextElement()` operation on the `java.lang.Enumeration` for the topic browser. Messages that have not yet been seen by the client will not be discarded during a purge. A purge operation may be performed multiple times on the same topic browser.

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, the effects of the purge operation are also undone.

Example Scenario and Code

In the BooksOnline application, all booked orders are published to the `OE_booked_orders_topic`. A client can then browse selected messages.

```
import oracle.jms.TopicBrowser;
// ...
public void browse_rush_orders(TopicSession jms_session)
{
    TopicBrowser    browser;
    Topic           topic;
    ObjectMessage   obj_message;
    BolOrder        new_order;
    Enumeration     messages;
```

```
    String          customer_name;

    try
    {
        /* get a handle to the OE_booked_orders_topic topic */
        topic = ((AQjmsSession) jms_session).getTopic("OE",
            "OE_booked_orders_topic");

        /* create a Browser to look at RUSH orders */
        browser = jms_session.createBrowser(
            topic, "SUBS1", "JMSCorrelationID = 'RUSH'");

        int count = 0;
        for (messages = browser.getEnumeration() ; messages.hasMoreElements() ;)
        {
            obj_message = (ObjectMessage)messages.nextElement();
            new_order = (BolOrder)obj_message.getObject();

            customer_name = new_order.getCustomer().getName();
            System.out.println("Customer " +  customer_name +
                " has placed a RUSH order");

            ++count;
        }

        /* purge messages seen during this browse if there are too many */
        if (count > 100)
        {
            browser.purgeSeen();
        }

        browser.close();
    }
    catch (Exception ex)
    {
    System.out.println("Exception " + ex);
    }
}
```

JMS Message Producer 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 will be 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" in Chapter 9, "Administrative Interface"). Currently, AQ supports ordering on two of the message attributes:

- Priority
- Enqueue Time.

When combined, they lead to four possible ways of ordering:

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 AQ at message publish/send time. This is also the default ordering.

Priority Ordering of Messages. If priority ordering is chosen, each message will be assigned a priority. Priority can be specified as a message property at publish/send time by the Message Producer. The messages will be received in the order of the priorities assigned.

First-In, First-Out (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 behaves like a priority queue, except if two messages are assigned the same priority, they will be received in the order of their enqueue time.

Enqueue Time Followed by Priority. Messages with the same enqueue time will be received according to their priorities. If the ordering criteria of two message is the same, then the order they are received is indeterminate. However, AQ does ensure that messages send/published in the same session with the same ordering criteria will be received in the order they were sent.

Example Scenario and Code

Using the BooksOnLine application, a customer can request

- FedEx shipping (priority 1)
- Priority air shipping (priority 2)
- Regular ground shipping (priority 3)

Priority can be specified at the Message Producer level using the setPriority call, or during the send or publish call. The latter overrides the former.

The Order Entry application uses a FIFO queue to store new orders. New orders are processed by the order entry application and published to the booked orders topic. The order entry application will retrieve messages from the new orders queue in the order of their enqueue time. It uses a FIFO-priority topic to store booked orders. Booked orders are propagated to the regional booked orders topics. At each region, orders in these regional booked orders topics are processed in the order of the shipping priorities. The following calls create the FIFO-priority topic for the Order Entry application to store booked orders.

```
public static void createPriorityTopic(TopicSession jms_session)
{
    AQQueueTableProperty       qt_prop;
    AQQueueTable               pr_qtable;
    AQjmsDestinationProperty   dest_prop;
    Topic                      bookedorders_topic;

    try
    {

   /* Create a priority queue table for OE */
   qt_prop = new AQQueueTableProperty("SYS.AQ$_JMS_OBJECT_MESSAGE");
   qt_prop.setComment("Order Entry Priority " +
             "MultiConsumer Orders queue table");
   qt_prop.setCompatible("8.1");
   qt_prop.setMultiConsumer(true);

   /* Set a FIFO-priority order */
   qt_prop.setSortOrder("priority, enq_time");

   pr_qtable = ((AQjmsSession)jms_session).createQueueTable("OE",
                     "OE_orders_pr_mqtab", qt_prop);

   /* Create a Queue in this queue table */
   dest_prop = new AQjmsDestinationProperty();

   bookedorders_topic =((AQjmsSession)jms_session).createTopic(pr_qtable,
              "OE_bookedorders_topic", dest_prop);
```

```
   /* Enable enqueue and dequeue on the topic */
   ((AQjmsDestination)bookedorders_topic).start(jms_session, true, true);

    }
    catch (Exception ex)
    {
   System.out.println("Exception: " + ex);
    }
}

/* When an order arrives, the order entry application can use the following
   procedure to publish the order into its booked orders topic. A shipping
   priority is specified for each order: */
public static void order_enqueue(TopicSession jms_session, String book_title,
             int book_qty, int order_num, int cust_no,
             String cust_name, int ship_priority,
             String cust_state, String cust_country,
             String cust_order_type)
{
    BolOrder          order;
    BolCustomer       cust_data;
    BolBook           book_data;
    BolOrderItem[]    item_list;
    Topic             topic;
    ObjectMessage     obj_message;
    TopicPublisher    tpub;

    try
    {
      book_data = new BolBook(book_title);
      cust_data = new BolCustomer(cust_no, cust_name);

      order = new BolOrder(order_num, cust_data);

      item_list = new BolOrderItem[1];
      item_list[0] = new BolOrderItem(book_data, book_qty);

      order.setItemList(item_list);

      /* get a handle to the OE bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("OE",
                     "OE_bookedorders_topic");

      /* Create the topic publisher */
```

```
      tpub = jms_session.createPublisher(topic);

      obj_message = jms_session.createObjectMessage();
      obj_message.setObject(order);

      /* Send message - specify priority */
      tpub.publish(topic, obj_message, DeliveryMode.PERSISTENT,
         ship_priority,0);

      jms_session.commit();
    }
    catch (Exception ex)
    {
      System.out.println("Exception ex: " + ex);
    }
}
```

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 AQ extension to JMS message properties.

Delay processing requires the AQ background process, the queue monitor to be started. Note also that receiving by `msgid` overrides the delay specification.

Example Scenario and Code

In the BooksOnLine application, delay can be used to implement deferred billing. The billing application defines a queue in which shipped orders that are not billed immediately are placed with a delay. For example, a certain class of customer accounts, such as corporate customers, may not be billed for 15 days. The billing application dequeues incoming shipped order messages (from the shipped orders queue) and if the order is for a corporate customer, this order is enqueued into a deferred billing queue with a delay. Delay works similarly for publish, though a scenario has not been provided.

```
public static void defer_billing(QueueSession jms_session,
            BolOrder deferred_order)
{
    Queue             def_bill_q;
```

```
    ObjectMessage     obj_message;
    QueueSender       qsender;

    try
    {
   /* get a handle to the deferred billing queue */
   def_bill_q = ((AQjmsSession)jms_session).getQueue("CBADM",
                   "deferbilling_que");

   /* Create the QueueSender */
   qsender = jms_session.createSender(def_bill_q);

   obj_message = jms_session.createObjectMessage();
   obj_message.setObject(deferred_order);

   /* Set Delay as 15 days
    * Delay is specified in seconds
         */
   obj_message.setIntProperty("JMS_OracleDelay", 15*60*60*24);

   qsender.send(obj_message);

   jms_session.commit();

    }
    catch (Exception ex)
    {
   System.out.println("Exception " + ex);
    }

}
```

Time Specification - Expiration

Producers of messages can specify expiration limits, or Time-to-Live (coded as `TimeToLive`) for messages. This defines the period of time the message is available for a Message Consumer.

Time-to-Live can be specified at send / publish time or using the set `TimeToLive` method of a Message Producer, with the former overriding the latter. Note that the AQ background process, the queue monitor must be running to implement Time-to-Live.

Example Scenario

In the BooksOnLine application, `TimeToLive` 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 on a back order topic. If the shipping policy is that all back orders must be shipped within a week, then messages can be published into the back order topic with an expiration of one week. In this case, any back orders that are not processed within one week are moved to the exception topic 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.

Example Code

```
/* Re-enqueue a back order into a back_order Topic and set a timeToLive of
   7 days;
   All back orders must be processed in 7 days or they are moved to the
   exception queue */
public static void requeue_back_order(TopicSession jms_session,
                  String sale_region, BolOrder back_order)
{
    Topic             back_order_topic;
    ObjectMessage     obj_message;
    TopicPublisher    tpub;
    long              timetolive;

    try
    {
   /* Look up a back order topic based on the region */
   if(sale_region.equals("WEST"))
   {
       back_order_topic = ((AQjmsSession)jms_session).getTopic("WS",
                      "WS_backorders_topic");
   }
   else if(sale_region.equals("EAST"))
   {
       back_order_topic = ((AQjmsSession)jms_session).getTopic("ES",
                      "ES_backorders_topic");
   }
   else
   {
       back_order_topic = ((AQjmsSession)jms_session).getTopic("OS",
                      "OS_backorders_topic");
   }

   obj_message = jms_session.createObjectMessage();
```

```
    obj_message.setObject(back_order);

    tpub = jms_session.createPublisher(null);


    /* Set message expiration to 7 days: */
    timetolive = 7*60*60*24*1000;          // specified in  milliseconds


    /* Publish the message */
    tpub.publish(back_order_topic, obj_message, DeliveryMode.PERSISTENT,
            1, timetolive);


    jms_session.commit();
     }
     catch (Exception ex)
     {
    System.out.println("Exception :" + ex);
     }
}
```

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", Chapter 9, "Administrative Interface"). All messages belonging to a group have to be created in the same transaction and all messages created in one transaction belong to the same group. This feature allows you to segment a complex message into simple messages. This is an 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, it is important to note that the destination topic to which messages have to be propagated must also be enabled for transactional grouping. There are also some restrictions you need to keep in mind if the message grouping property is to be preserved while dequeuing messages from a queue enabled for transactional grouping (see “Dequeue Methods" and "Modes of Dequeuing" for additional information).

Example 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 behavior.

In the example given below, each `send` corresponds to an individual book that is part of an order, and the group / transaction represents a complete order. Only the first message contains customer information. Note that the `OE_neworders_que` is defined in the queue table `OE_orders_sqtab` which has been enabled for transactional grouping.

Example Code

```
public static void createMsgGroupQueueTable(QueueSession jms_session)
{
    AQQueueTableProperty       sqt_prop;
    AQQueueTable               sq_table;
    AQjmsDestinationProperty   dest_prop;
    Queue                      neworders_q;

    try
    {
   /* Create a single-consumer orders queue table
         * with message grouping = TRANSACTIONAL
         */
   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 = ((AQjmsSession)jms_session).createQueueTable("OE",
                  "OE_orders_sqtab", sqt_prop);

   /* Create new orders queue for OE */
   dest_prop = new AQjmsDestinationProperty();
   neworders_q = ((AQjmsSession)jms_session).createQueue(sq_table,
```

```
                                "OE_neworders_que",
                            dest_prop);

    }
    catch (Exception ex)
    {
   System.out.println("Exception: " + ex);
    }
}

/* This method send an order to the specified queue */
public static void enqueue_order(QueueSession jms_session, Queue queue,
                  int order_num, String cust_name, int cust_id,
             int book_qty, String book_title)
{
   QueueSender       sender;
   ObjectMessage     obj_message;
   BolOrder          order;
   BolCustomer       cust_data=null;
   BolBook           book_data;
   BolOrderItem[]    item_list;

   try
   {
     book_data = new BolBook(book_title);

     if(cust_name != null)
     {
       cust_data = new BolCustomer(cust_id, cust_name);
     }

     order = new BolOrder(order_num, cust_data);

     item_list = new BolOrderItem[1];
     item_list[0] = new BolOrderItem(book_data, book_qty);

     order.setItemList(item_list);


     sender = jms_session.createSender(queue);

     obj_message = jms_session.createObjectMessage();

     obj_message.setObject(order);
```

```
      sender.send(obj_message);
    }
    catch (Exception ex)
    {
      System.out.println("Exception ex: " + ex);
    }
}

/* Enqueue groups of orders */
public static void enqueue_order_groups(QueueSession jms_session)
{
  Queue neworders_q;

  try
  {
    neworders_q = ((AQjmsSession)jms_session).getQueue("OE",
                          "OE_neworders_que");

    /* Enqueue first group */
    enqueue_order(jms_session, neworders_q, 1, "John", 1000, 2,
        "John's first book");

    enqueue_order(jms_session, neworders_q, 1, null, 0, 1,
        "John's second book");

    jms_session.commit();

    /* Enqueue second group */
    enqueue_order(jms_session, neworders_q, 2, "Mary", 1001, 1,
        "Mary's first book");

    enqueue_order(jms_session, neworders_q, 2, null, 0, 1,
        "Mary's second book");

    enqueue_order(jms_session, neworders_q, 2, null, 0, 1,
        "Mary's third book");

    jms_session.commit();

    /* Enqueue third group */
    enqueue_order(jms_session, neworders_q, 3, "Scott", 1002, 1,
        "Scott's first book");

    enqueue_order(jms_session, neworders_q, 3, null, 0, 2,
        "Scott's second book");
```

```
    enqueue_order(jms_session, neworders_q, 3, null, 0, 2,
        "Scott's third book");

    jms_session.commit();
  }
  catch (Exception ex)
  {
    System.out.println("Exception ex: " + ex);
  }

}
```

JMS Message Consumer Features

- Receiving Messages
- Message Navigation in Receive
- Modes for Receiving Messages
- Retry With Delay Interval
- Asynchronously Receiving Message Using Message Listener
- 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 an synchronously via 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
- non-blocking

Example Code: Block 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: Block 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 Code: Non-Blocking

```
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 behavior works well for FIFO queues and topics but not for priority ordered queues. If a high priority message arrives for the consumer, this client program will not receive the message until it has cleared the messages that were already there for it.

To provide the consumer a better control in navigating the queue for its messages, the 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 as it allows the consumer to remove the message on the top of the queue.
- `NEXT_MESSAGE` get the message after the established position of the consumer. For example, a `NEXT_MESSAGE` issued after the position is at the fourth message, will get the second message in the queue. This is the default behavior.

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.

Note that the transaction grouping property may be negated if messages are received in the following ways:

- Receive 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, Topic Receiver, 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 message in the current group / transaction have been fetched, it changes the navigation option to next transaction and get 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 Code

```
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);
     }
  }
```

Modes for Receiving Messages

For Point-to-Point Mode

Aside from the normal `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 via the `createBrowser` method from `QueueSession`.

If a message is browsed, it remains available for further processing. Note that after a message has been browsed there is no guarantee that the message will be available to the JMS session again as 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 need to create a `QueueBrowser` with the locked mode using the 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

Refer to the `QueueBrowser` Example in Point-to-Point features

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.

Example Scenario and Code

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) via the `QueueBrowser` and the customer country (message payload) is checked. If the customer country is Mexico or Canada the message be deleted from the queue using the remove with no data (since the payload is already known) mode. Alternatively, the lock on the message is released by the `commit` call. Note that the receive call uses the message identifier obtained from the locked mode `browse`.

```
public void process_international_orders(QueueSession jms_session)
 {
  QueueBrowser    browser;
  Queue           queue;
  ObjectMessage   obj_message;
  BolOrder        new_order;
  Enumeration     messages;
  String          customer_name;
  String          customer_country;
  QueueReceiver   qrec;
  String          msg_sel;

  try
  {
    /* get a handle to the new_orders queue */
    queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

    /* create a Browser to look at RUSH orders */
    browser = ((AQjmsSession)jms_session).createBrowser(queue, null, true);

    for (messages = browser.getEnumeration() ; messages.hasMoreElements() ;)
    {
      obj_message = (ObjectMessage)messages.nextElement();

      new_order = (BolOrder)obj_message.getObject();

      customer_name = new_order.getCustomer().getName();

      customer_country = new_order.getCustomer().getCountry();
```

```
      if (customer_country equals ("Canada") || customer_country equals (
"Mexico"))
      {
        System.out.println("Order for Canada or Mexico");
        msg_sel = "JMSMessageID = '" + obj_message. getJMSMessageID()+ "'";
        qrec = jms_session.createReceiver(queue, msg_sel);
        ((AQjmsQueueReceiver)qrec).receiveNoData();
      }
    }
  }catch (JMSException ex)
   { System.out.println("Exception " + ex);

   }
 }
```

Retry With Delay Interval

Max Retries

If the transaction receiving the message from a queue/topic fails, it is regarded as an unsuccessful attempt to remove the message. AQ records the number of failed attempts to remove the message in the message history.

In addition, it also allows the application to specify at the queue/topic level, the maximum number of retries supported on messages. If the number of failed attempts to remove a message exceed this number, the message is moved to the exception queue and is no longer available to applications.

Retry Delay

If the transaction receiving a message aborted, this could be because of a 'bad' condition, for example, an order that could not be fulfilled because there were insufficient books in stock. Since inventory updates are made every 12 hours, it makes sense to `retry` after that time. If an order was not filled after 4 attempts, this could indicates there is a problem.

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 will remain visible in the queue for dequeue after '`retry_delay`' interval. Until then it will be in the '`WAITING`' state. The AQ background process, the 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 via the alter method on the queue/topic. The default value for MAX_RETRIES is 5.

Example Scenario and Code

If an order cannot be filled because of insufficient inventory, the transaction processing the order is aborted. The `booked_orders` topic is set up with `max_retries` = 4 and `retry_delay` = 12 hours.Thus, if an order is not filled up in two days, 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 Message Using Message Listener

Message Listener for a Message Consumer

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 abort the receipt of the message. The message listener will 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.

Example

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 queue receiver */
    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);
    }
  }
```

Message Listener for All Consumers on a Session

The JMS client can receive messages asynchronously for all the consumers of the session by setting the `MessageListener` at the session.

When a message arrives for any of the message consumers of the session, the `onMessage` method of the message listener is invoked with the message. The message listener can commit or abort the receipt of the message. The message listener will not receive messages if the JMS connection has been stopped. No other mode for receiving messages must be used in the session once the message listener has been set.

Example Scenario and Code

In the customer service component of the BooksOnLine example, messages from different databases arrive at the customer service topics, indicating the state of the order. The customer service application monitors the topics and whenever there is a message about a customer order, it updates the order status in the `order_status_table`. The application uses the session listener to monitor the different topics. Whenever there is a message in any of the topics, the `onMessage` method of the session `MessageListener` is invoked.

```
/* define our message listener class */
   public class CustomerListener implements MessageListener
   {
      TopicSession   the_sess;

      /* constructor */
      CustomerListener(TopicSession  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 update status method
         * NOTE: we are assuming it is defined elsewhere
         * /
        update_status(new_order, new_order.getStatus());

        /* commit the asynchronous receipt of the message */
        the_sess.commit();
     }catch (JMSException ex)
    {
      System.out.println("Exception: " + ex);
    }
  }

}
public void  monitor_status_topics(TopicSession jms_session)
 {
   Topic[]             topic = new Topic[4];
   TopicSubscriber[]   tsubs= new TopicSubscriber[4];

   try
   {
     /* get a handle to the OE_bookedorders_topic */
     topic[0] = ((AQjmsSession)jms_session).getTopic("CS",
                                                "CS_bookedorders_topic");
     tsubs[0] = jms_session.createDurableSubscriber(topic[0], "BOOKED_ORDER");

     topic[1] = ((AQjmsSession)jms_session).getTopic("CS",
                                                "CS_billedorders_topic");
     tsubs[1] = jms_session.createDurableSubscriber(topic[1], "BILLED_ORDER");

     topic[2] = ((AQjmsSession)jms_session).getTopic("CS",
                                                "CS_backdorders_topic");
     tsubs[2] = jms_session.createDurableSubscriber(topic[2], "BACKED_ORDER");

     topic[3] = ((AQjmsSession)jms_session).getTopic("CS",
                                                "CS_shippedorders_topic");
```

```
    tsubs[3] = jms_session.createDurableSubscriber(topic[3], "SHIPPED_ORDER");

    MessageListener  mL = new CustomerListener(jms_session);

    /* set the session's message listener */
    jms_session.setMessageListener(mL);

  }catch(JMSException ex)
   { System.out.println("Exception: " + ex); }
}
```

AQ Exception Handling

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 will 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, the default exception queue is used. If the queue/topic is created in a queue table, say QTAB, the default exception queue will be 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 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 will be 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), it will never expire.

- The message was received successfully. However, because of an error while processing the message, the application aborts the transaction that performed the `receive`. The message is returned to the queue/topic and will be available for any applications that are waiting to receive messages. Since 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, 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 aborts the entire transaction, or if it rolls back to a savepoint that was taken before the receive.

- The client program successfully received a message but terminated before committing the transaction.

Example Scenarios

The section retry with delay interval has an example with `MAX_RETRIES`. In the BooksOnLine application, the business rule for each shipping region is that an order will be placed in a back order queue if the order cannot be filled immediately. The back order application will try to fill the order once a day. If the order cannot be filled within 7 days, it is placed in an exception queue for special processing. We implement this using the Time-to-Live property of messages in conjunction with exception queues.

1. *Create the exception queue* `WS_back_order_exp_que`

```
public void create_excp_que(TopicSession jms_session)
    {
      AQQueueTable     q_table;
      Queue            excpq;

      try {
         /* create the exception queue in the queue table with multiple
          * consumer flag true
          */
          q_table = ((AQjmsSession)jms_session).getQueueTable("WS", "WS_orders_
mqtab");

          AQjmsDestinationProperty dest_prop = new AQjmsDestinationProperty();

          dest_prop.setQueueType(AQjmsDestinationProperty.EXCEPTION_QUEUE);
     excpq = ((AQjmsSession)jms_session).createQueue(q_table,
                              "WS_back_orders_excp_que",
```

```
                        dest_prop);
          /* start the exception queue for receiving (dequeuing) messages only
*/
          ((AQjmsDestination)excpq).start(jms_session, false, true);

          }
      catch (JMSException ex)
      { System.out.println("Exception  " + ex); }
    }
```

2. *Publish message on back orders queue with exception queue set to* WS_back_orders_excp_que

```
 public static void requeue_back_order(TopicSession jms_session,
                 String sale_region, BolOrder back_order)
 {
    Topic             back_order_topic;
    ObjectMessage     obj_message;
    TopicPublisher    tpub;
    long              timetolive;

    try
    {
   back_order_topic = ((AQjmsSession)jms_session).getTopic("WS",
                    "WS_backorders_topic");
   obj_message = jms_session.createObjectMessage();
   obj_message.setObject(back_order);

        /* set exception queue */
        obj_message.setStringProperty("JMS_OracleExcpQ", "WS.WS_back_orders_
excp_que");

   tpub = jms_session.createPublisher(null);

   /* Set message expiration to 7 days: */
   timetolive = 7*60*60*24*1000;          // specified in  milliseconds

    /* Publish the message */
   tpub.publish(back_order_topic, obj_message, DeliveryMode.PERSISTENT,
           1, timetolive);
   jms_session.commit();
    }
    catch (Exception ex)
    {
   System.out.println("Exception :" + ex);
```

```
    }
}
```

3. *Receive expired messages from the exception queue using the point-to-point interface*

```
public BolOrder get_expired_order(QueueSession jms_session)
   {
      Queue            queue;
      QueueReceiver    qrec;
      ObjectMessage    obj_message;
      BolCustomer      customer;
      BolOrder         exp_order = null;

      try
      {
      /* get a handle to the exception queue */
       queue = ((AQjmsSession) jms_session).getQueue("WS", "WS_back_orders_excp_
que");

       qrec = jms_session.createReceiver(queue);

       /* wait for a message to show up in the queue */
       obj_message = (ObjectMessage)qrec.receive();

       exp_order = (BolOrder)obj_message.getObject();

       customer = exp_order.getCustomer();

       System.out.println("Expired Order:  for customer " +
                           customer.getName());

      }
      catch (JMSException ex)
      {
        System.out.println("Exception: " + ex);
      }
      return exp_order;
   }
```

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.

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, AQ will automatically propagate 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 Net8

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 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 dblink `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.

Example 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 dblink 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_booked_orders_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_booked_orders_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_booked_orders_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 via 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 may depend on a number of factors such as network traffic, load at source database, load at destination database, and so on. The schedule therefore has to 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 (see "Scheduling a Queue Propagation", Chapter 9, "Administrative Interface"). The duration or propagation window parameter of a schedule specifies the time frame during which propagation has to take place. If the duration is unspecified then the time frame is an infinite single window. If a window has to be repeated periodically then a finite duration is specified along with a `next_time` function that defines the periodic interval between successive windows.

The latency parameter for a schedule is relevant only when a queue does not have any messages to be propagated. This parameter specifies the time interval within which a queue has to be rechecked for messages. Note that if the latency parameter is to be enforced, then the `job_queue_interval` parameter for the `job_queue_processes` should be less than or equal to the latency parameter. The propagation schedules defined for a queue can be changed or dropped at anytime 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 will take 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 dblinks 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 dblink must also have enqueue privileges to the remote topic.

Example Code

```
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 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 the remote database is unavailable, or 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, 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. At anytime it is possible to check 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.

During a retry if propagation is successful then the number of failures is reset to 0. Propagation has support built in for Real Application Clusters and is completely 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 anytime there is a failure at an instance and the queue table that stores the topic is migrated to a different instance, the propagation job is also automatically migrated to the new instance. This will minimize the pinging between instances and thus offer better performance. Propagation has been designed to handle any number of concurrent schedules.

Note that the number of `job_queue_processes` is limited to a maximum of 36 and some of these may be used to handle non-propagation related jobs.Hence, propagation has built in support for multi-tasking 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, propagation automatically re-distributes the schedules among the processes such that they are loaded uniformly.

Example Scenario

In the BooksOnLine example, the `OE_bookedorders_topic` is busy since 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.

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;
```

Exception Handling During Propagation

When a system errors such as a network failure occurs, AQ will continue to attempt to propagate messages using an exponential back-off algorithm. In some situations that indicate application errors AQ will mark 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

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

Transformations 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 have to 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 Topic Subscriber
- Creating a Remote Subscriber. This will enable propagation of messages between Topics of different formats.

The Message Transformation feature is an AQ extension to the standard JMS interface.

Example Scenario

In the BooksOnLine example, we will consider the order entry and shipping applications. For these examples, we will use topics with ADT type payloads.

Example Code

Lets say that the Order entry topic OE.OE_bookedorders_topic has a payload of type OE.OE_ORDER

```
create or replace TYPE OE_order as OBJECT (
            orderno         NUMBER,
            status          VARCHAR2(30),
            ordertype       VARCHAR2(30),
            orderregion     VARCHAR2(30),
            customer        CUSTOMER_TYP,
            paymentmethod   VARCHAR2(30),
            creditcard#     VARCHAR2(30);
            items           ORDERITEMLIST_VARTYP,
            order_date      DATE,
            total           NUMBER);
```

The Western Shipping topic WS_bookedorders_topic has payload of type WS.WS_ORDER:

```
create or replace TYPE WS_Order AS OBJECT (
       customer_name   VARCHAR2(100),
       address         VARCHAR2(1000),
       city            VARCHAR2(1000),
       state           VARCHAR2(1000),
       country         VARCHAR2(1000),
       zipcode         VARCHAR2(1000),
       orderno         NUMBER,
       status          VARCHAR2(30),
       ordertype       VARCHAR2(30),
       items           ORDERITEMLIST_VARTYP,
       order_date      VARCHAR2(10));
```

The java classes (that implement the CustomDatum interface) can be generated for these types using the Jpublisher utility.

We will define a transformation that defines the mapping from OE.OE_Order to WS.WS_ORDER as:

```
execute dbms_transform.create_transformation(
schema => 'OE',
        name => 'OE2WS',
```

```
from_schema => 'OE,
from_type => 'OE_order',
to_schema => 'WS',
to_type => 'WS_order',
transformation => 'OE_order(source.user_data.customer.name, \
    source.user_data.customer.street, \
    source.user_data.customer.city, \
    source.user_data.customer.state, \
    source.user_data.customer.country, \
    source.user_data.customer.zipcode, \
    source.user_data.customer.country, \
    source.user_data.orderno, \
    source.user_data.status, \
    source.user_data.ordertype, \
    source.user_date.items, \
    TO_CHAR(source.user_date.order_date, 'MM:DD:YYYY'))');
```

Sending Messages to a Destination Using a Transformation

A transformation can be supplied when sending/publishing a message to a queue/topic. The transformation will be applied before putting the message into the queue/topic.

The application can specify a transformation using the `setTransformation` interface in the `AQjmsQueueSender` and `AQjmsTopicPublisher` interfaces.

Example Code

Lets say that the orders that are processed by the order entry application should be published to the 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_booked_orders(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 will be applied to the message before returning it to JMS application.

The transformation can be specified via `setTransformation( )` interface of the `AQjmsQueueReceiver, AQjmsTopicSubscriber` and `AQjmsTopicReceiver`.

Example Code

Lets say the Western Shipping application retrieves messages from the OE_bookedorders_topic. It specifies the transformation 'OE2WS' to retrieve the message as the WS_order ADT.

Lets say that the WSOrder Java class has been generated by Jpublisher to map to the Oracle Object WS.WS_order

```
public AQjmsAdtMessage retrieve_booked_orders(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, it's transformation is set to the specified transformation.

Example Code

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.

Lets say that the WSOrder java class has been generated by Jpublisher to map to the Oracle Object WS.WS_order:

```
public AQjmsAdtMessage retrieve_booked_orders(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

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, AQ will automatically propagate the message to the queue/topic at the remote database specified for the remote subscriber. If a transformation is also specified, AQ will apply the transformation to the message before propagating it to the queue/topic at the remote database.

Example Code

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.

Lets say 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);
      }
}
```

13

JMS Administrative Interface: Basic Operations

In this chapter we describe the administrative interface to Oracle Advanced Queuing in terms of use cases. That is, we discuss each operation (such as "Creating a Queue Table") as a use case by that name. A table listing all the use cases is provided at the head of the chapter (see Use Case Model: JMS Administrative Interface — Basic Operations on page 13-2).

A summary figure, "Use Case Diagram: Administrator's Interface — Basic Operations", locates all the use cases in a single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case in which you are interested, by clicking on the relevant use case title.

Each use case is laid out as follows:

- *Use case figure*. A figure that depicts the use case.
- *Purpose*. The purpose of this use case.
- *Usage Notes*. Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment which illustrate the use case.

Use Case Model: JMS Administrative Interface — Basic Operations

Table 13–1 Use Case Model: JMS Administrative Interface — Basic Operations

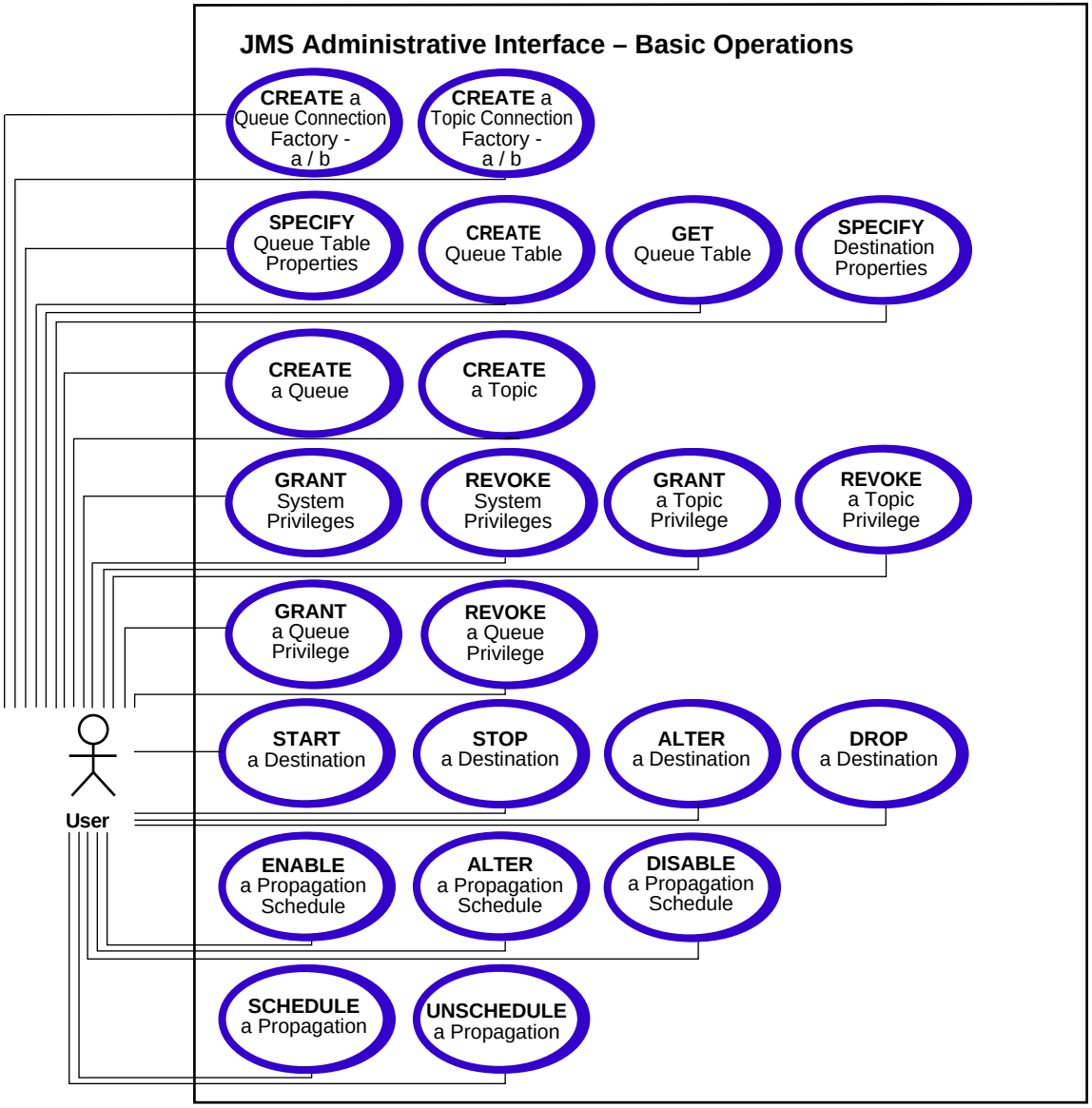
Use Case
Two Ways to Register a Queue/Topic Connection Factory through the Database on page 13-5
Registering through the Database with JDBC Connection Parameters on page 13-6
Registering through the Database with a JDBC URL on page 13-8
Two Ways to Register a Queue/Topic Connection Factory through LDAP on page 13-10
Registering through LDAP with JDBC Connection Parameters on page 13-11
Registering through LDAP with a JDBC URL on page 13-14
Unregister a Queue / Topic Connection Factory in LDAP through the Database on page 13-16
Unregister a Queue / Topic Connection Factory in LDAP through LDAP on page 13-18
Point-to-Point - Two Ways to Create a Queue Connection Factory on page 13-20
Getting a Queue Connection Factory with JDBC URL on page 13-21
Getting a Queue Connection Factory with JDBC Connection Parameters on page 13-23
Publish-Subscribe - Two Ways to Create a Topic Connection Factory on page 13-25
Getting a Topic Connection Factory with JDBC URL on page 13-26
Getting a Topic Connection Factory with JDBC Connection Parameters on page 13-28
Getting a Queue / Topic Connection Factory in LDAP on page 13-30
Getting a Queue / Topic in LDAP on page 13-31
Creating a Queue Table on page 13-32
Creating a Queue Table [Specify Queue Table Property] on page 13-34
Getting a Queue Table on page 13-36
Specifying Destination Properties on page 13-38
Point-to-Point - Creating a Queue on page 13-40
Publish-Subscribe - Creating a Topic on page 13-42
Granting System Privileges on page 13-44
Revoking System Privileges on page 13-46
Publish-Subscribe - Granting Topic Privileges on page 13-48

Table 13–1 (Cont.) Use Case Model: JMS Administrative Interface — Basic Operations

Use Case
Publish-Subscribe - Revoking Topic Privileges on page 13-50
Point-to-Point: Granting Queue Privileges on page 13-52
Point-to-Point: Revoking Queue Privileges on page 13-54
Starting a Destination on page 13-56
Stopping a Destination on page 13-58
Altering a Destination on page 13-60
Dropping a Destination on page 13-62
Scheduling a Propagation on page 13-64
Enabling a Propagation Schedule on page 13-66
Altering a Propagation Schedule on page 13-68
Disabling a Propagation Schedule on page 13-70
Unscheduling a Propagation on page 13-72

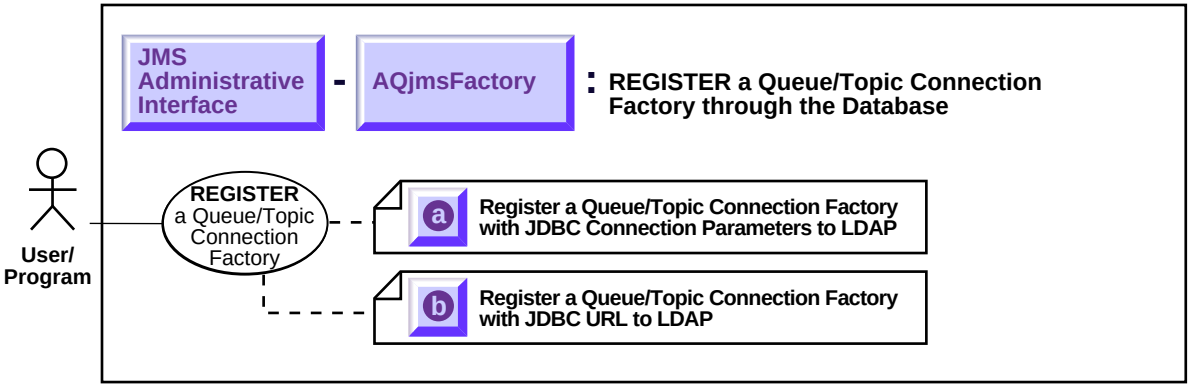
Use Case Model Diagram: JMS Administrative Interface — Basic Operations

Figure 13–1 Use Case Model Diagram: Administrator's Interface — Basic Operations



Two Ways to Register a Queue/Topic Connection Factory through the Database

Figure 13–2 Use Case Diagram: Registering a Queue/Topic Connection Factory through the Database



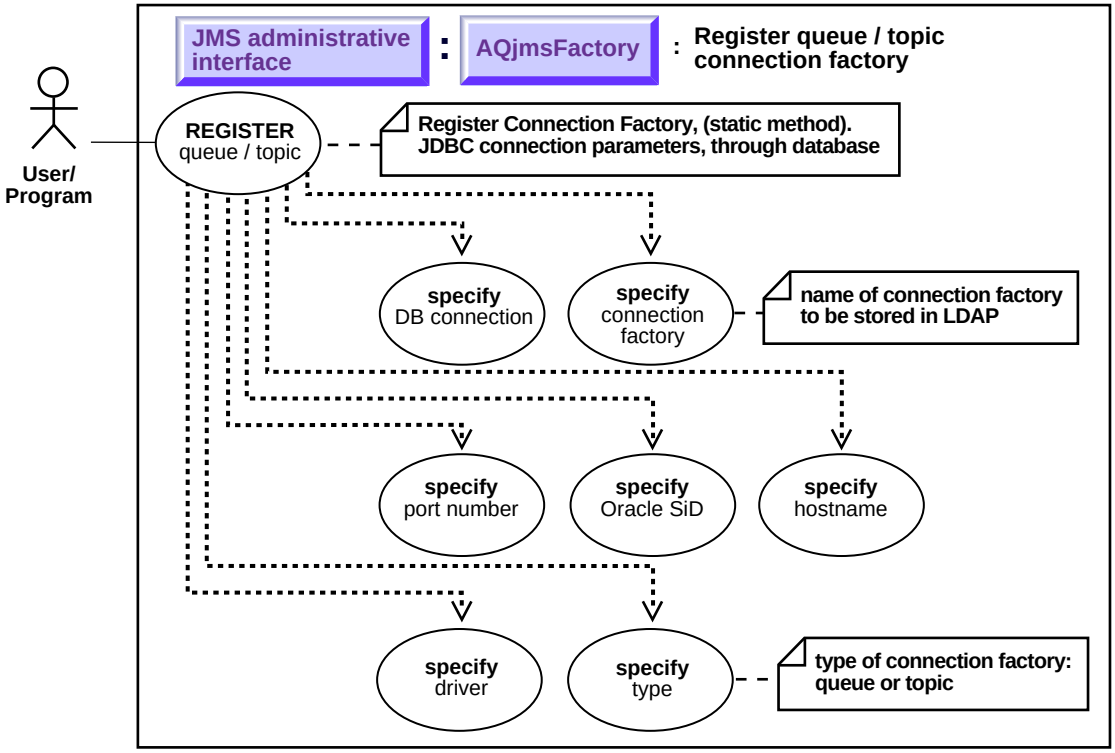
Register a queue/topic connection factory in multiple ways:

a. With JDBC connection parameters to LDAP—see "Registering through the Database with JDBC Connection Parameters" on page 13-6

b. With a JDBC URL to LDAP—see "Registering through the Database with a JDBC URL" on page 13-8

Registering through the Database with JDBC Connection Parameters

Figure 13–3 Use Case Diagram: Register through the Database with JDBC Connection Parameters



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Register a queue / topic connection factory through the database with JDBC connection parameters to LDAP.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the DB connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, look up the connection factory using JNDI.

Syntax

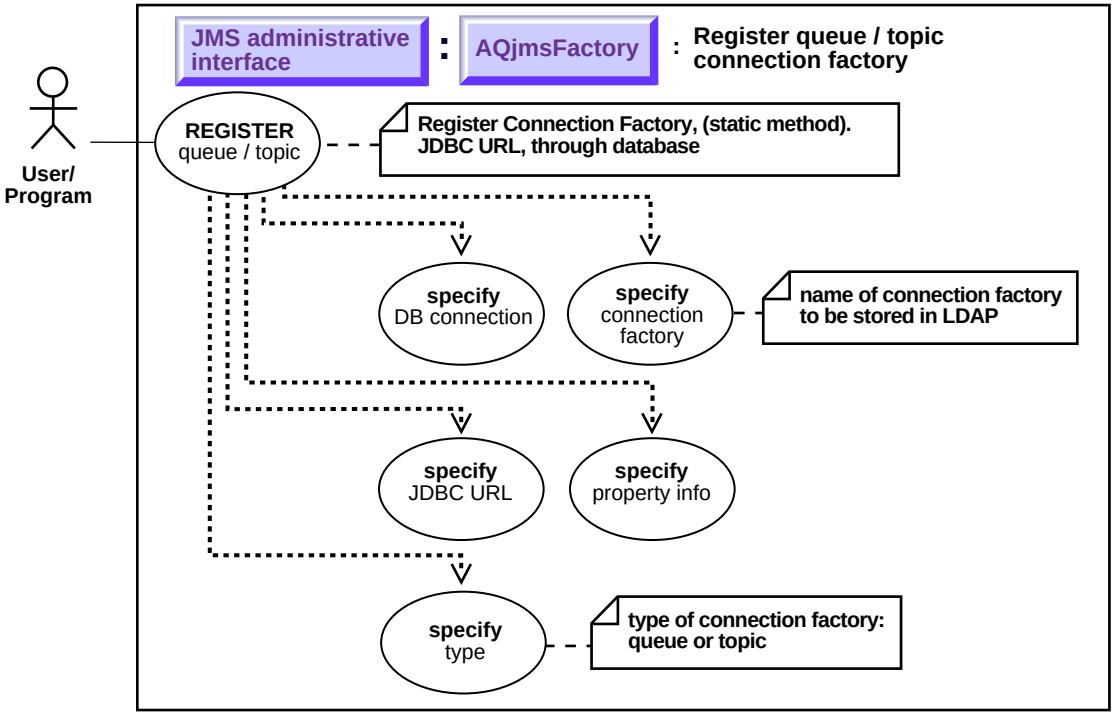
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.registerConnectionFactory.

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 with a JDBC URL

Figure 13–4 Use Case Diagram: Register through the Database with a JDBC URL



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Register a queue / topic connection factory through the database with a JDBC URL to LDAP.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the DB connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, look up the connection factory using JNDI.

Syntax

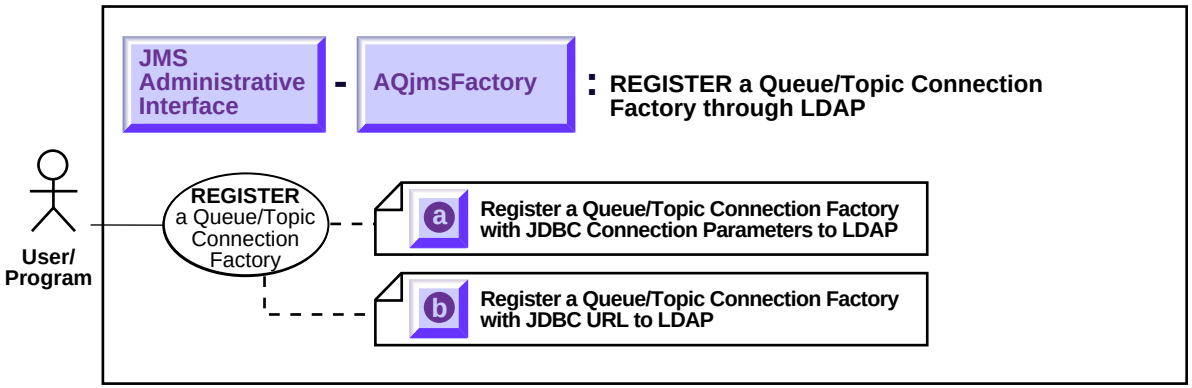
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.registerConnectionFactory.

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");
```

Two Ways to Register a Queue/Topic Connection Factory through LDAP

Figure 13–5 Use Case Diagram: Registering a Queue/Topic Connection Factory through LDAP



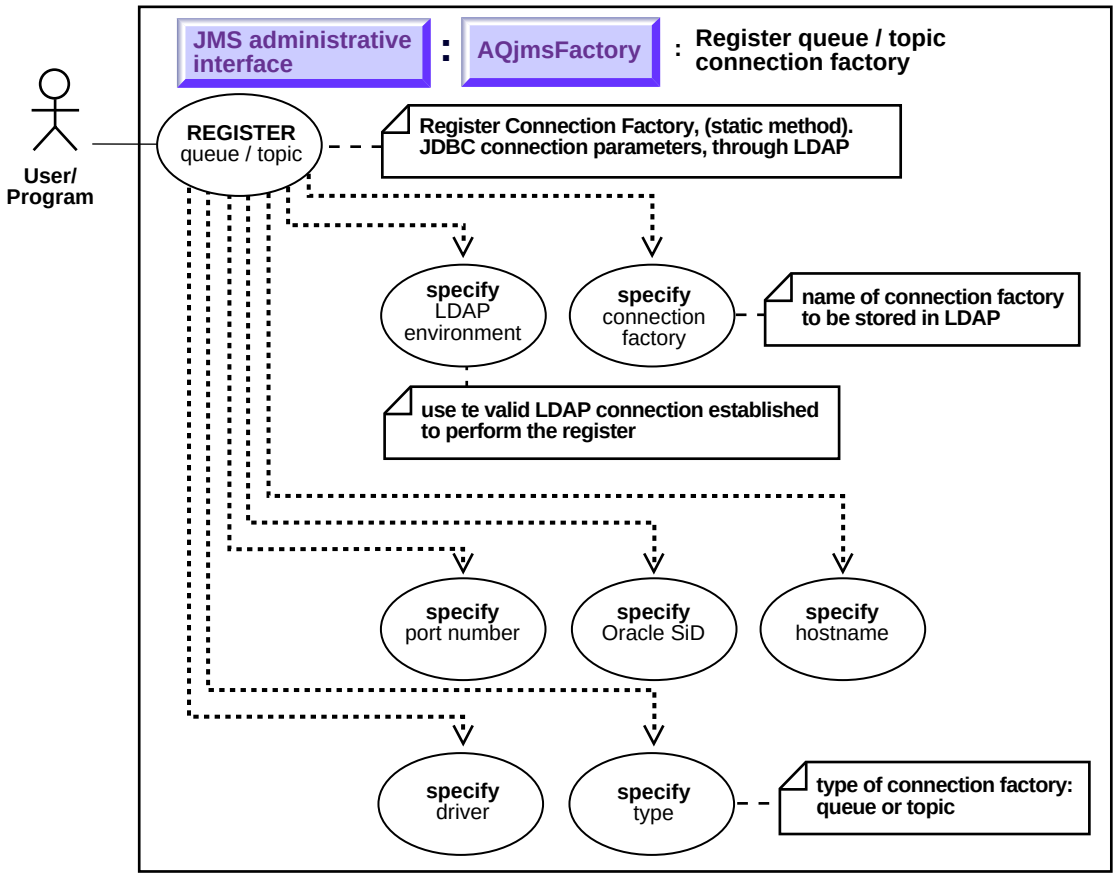
Register a queue / topic connection factory in multiple ways:

a. With JDBC connection parameters to LDAP—see "Registering through LDAP with JDBC Connection Parameters" on page 13-11

b. With a JDBC URL to LDAP—"Registering through LDAP with a JDBC URL" on page 13-14

Registering through LDAP with JDBC Connection Parameters

Figure 13–6 Use Case Diagram: Register through LDAP with JDBC Connection Parameters



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Register a queue/topic connection factory through LDAP with JDBC connection parameters to LDAP.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the hashtable 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.

Syntax

See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.registerConnectionFactory.

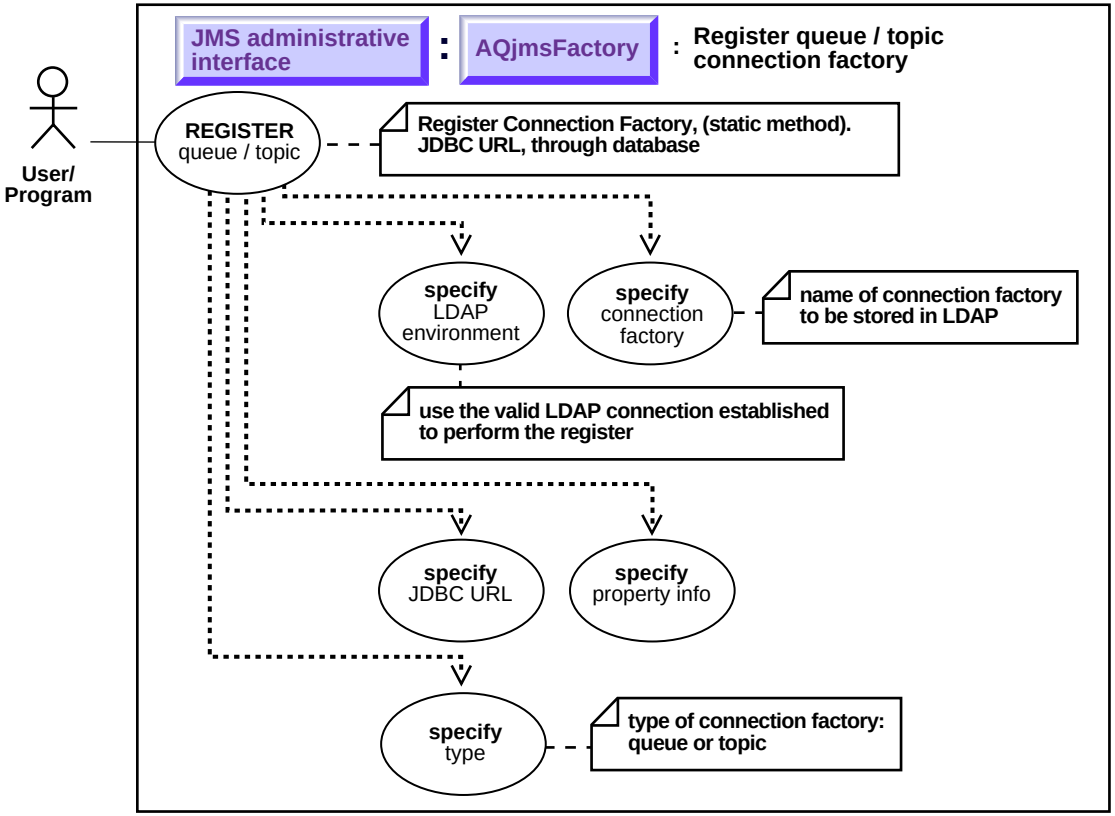
Example

```
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 user name
   * 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 with a JDBC URL

Figure 13–7 Use Case Diagram: Register through LDAP with a JDBC URL



Purpose

Register a queue/topic connection factory through LDAP with JDBC connection parameters to LDAP.

Usage Notes

`registerConnectionFactory` is a static method. To successfully register the connection factory, the hashtable 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.

Syntax

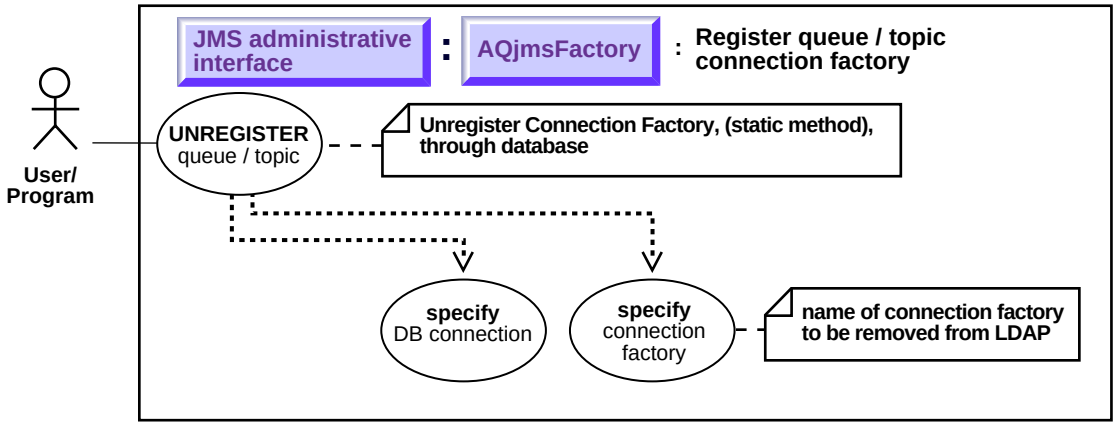
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.registerConnectionFactory.

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 user name
   * 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");
```

Unregister a Queue/Topic Connection Factory in LDAP through the Database

Figure 13–8 Use Case Diagram: Unregister a Queue/Topic Connection Factory in LDAP through the Database



Purpose

Unregister a queue/topic connection factory in LDAP.

Usage Notes

`unregisterConnectionFactory` is a static method. To successfully unregister the connection factory, the DB connection passed to `unregisterConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`.

Syntax

See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.unregisterConnectionFactory.

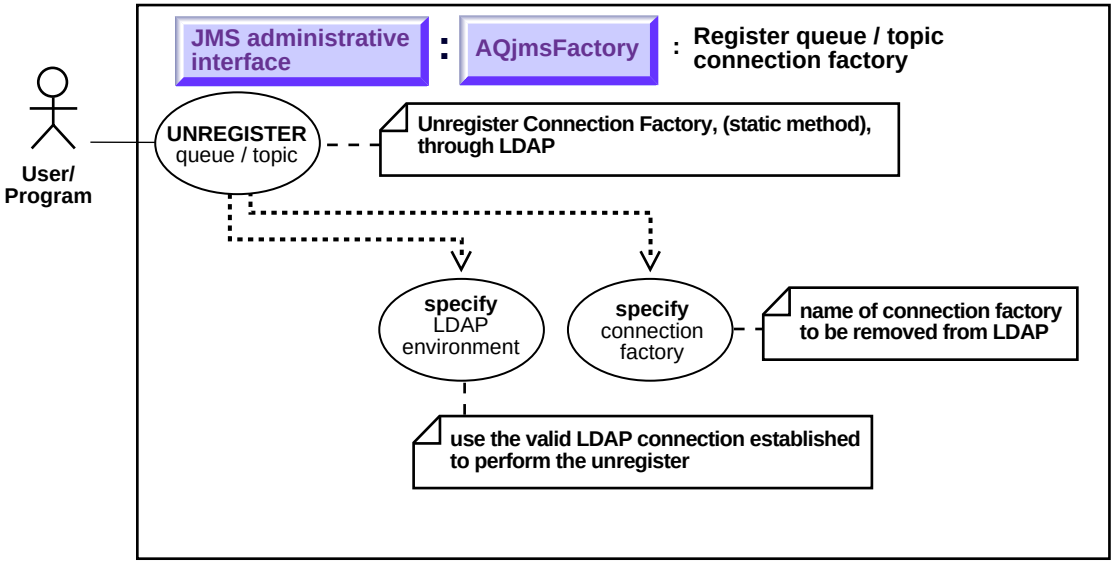
Example

```
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");
```

Unregister a Queue/Topic Connection Factory in LDAP through LDAP

Figure 13–9 Use Case Diagram: Unregister a Queue/Topic Connection Factory in LDAP through LDAP



Purpose

Register a queue / topic connection factory in LDAP.

Usage Notes

`unregisterConnectionFactory` is a static method. To successfully unregister the connection factory, the hashtable 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`).

Syntax

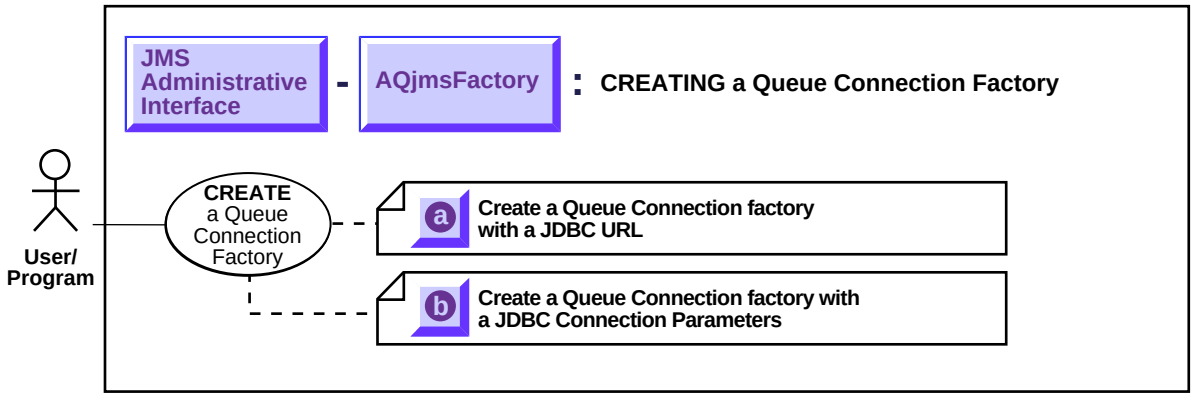
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.unregisterConnectionFactory.

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 user name
   * 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");
```

Point-to-Point - Two Ways to Create a Queue Connection Factory

Figure 13–10 Use Case Diagram: Point-to-Point - Two Ways to Create a Queue Connection Factory



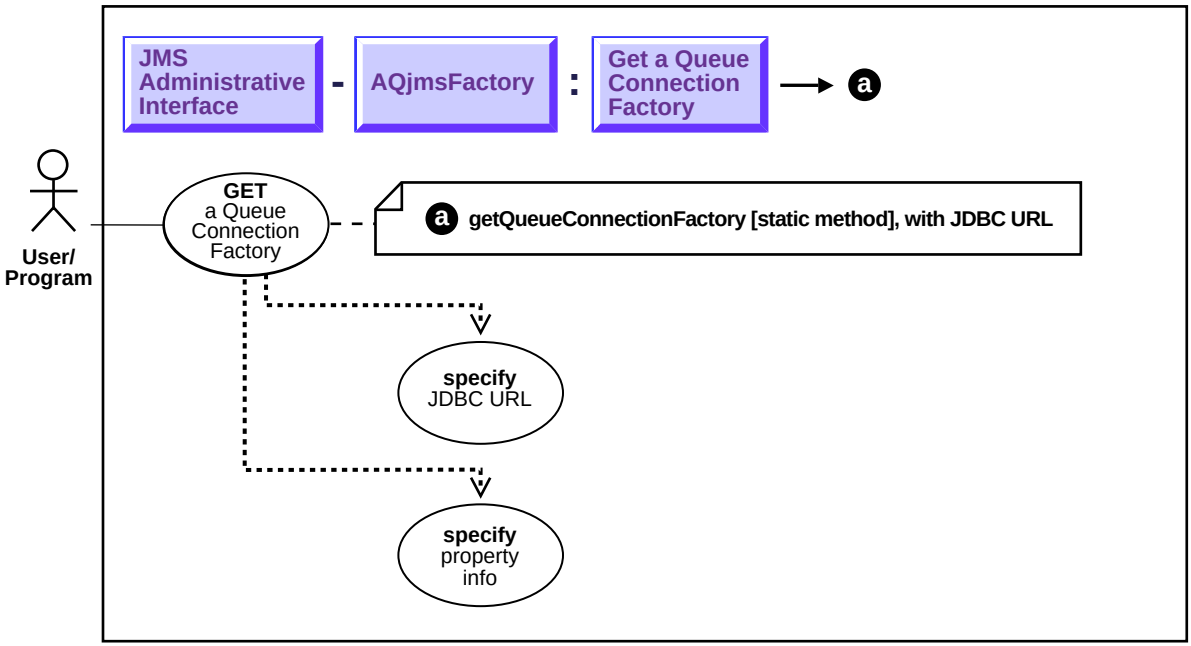
It is possible to Create a queue connection factory two ways.

a. With a JDBC URL — see Getting a Queue Connection Factory with JDBC URL on page 13-21

a. With a JDBC connection parameters — see Getting a Queue Connection Factory with JDBC Connection Parameters on page 13-23

Getting a Queue Connection Factory with JDBC URL

Figure 13–11 Use Case Diagram: Getting a Queue Connection Factory with JDBC Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get a Queue Connection Factory with JDBC URL

Usage Notes

getQueueConnectionFactory is a static method.

Syntax

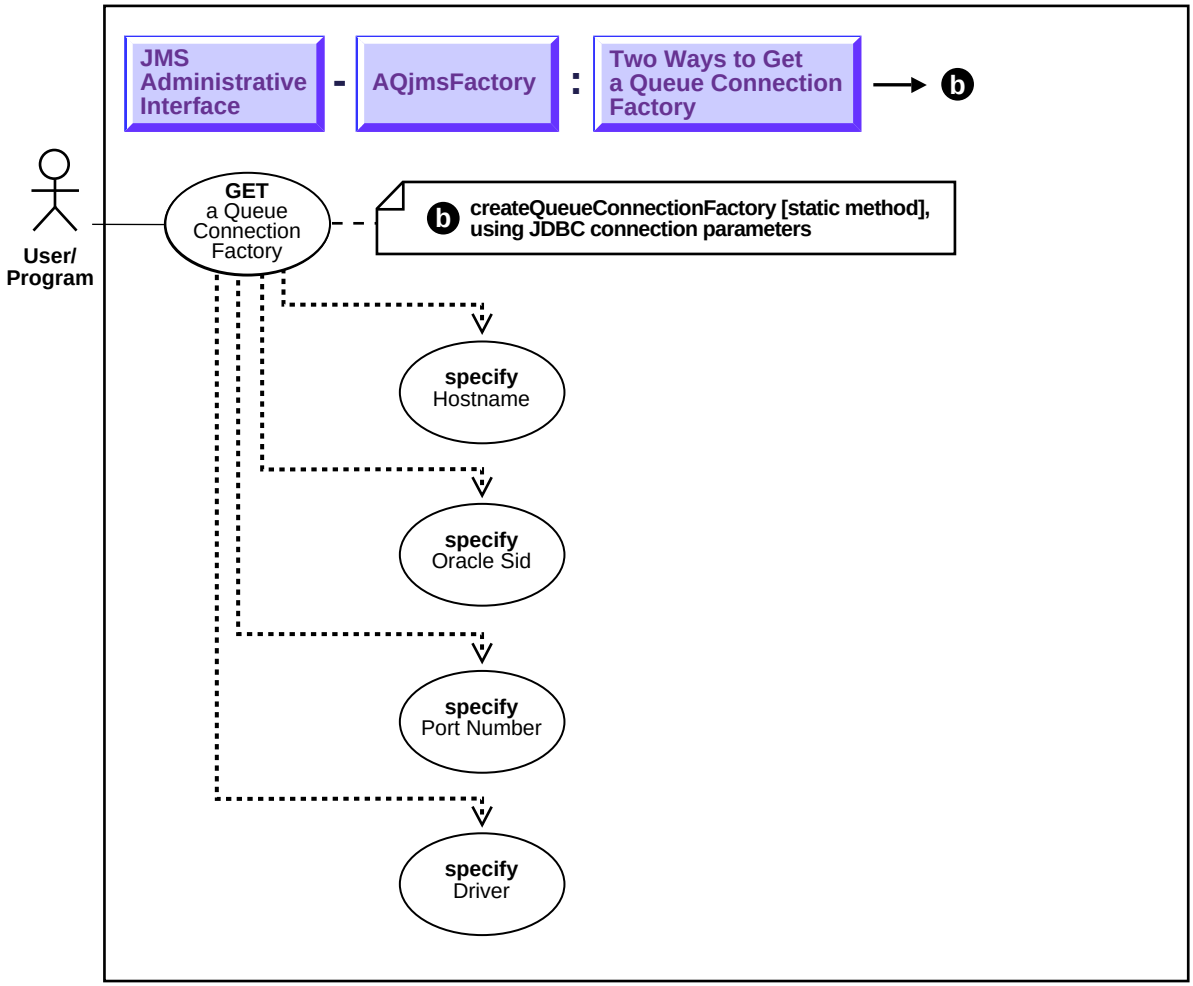
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.getQueueConnectionFactory

Example

```
String      url          = "jdbc:oracle:oci8: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

Figure 13–12 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Topic Connection Factory



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get a Queue Connection Factory with JDBC Connection Parameters

Usage Notes

getQueueConnectionFactory is a static method.

Syntax

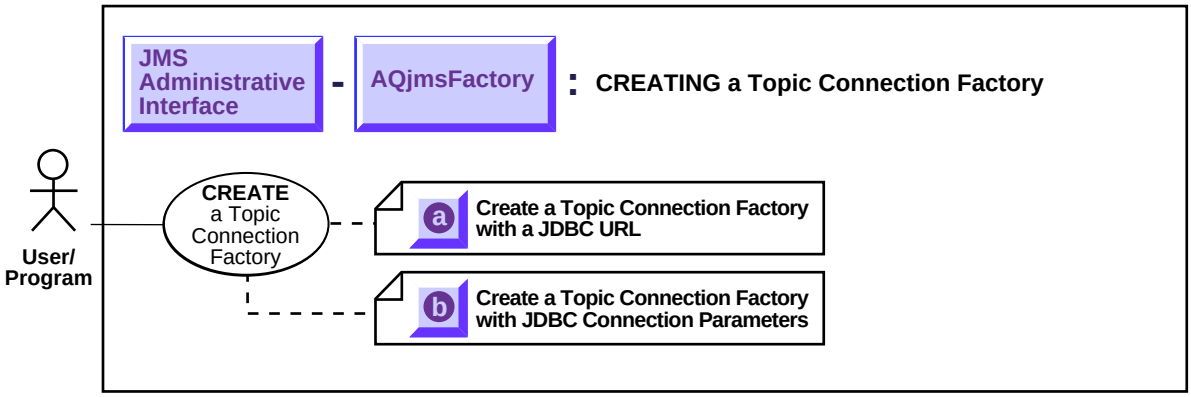
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.getQueueConnectionFactory

Example

```
String      host         = "dlsun";
String      ora_sid      = "rdbms8i"
String      driver       = "thin";
int         port         = 5521;
QueueConnectionFactory   qc_fact;

qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid, port, driver);
```

Publish-Subscribe - Two Ways to Create a Topic Connection Factory

Figure 13–13 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Topic Connection Factory



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

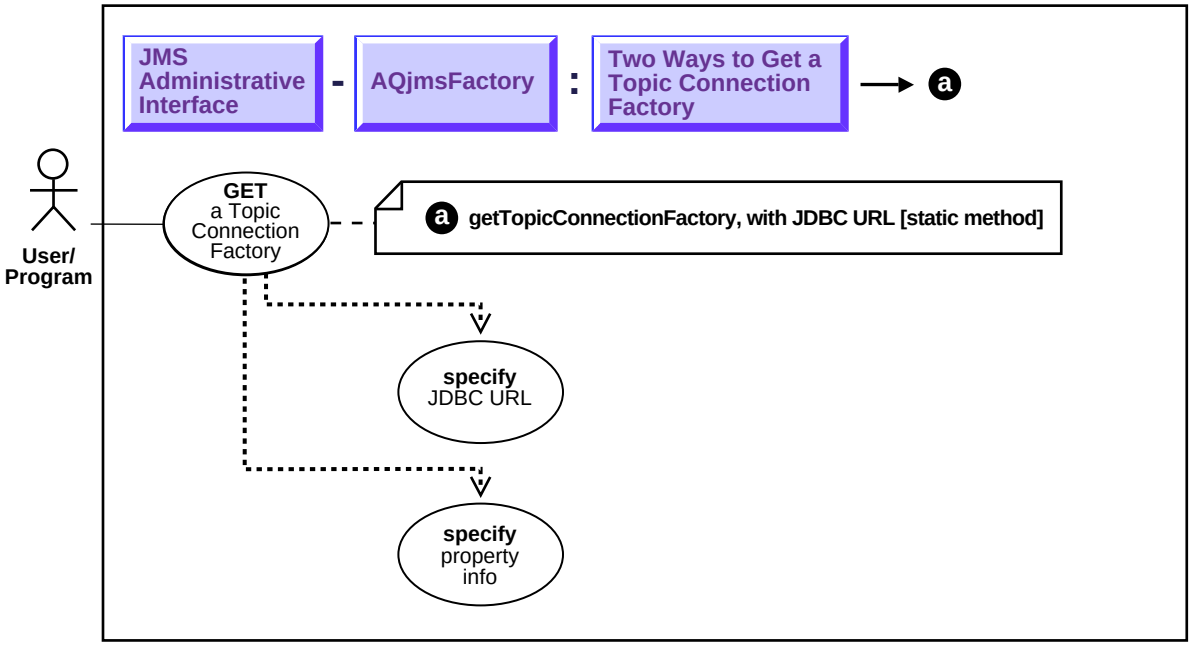
It is possible to create a topic connection factory two ways.

a. With a JDBC URL — see Getting a Topic Connection Factory with JDBC URL on page 13-26

b. With a JDBC connection parameters — see Getting a Topic Connection Factory with JDBC URL on page 13-28

Getting a Topic Connection Factory with JDBC URL

Figure 13–14 Use Case Diagram: Getting a Topic Connection Factory with JDBC URL



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get a Topic Connection Factory with a JDBC URL.

Usage Notes

getTopicConnectionFactory is a static method.

Syntax

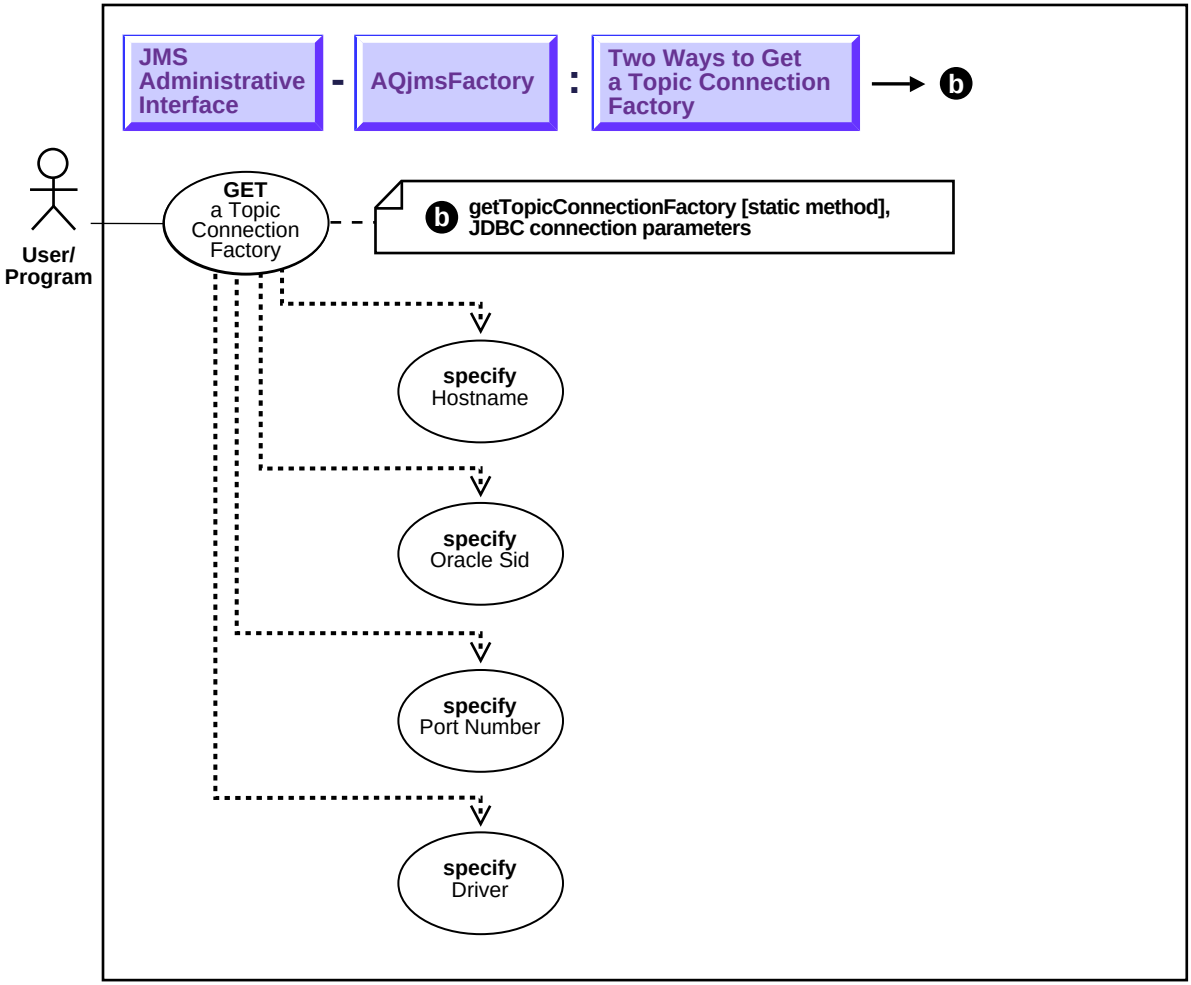
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.getTopicConnectionFactory

Example

```
String      url          = "jdbc:oracle:oci8: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

Figure 13–15 Use Case Diagram: Getting a Topic Connection Factory with JDBC Connection Parameters



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Usage Note

getTopicConnectionFactory is a Static Method.

Purpose

Get a topic connection factory with JDBC connection parameters.

Syntax

See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsFactory.getTopicConnectionFactory

Example

```
String      host         = "dlsun";
String      ora_sid      = "rdbms8i"
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

Get 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 user name
   * 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

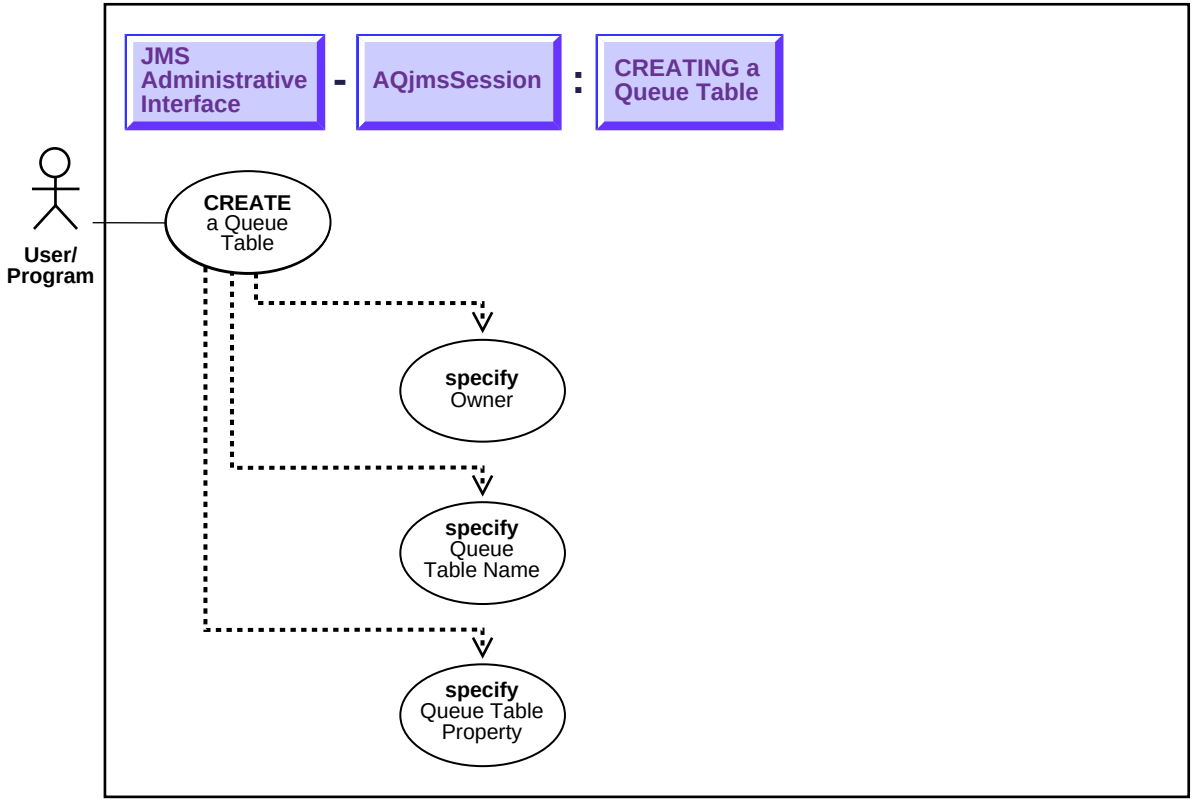
Get a queue/topic from LDAP.

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 user name
   * 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

Figure 13–16 Use Case Diagram: Create Queue Table



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue table.

Usage Notes

CLOB, BLOB, BFILE objects are valid attributes for an AQ object type load. However, only CLOB and BLOB can be propagated using AQ propagation in Oracle8i.

Syntax

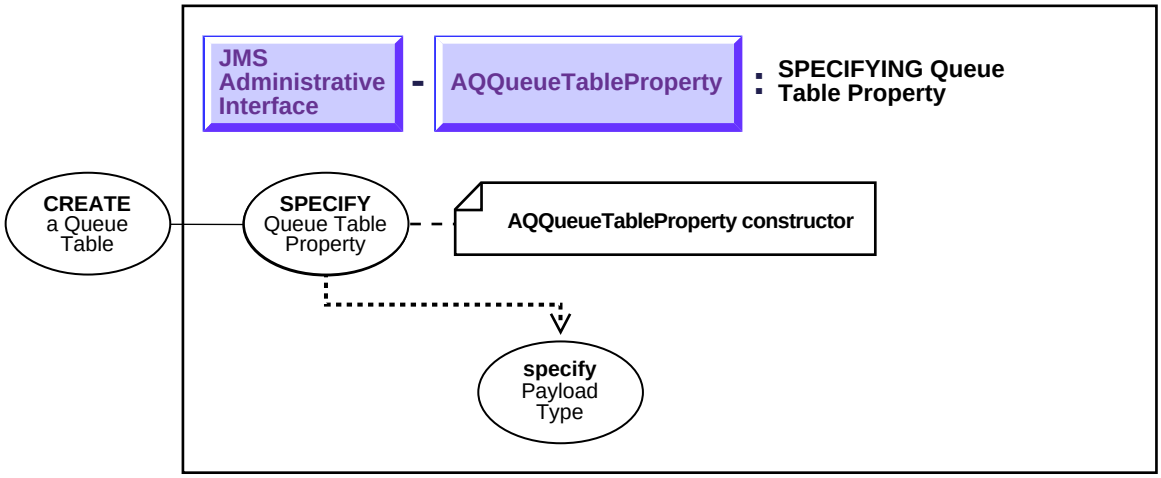
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createQueueTable

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);
```

Creating a Queue Table [Specify Queue Table Property]

Figure 13–17 Use Case Diagram: Specify Queue Table Property



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify queue table properties

Usage Notes

Not applicable.

Syntax

See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.AQ, AQQueueTableProperty

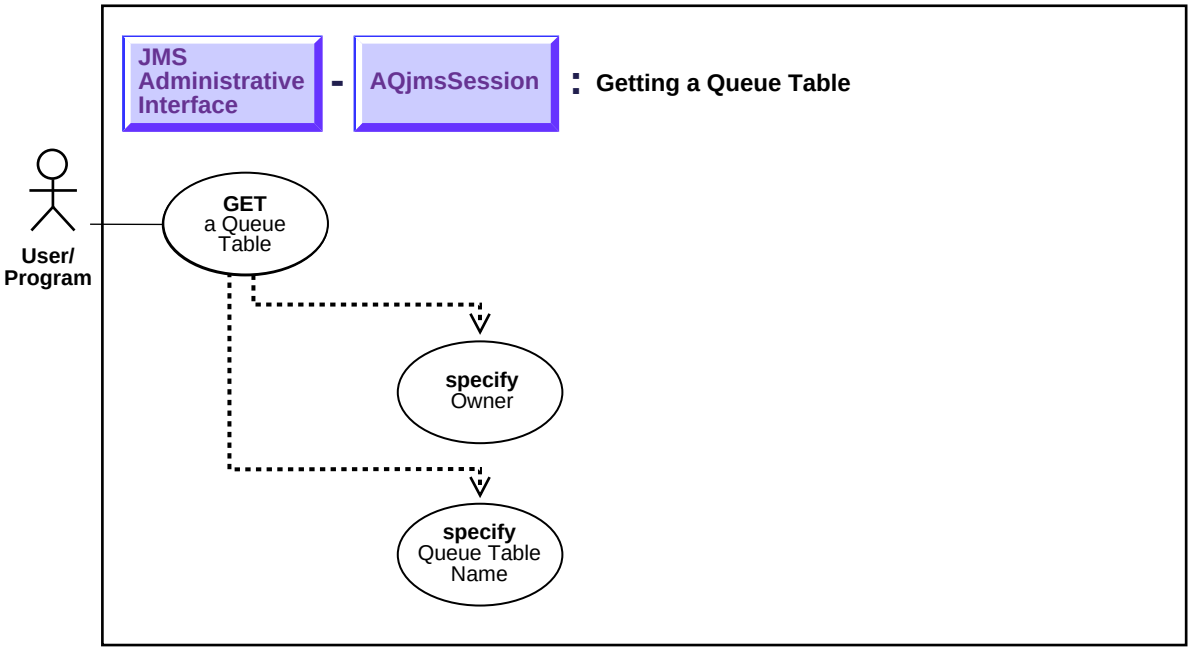
Example

```
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

Figure 13–18 Use Case Diagram: Get Queue Table



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get a queue table.

Usage Notes

If the caller that opened the connection is not the owner of the queue table, the caller must have AQ enqueue/dequeue privileges on queues/topics in the queue table. Otherwise the queue-table will not be returned.

Syntax

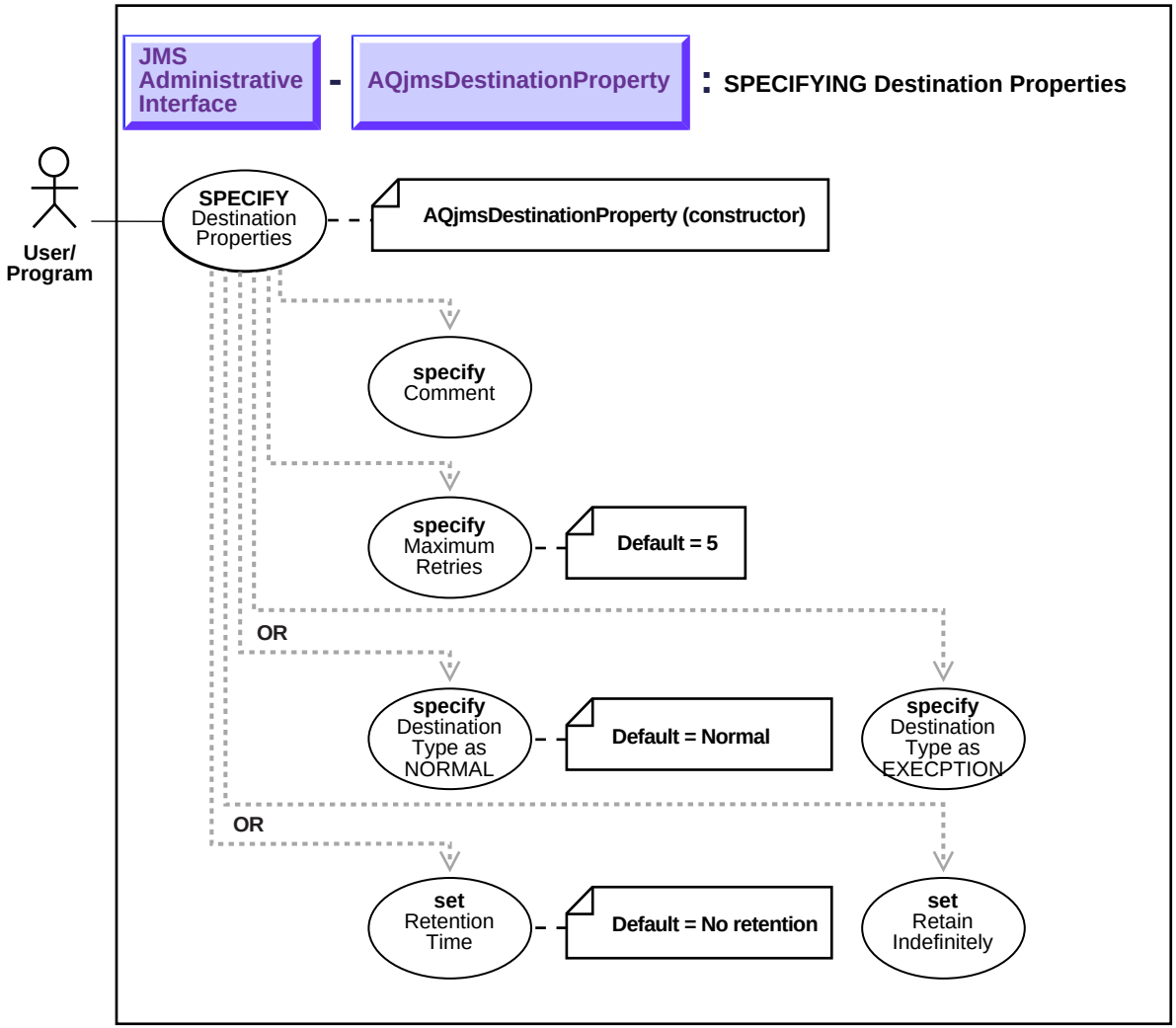
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.getQueueTable

Example

```
QueueSession            q_sess;
AQQueueTable            q_table;

q_table = ((AQjmsSession)q_sess).getQueueTable("boluser",
"bol_ship_queue_table");
```

Specifying Destination Properties

Figure 13–19 Use Case Diagram: Specify Destination Properties



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify destination properties.

Usage Notes

Not applicable.

Syntax

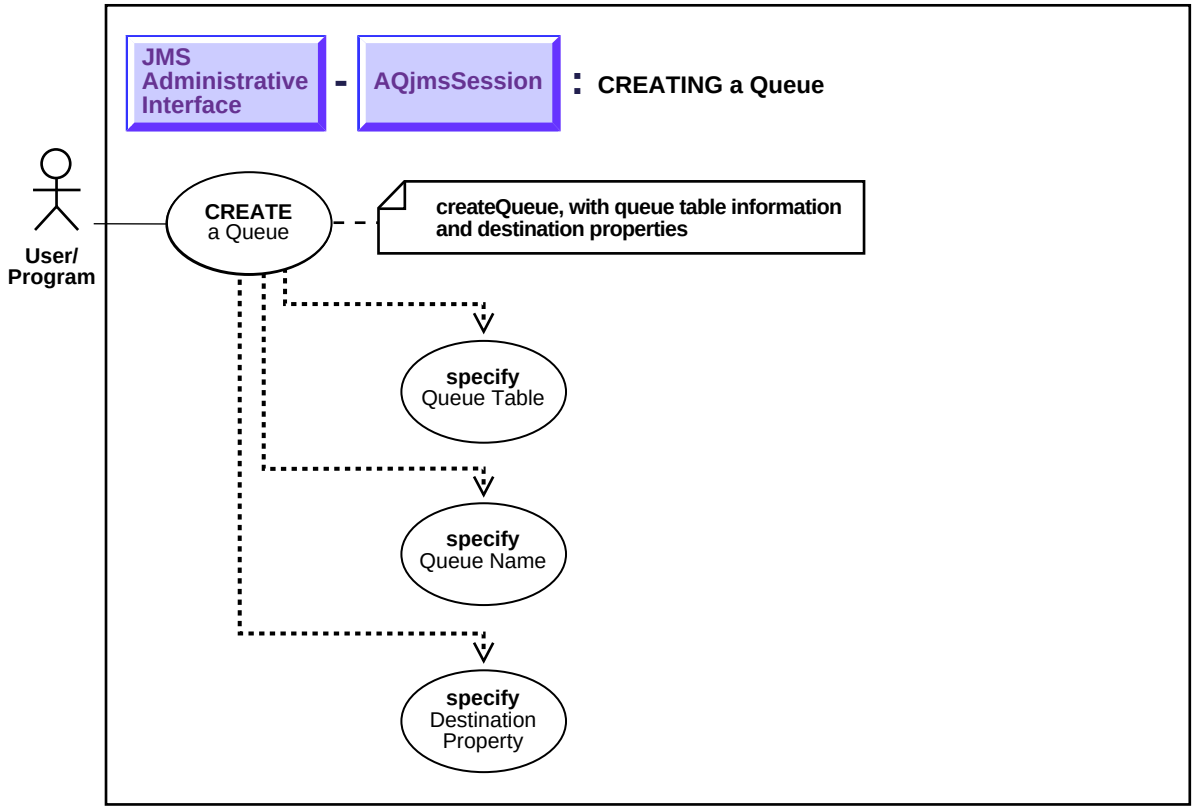
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestinationProperty

Example

No example is provided with this release.

Point-to-Point - Creating a Queue

Figure 13–20 Use Case Diagram: Point-to-Point - Create a Queue



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue in a specified queue table.

Usage Notes

The queue table in which a queue is created has to be a single-consumer queue table.

Syntax

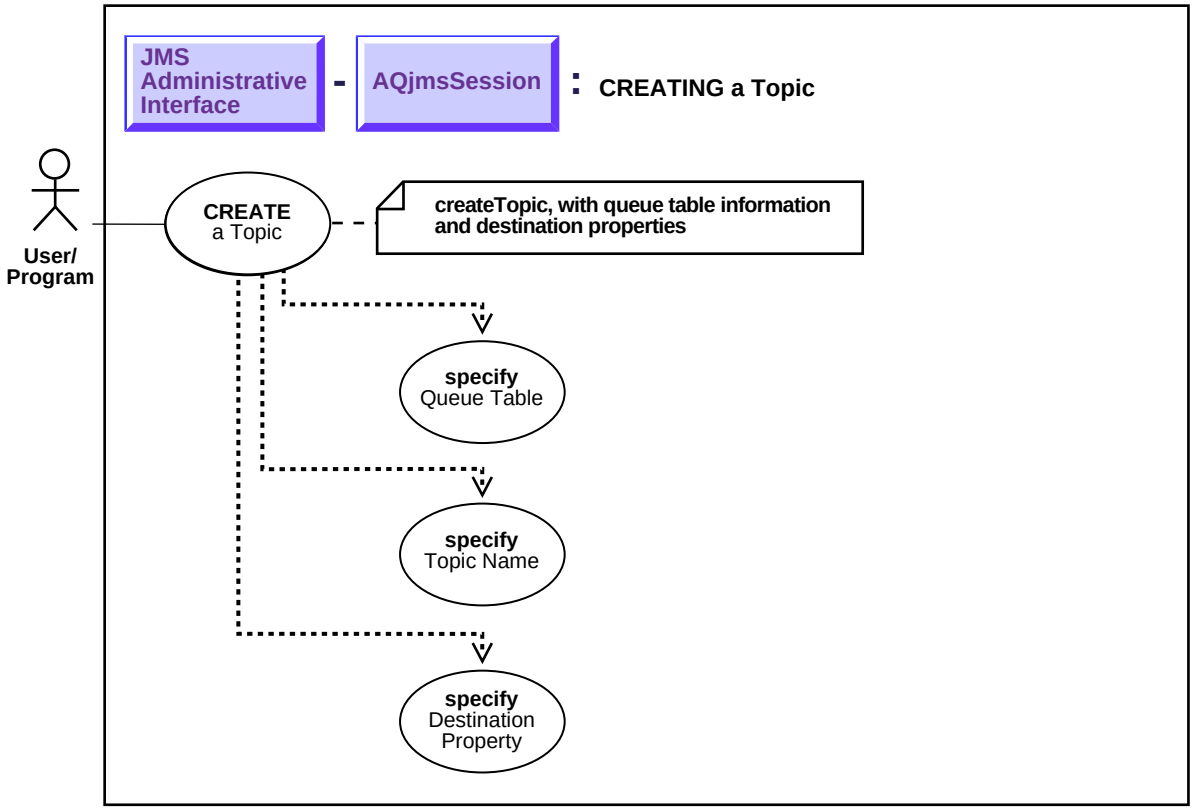
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createQueue

Example

```
QueueSession             q_sess;
AQQueueTable             q_table;
AqjmsDestinationProperty dest_prop;
Queue                    queue;

queue = ((AQjmsSession)q_sess).createQueue(q_table, "jms_q1", dest_prop);
```

Publish-Subscribe - Creating a Topic

Figure 13–21 Use Case Diagram: Publish-Subscribe Create a Topic



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic in the publish-subscribe model.

Usage Notes

Not applicable.

Syntax

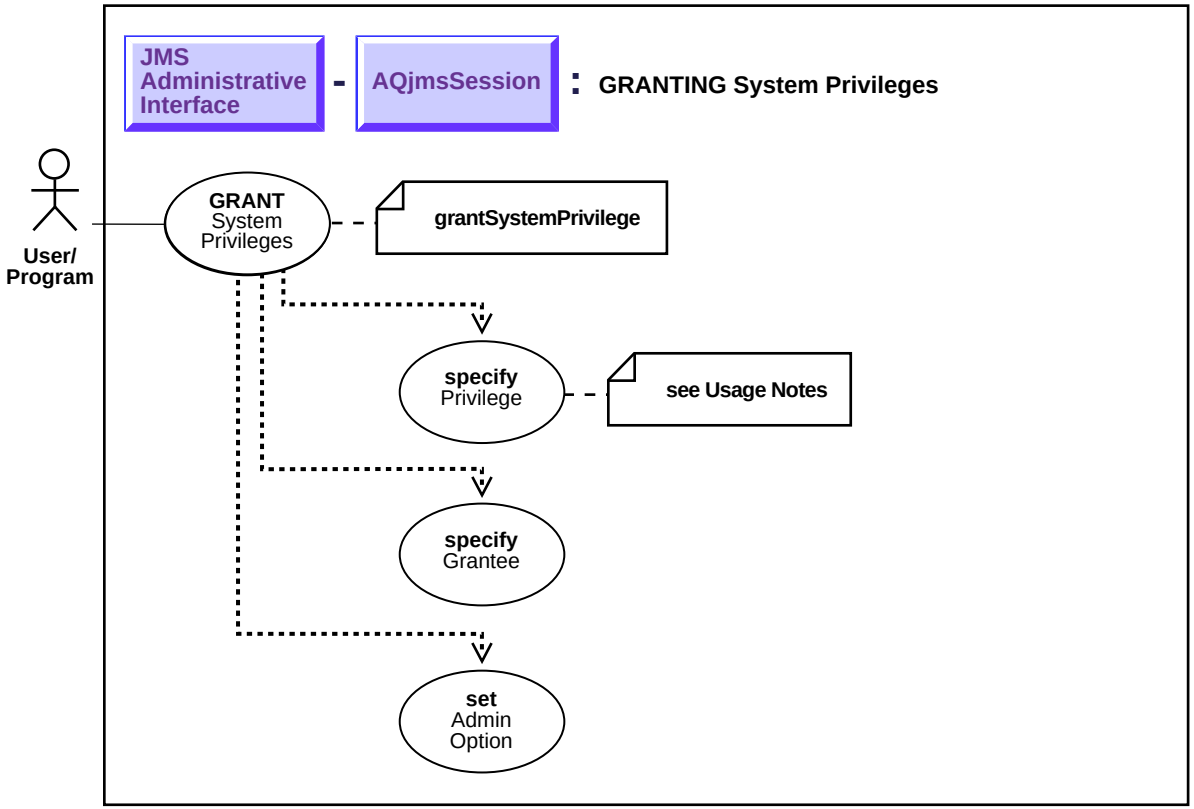
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createTopic

Example

```
TopicSession             t_sess;
AQQueueTable             q_table;
AqjmsDestinationProperty dest_prop;
Topic                    topic;

topic = ((AQjmsSessa)t_sess).createTopic(q_table, "jms_t1", dest_prop);
```

Granting System Privileges

Figure 13–22 Use Case Diagram: Grant System Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Grant AQ system privileges to a user / roles.

Usage Notes

Initially only SYS and SYSTEM can use this procedure successfully.

The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY.

Syntax

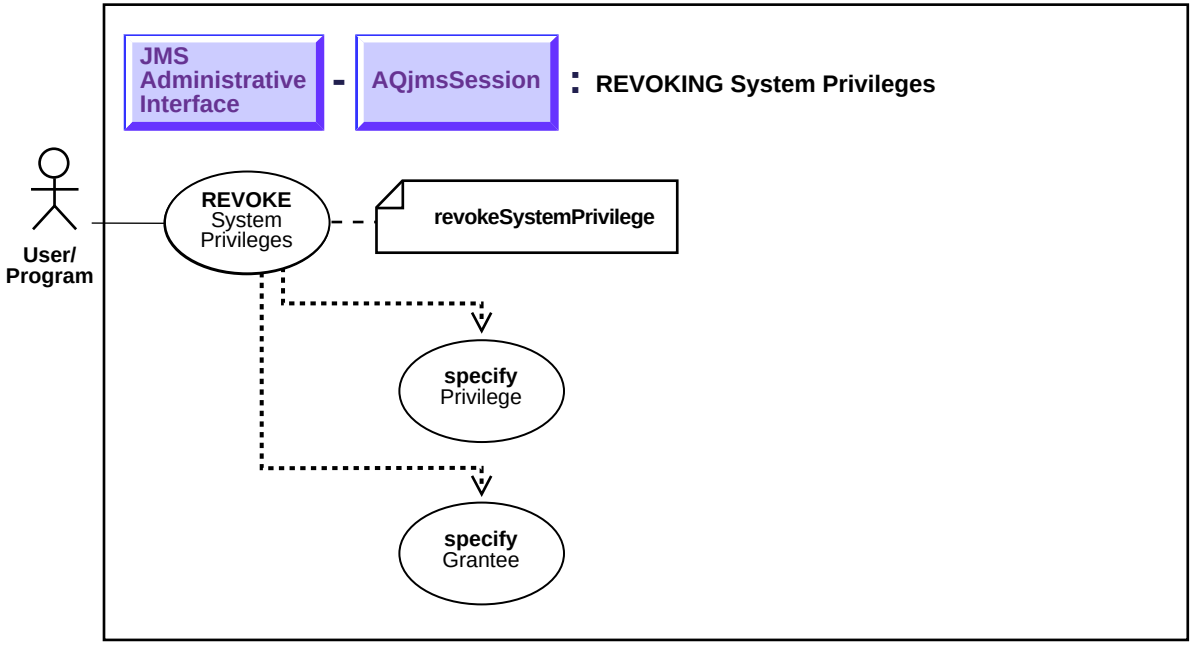
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.grantSystemPrivilege

Example

```
TopicSession            t_sess;

((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "scott", false);
```

Revoking System Privileges

Figure 13–23 Use Case Diagram: Revoke System Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Revoke AQ system privileges from user / roles.

Usage Notes

The privileges are EUQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY.

Syntax

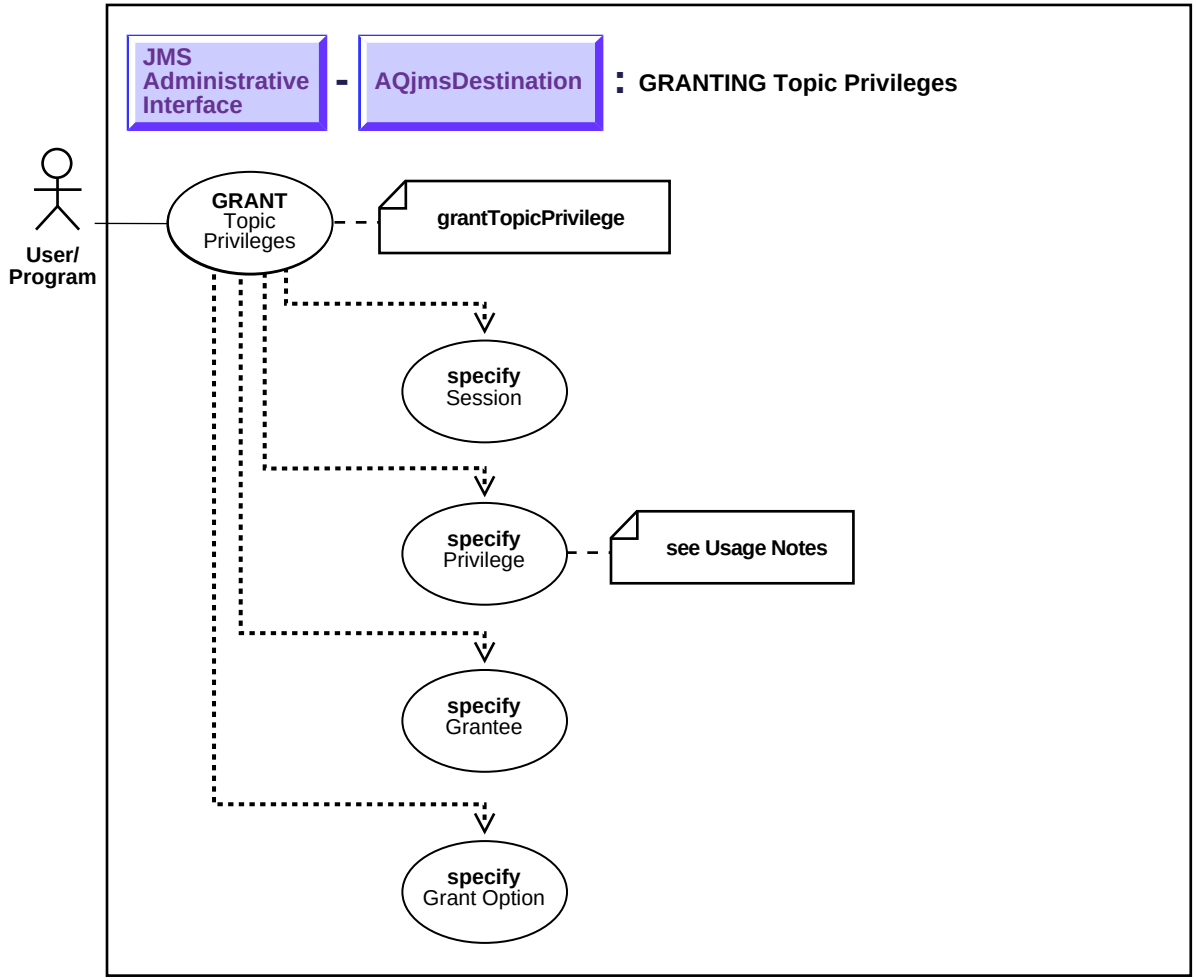
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.revokeSystemPrivilege

Example

```
TopicSession            t_sess;

((AQjmsSession)t_sess).revokeSystemPrivilege("ENQUEUE_ANY", "scott");
```

Publish-Subscribe - Granting Topic Privileges

Figure 13–24 Use Case Diagram: Publish-Subscribe - Grant Topic Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Grant a topic privilege in the publish-subscribe model.

Usage Notes

The privileges are ENQUEUE, DEQUEUE and ALL. ALL means both. Initially only the queue table owner can use this procedure to grant privileges on the topic.

Syntax

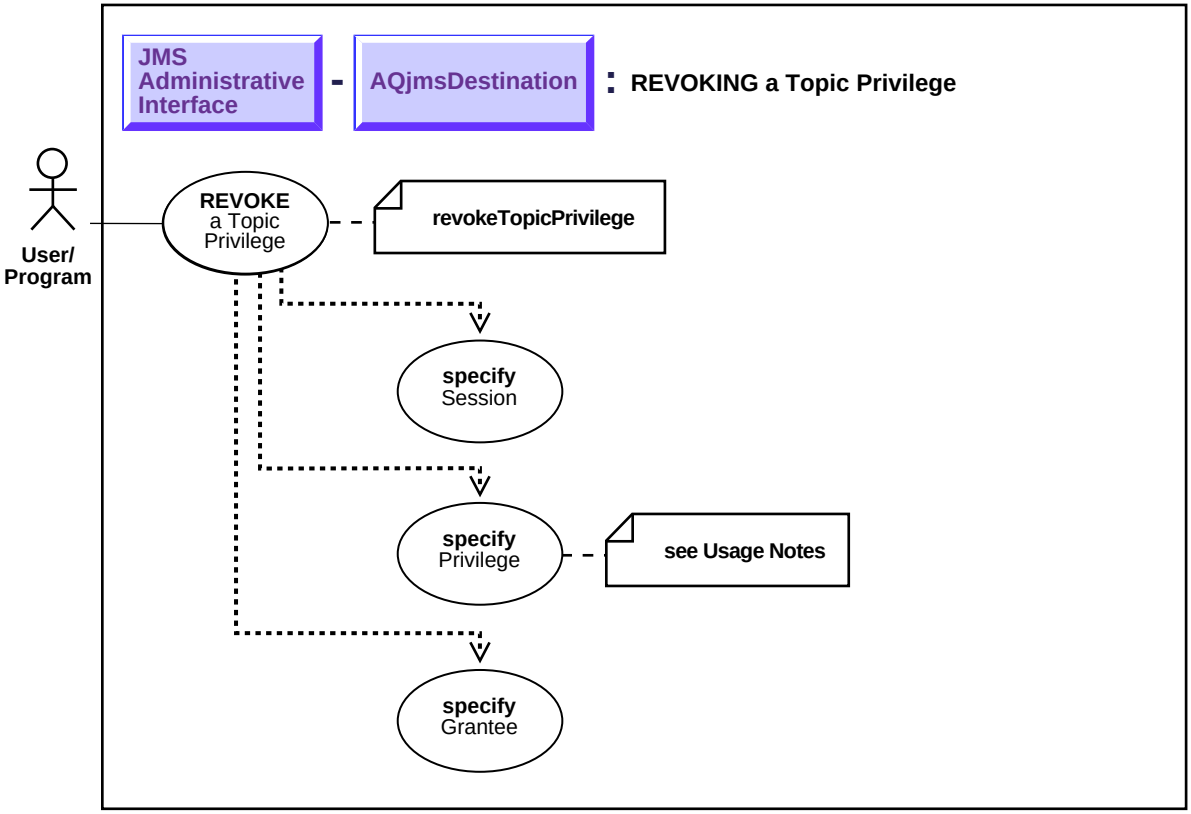
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.grantTopicPrivilege

Example

```
TopicSession            t_sess;
Topic                   topic;

((AQjmsDestination)topic).grantTopicPrivilege(t_sess, "ENQUEUE", "scott",
false);
```

Publish-Subscribe - Revoking Topic Privileges

Figure 13–25 Use Case Diagram: Publish-Subscribe - Revoke Topic Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Revoke a topic privilege in the publish-subscribe model

Usage Notes

The privileges are ENQUEUE, DEQUEUE, and ALL. ALL means both.

Syntax

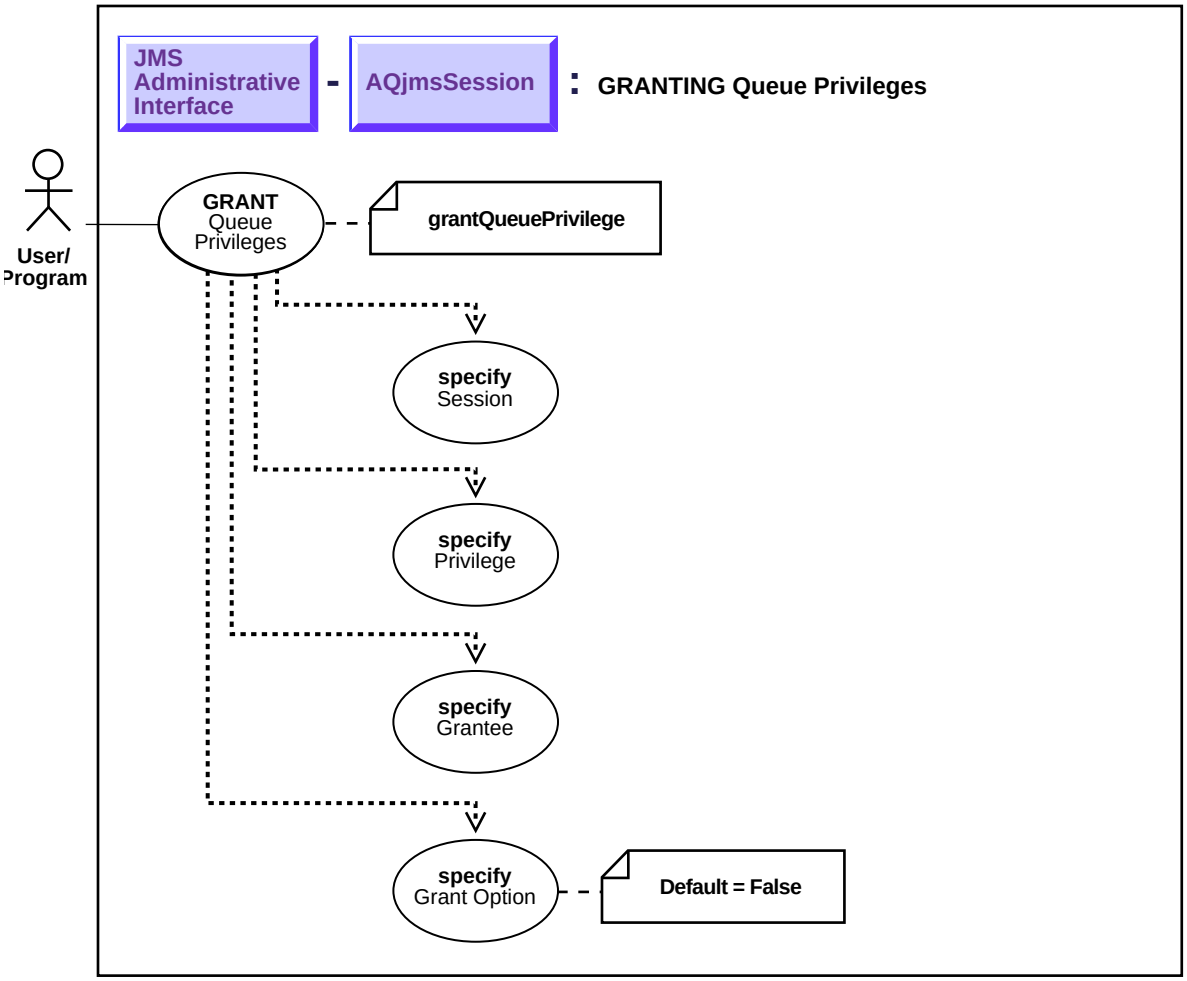
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.revokeTopicPrivilege

Example

```
TopicSession            t_sess;
Topic                   topic;

((AQjmsDestination)topic).revokeTopicPrivilege(t_sess, "ENQUEUE", "scott");
```

Point-to-Point: Granting Queue Privileges

Figure 13–26 Use Case Diagram: Point-to-Point - Grant Queue Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Grant a queue privilege in the point-to-point model

Usage Notes

The privileges are ENQUEUE, DEQUEUE and ALL. ALL means both. Initially only the queue table owner can use this procedure to grant privileges on the queue.

Syntax

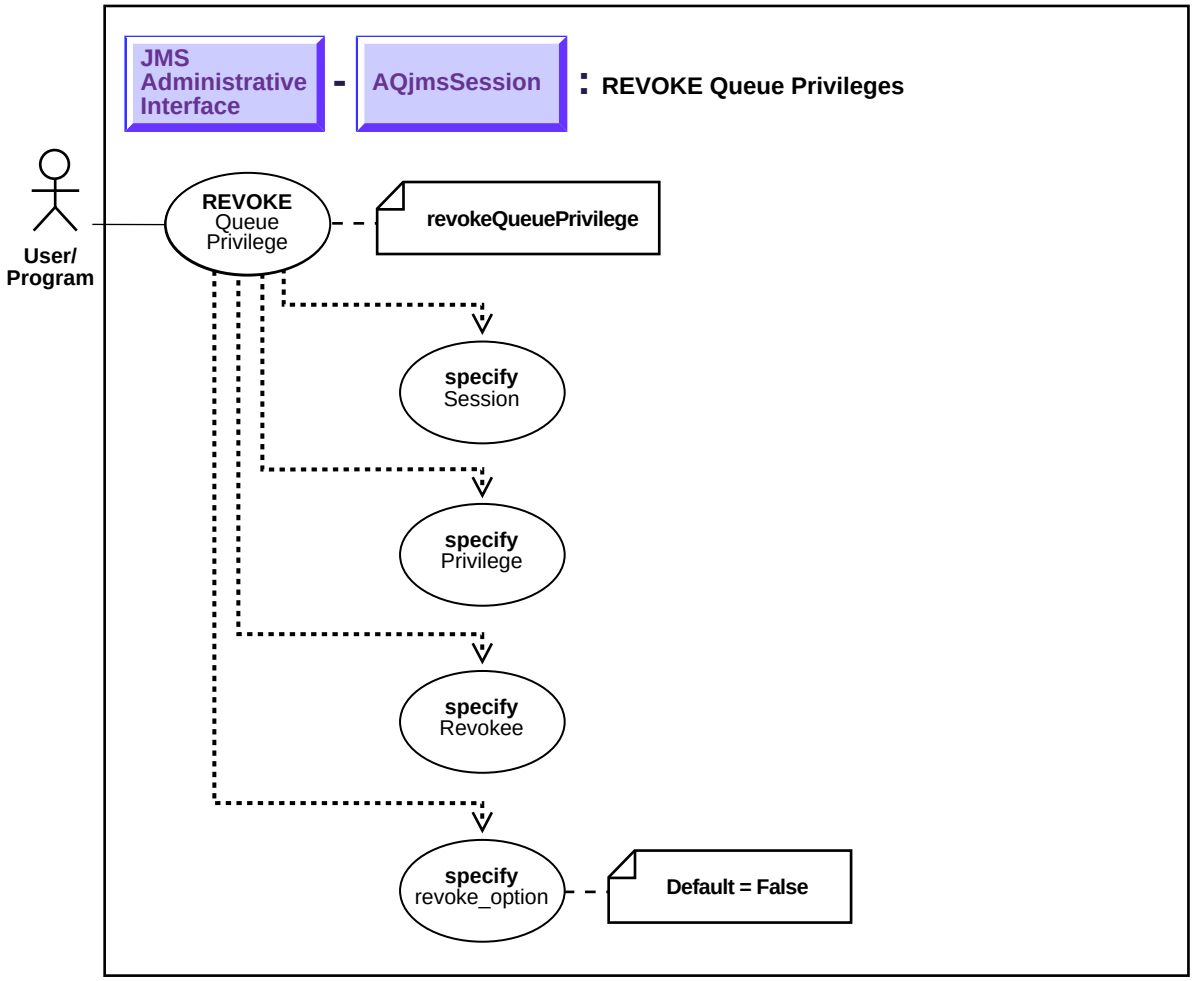
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.grantQueuePrivilege

Example

```
QueueSession            q_sess;
Queue                   queue;

((AQjmsDestination)queue).grantQueuePrivilege(q_sess, "ENQUEUE", "scott",
false);
```

Point-to-Point: Revoking Queue Privileges

Figure 13–27 Use Case Diagram: Point-to-Point: Revoke Queue Privileges



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Revoke queue privilege in the point-to-point model

Usage Notes

The privileges are ENQUEUE, DEQUEUE and ALL. ALL means both. 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.

Syntax

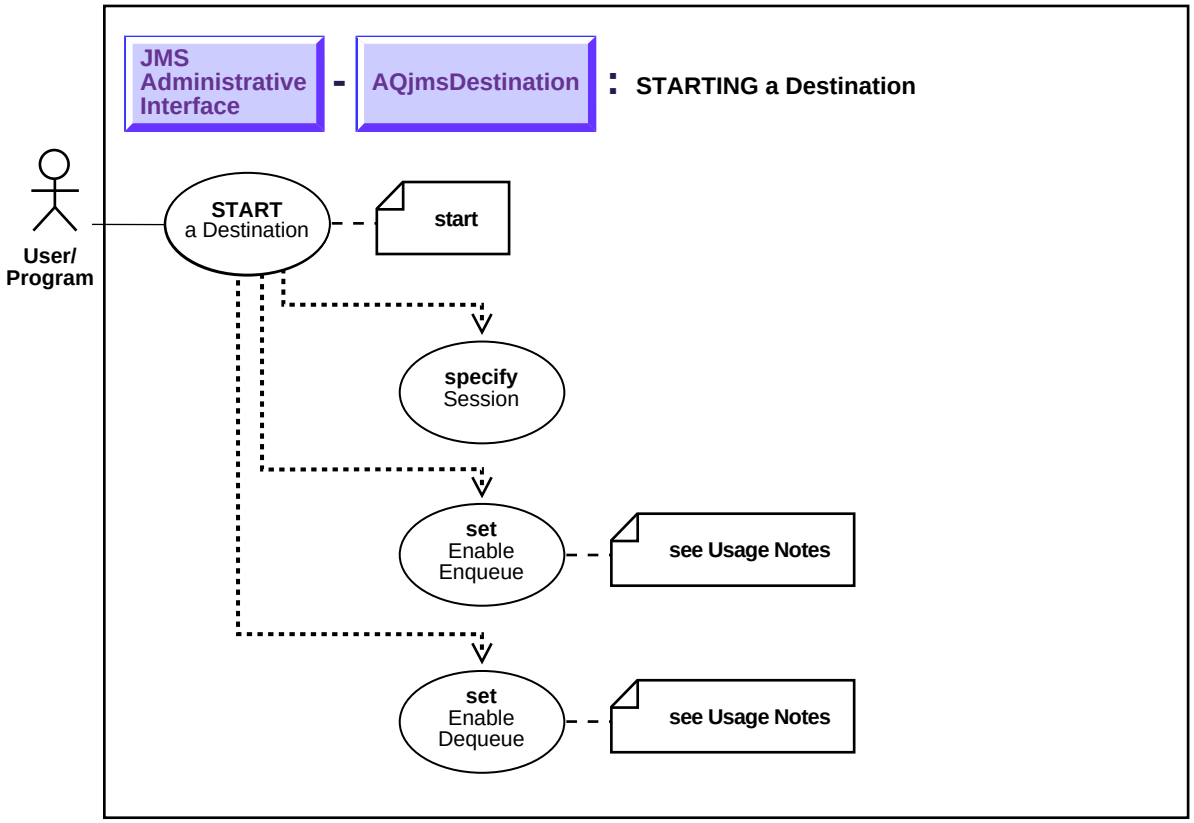
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.revokeQueuePrivilege

Example

```
QueueSession            q_sess;
Queue                   queue;

((AQjmsDestination)queue).revokeQueuePrivilege(q_sess, "ENQUEUE", "scott");
```

Starting a Destination

Figure 13–28 Use Case Diagram: Start a Destination



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Start a destination.

Usage Notes

After creating a destination, the administrator must use the start method to enable the destination. If Enable Enqueue is set to TRUE, then the destination is enabled for enqueue. If Enable Enqueue is set to FALSE, then the destination is disabled for enqueue. Similarly, if Enable Dequeue is set to TRUE, then the destination is enabled for dequeue. If Enable Dequeue is set to FALSE, the destination is disabled for dequeue.

Syntax

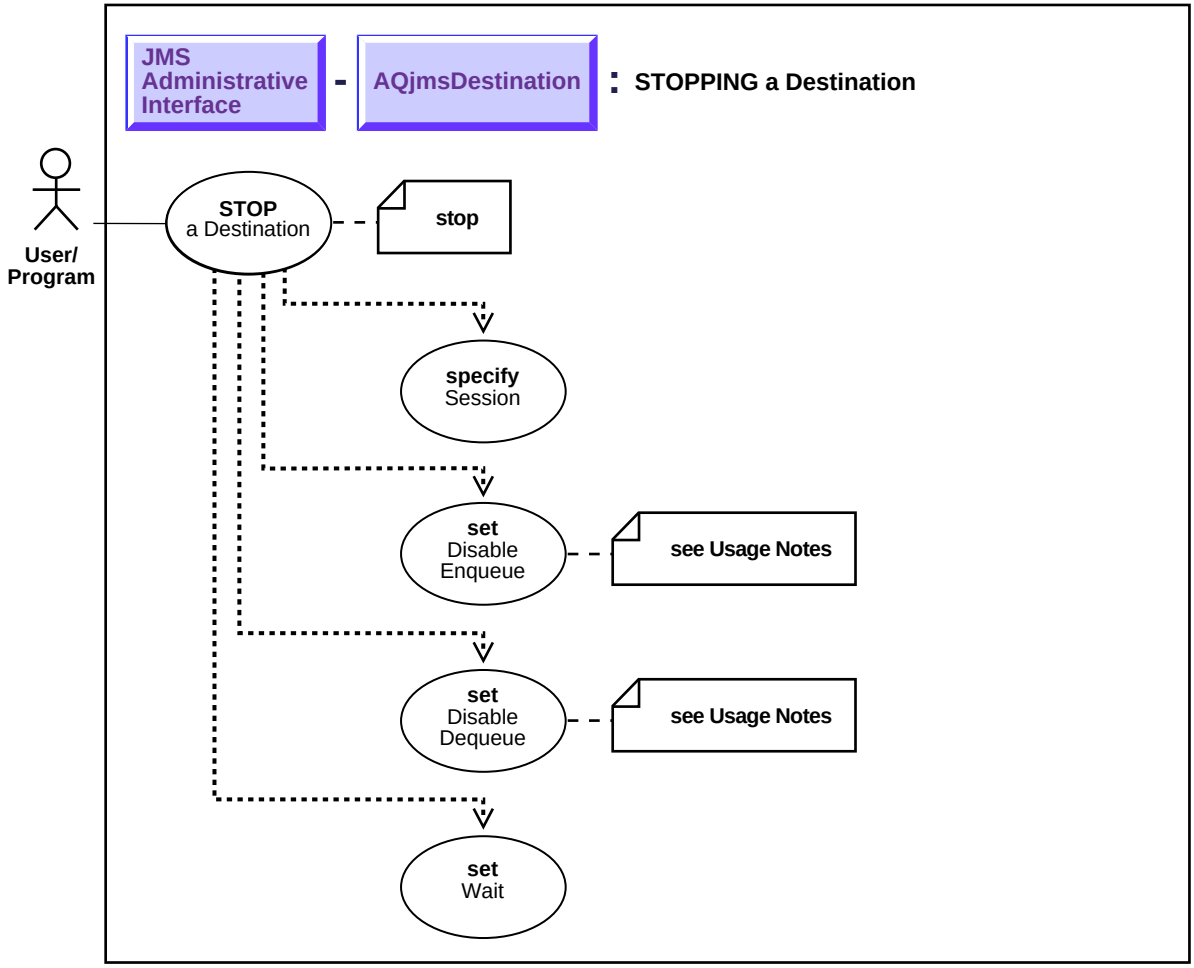
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.start

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

Figure 13–29 Use Case Diagram: Stop a Destination



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Stop a destination.

Usage Notes

If Disable Dequeue is set to TRUE, then the destination is disabled for dequeue. If Disable dequeue is set to FALSE, then the current setting is not altered. Similarly if Disable Enqueue set to TRUE, then the destination is disabled for enqueue. If Disable Enqueue is set to FALSE, then the current setting is not altered.

Syntax

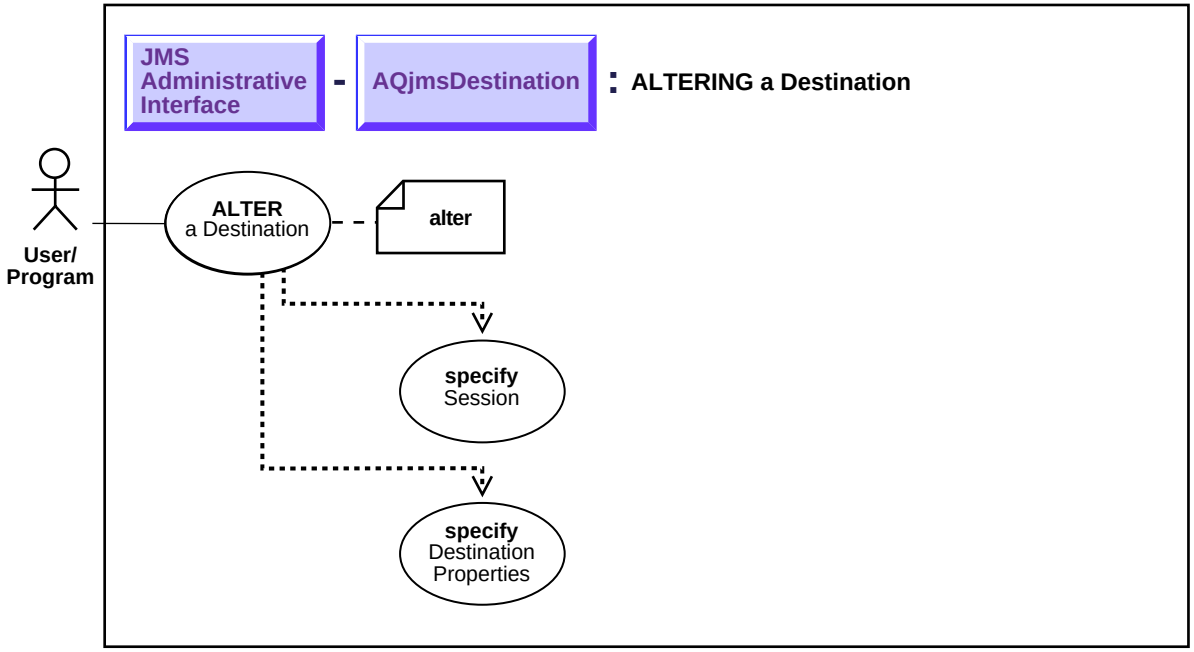
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.stop

Example

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).stop(t_sess, true, false);
```

Altering a Destination

Figure 13–30 Use Case Diagram: Alter a Destination



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Alter a destination.

Usage Notes

Not applicable.

Syntax

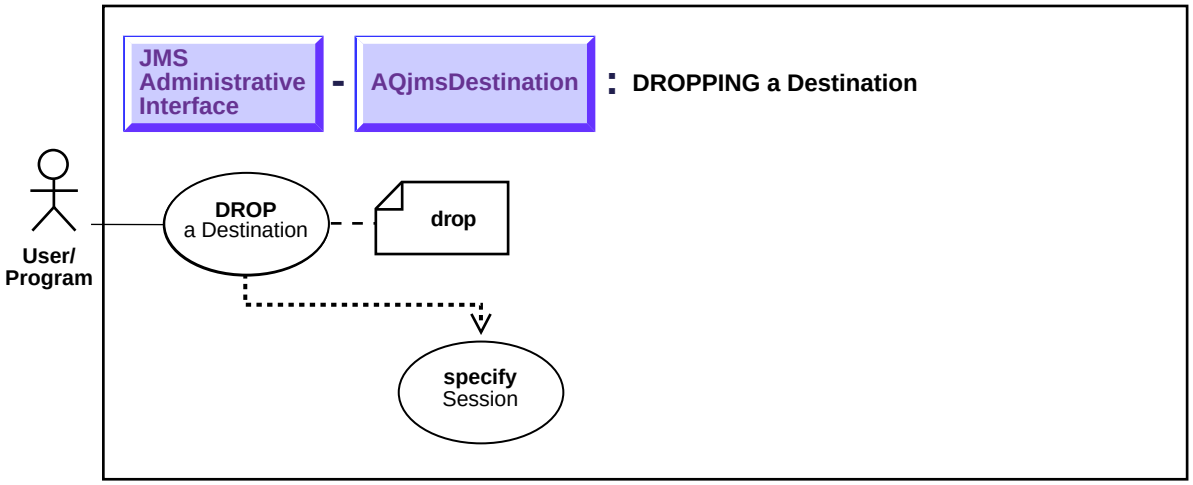
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.alter

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

Figure 13–31 Use Case Diagram: Drop a Destination



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Drop a destination.

Usage Notes

Not applicable.

Syntax

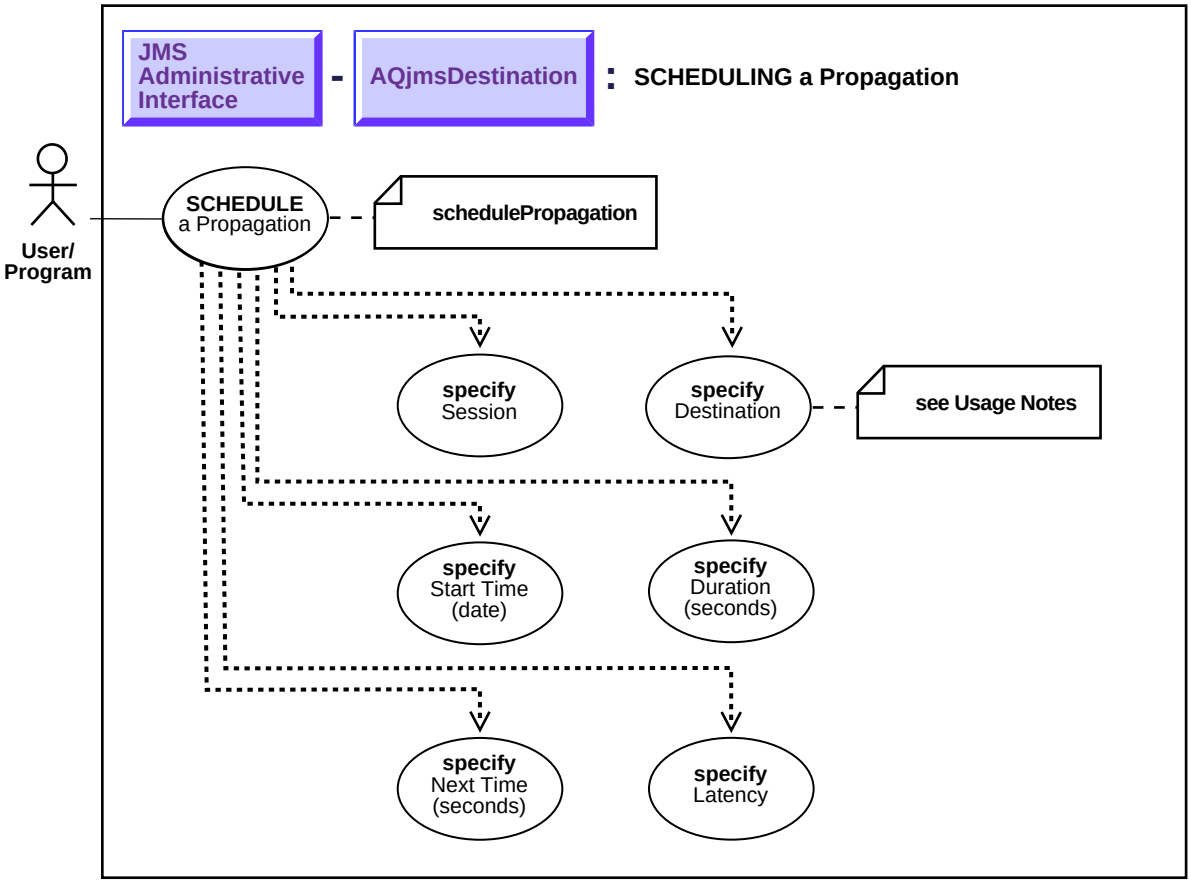
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.drop

Example

```
QueueSession q_sess;
Queue        queue;
TopicSession t_sess;
Topic        topic;

((AQjmsDestionation)queue).drop(q_sess);
((AQjmsDestionation)topic).drop(t_sess);
```

Scheduling a Propagation

Figure 13–32 Use Case Diagram: Schedule a Propagation



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Schedule a Propagation

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 the message will be propagated to all of them at the same time.

Syntax

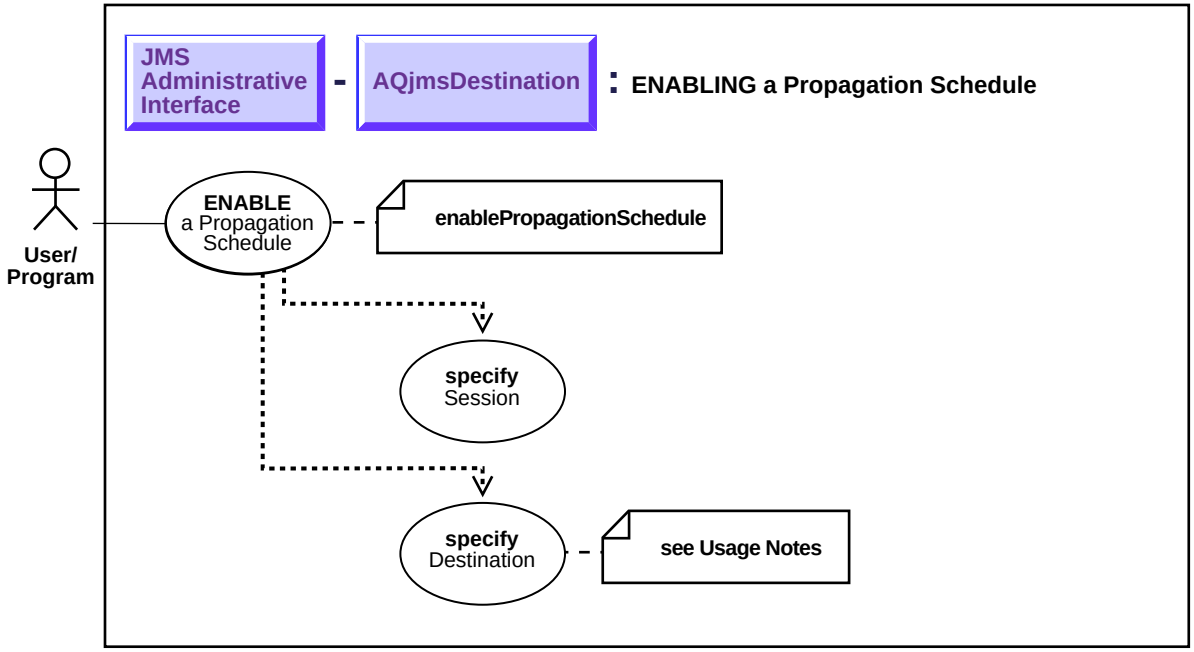
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.schedulePropagation

Example

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).schedulePropagation(t_sess, null, null, null, null,
new Double(0));
```

Enabling a Propagation Schedule

Figure 13–33 Use Case Diagram: Enable a Propagation Schedule



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Enable a Propagation Schedule

Usage Notes

NULL destination indicates that the propagation is to the local database.

Syntax

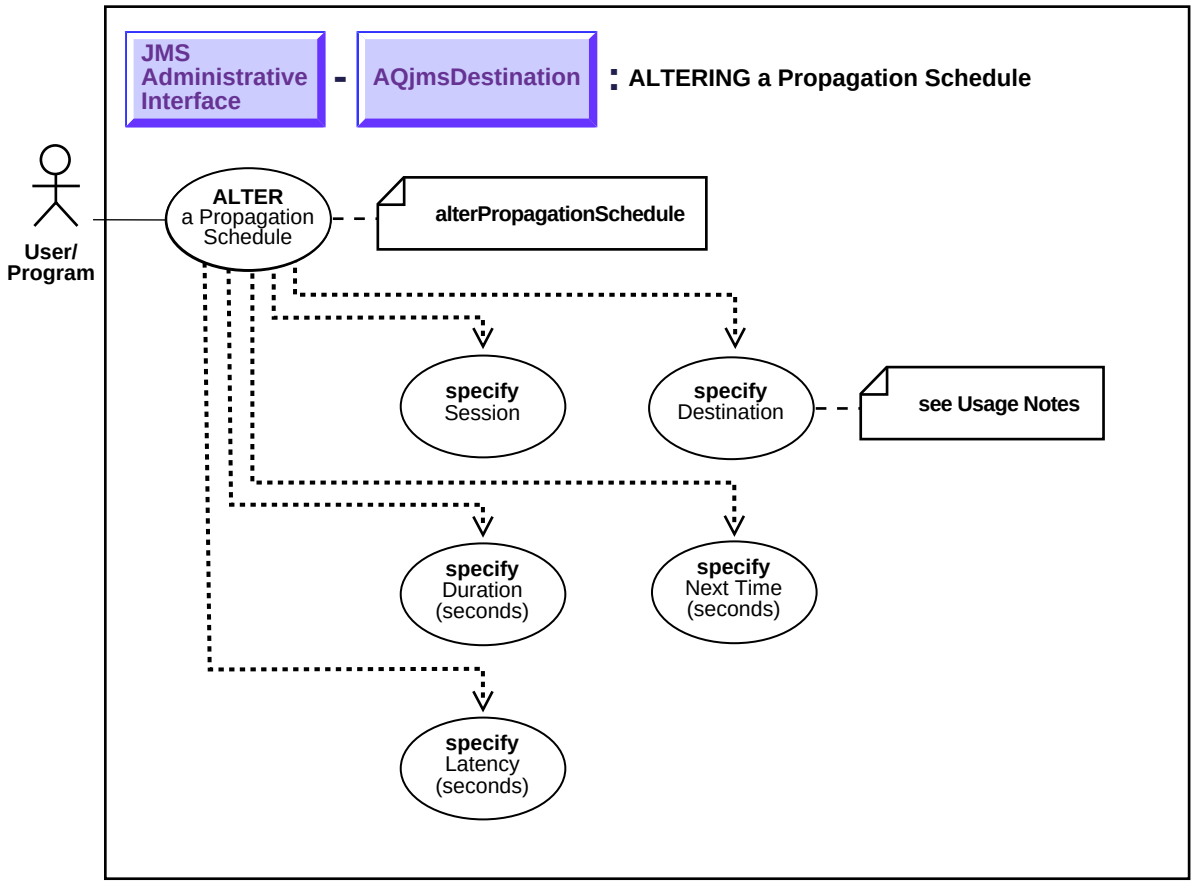
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.enablePropagationSchedule

Example

```
TopicSession            t_sess;
Topic                   topic;

 ((AQjmsDestination)topic).enablePropagationSchedule(t_sess, "dbs1");
```

Altering a Propagation Schedule

Figure 13–34 Use Case Diagram: Alter a Propagation Schedule



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Alter a propagation schedule.

Usage Notes

NULL destination indicates that the propagation is to the local database

Syntax

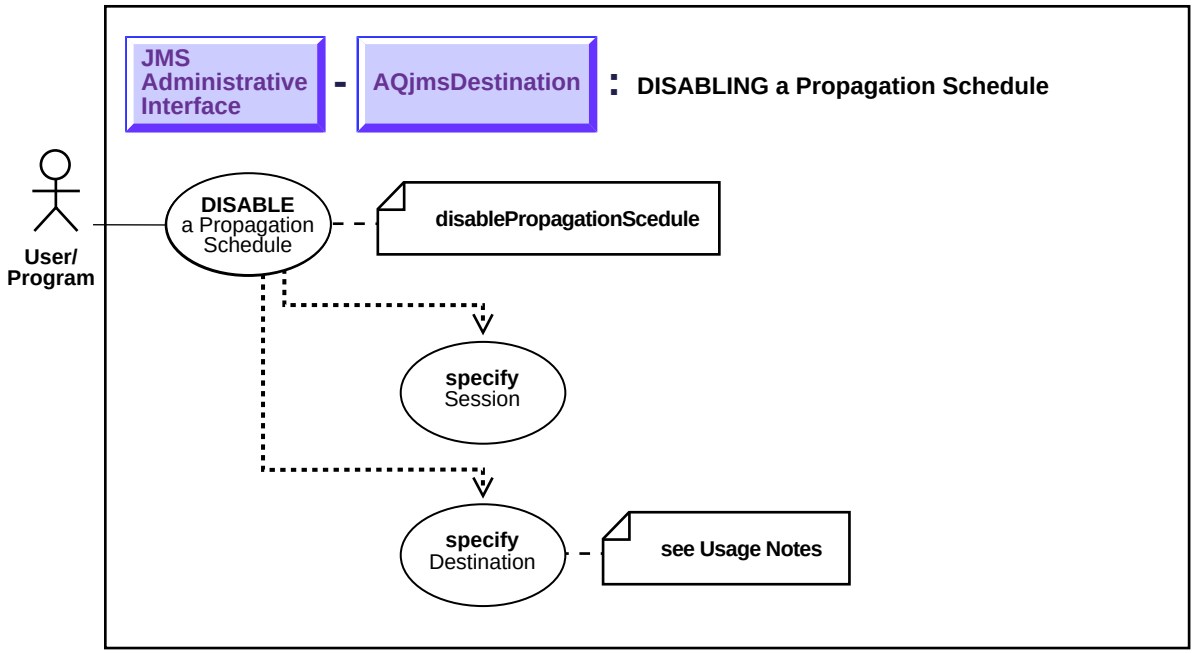
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.alterPropagationSchedule

Example

```
TopicSession            t_sess;
Topic                   topic;

 ((AQjmsDestination)topic).alterPropagationSchedule(t_sess, null, 30, null, new
Double(30));
```

Disabling a Propagation Schedule

Figure 13–35 Use Case Diagram: Disable a Propagation Schedule



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Disable a propagation schedule.

Usage Notes

NULL destination indicates that the propagation is to the local database

Syntax

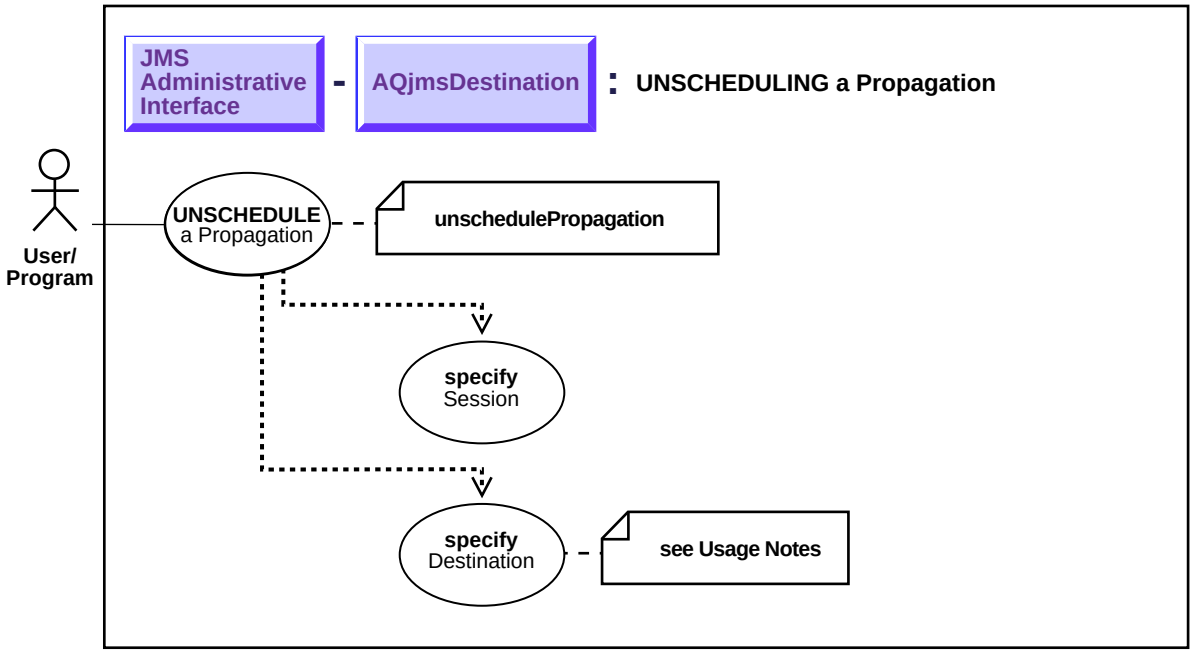
See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.disablePropagationSchedule

Example

```
TopicSession            t_sess;
Topic                   topic;

 ((AQjmsDestination)topic).disablePropagationSchedule(t_sess, "dbs1");
```

Unscheduling a Propagation

Figure 13–36 Use Case Diagram: Unschedule a Propagation



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Unschedule a propagation.

Usage Notes

Unschedule a previously scheduled propagation.

Syntax

See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsDestination.unschedulePropagation

Example

```
TopicSession   t_sess;
Topic          topic;

((AQjmsDestination)topic).unschedulePropagation(t_sess, "dbs1");
```

14

JMS Operational Interface: Basic Operations (Point-to-Point)

In this chapter we describe the operational interface to Oracle Advanced Queuing in terms of use cases. That is, we discuss each operation (such as "Creating a Queue Sender") as a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: Operational Interface — Basic Operations" on page 14-2).

A summary figure, "Use Case Diagram: Operational Interface — Basic Operations", locates all the use cases in a single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case that interests you by clicking on the relevant use case title.

Each use case is laid out as follows:

- *Use case figure.* A figure that depicts the use case.
- *Purpose.* The purpose of this use case.
- *Usage Notes.* Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment that illustrate the use case.

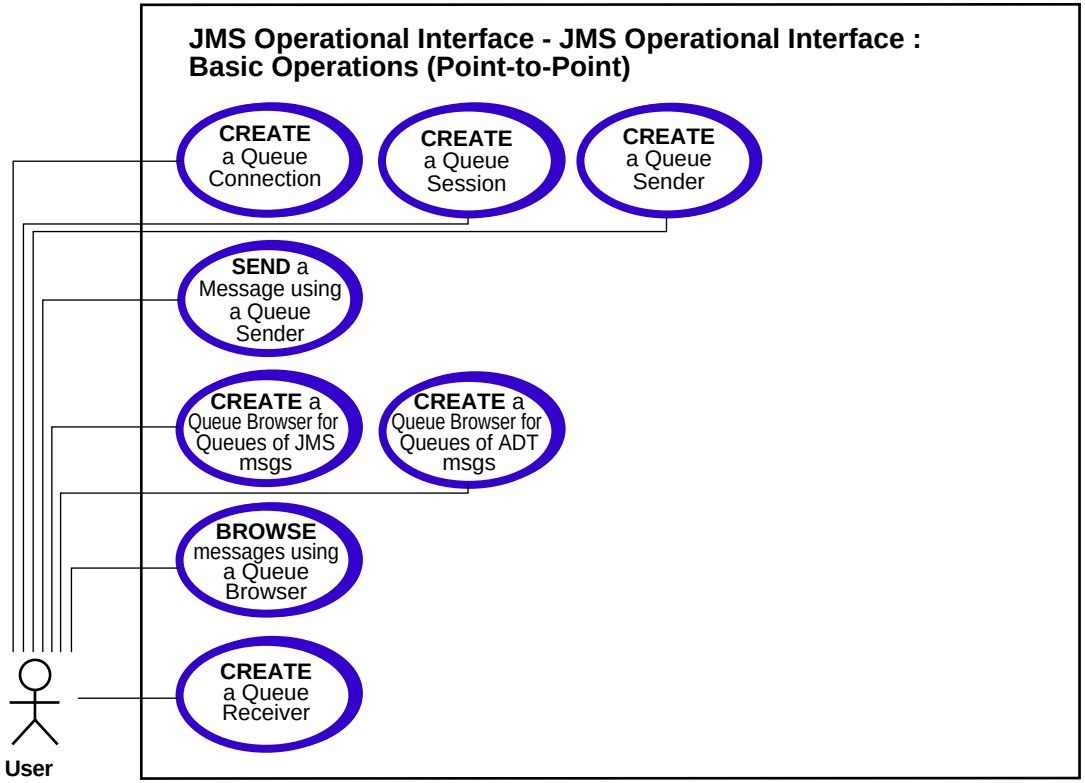
Use Case Model: Operational Interface — Basic Operations

Table 14–1 Use Case Model: Operational Interface — Basic Operations

Use Case
Four Ways of Creating a Queue Connection on page 14-3
Creating a Queue Connection with Username/Password on page 14-5
Creating a Queue Connection with Open JDBC Connection on page 14-6
Creating a Queue Connection with Default Connection Factory Parameters on page 14-8
Creating a Queue Connection with an Open OracleOCIConnection Pool on page 14-9
Creating a Queue Session on page 14-11
Creating a Queue Sender on page 14-13
Two Ways of Sending Messages Using a Queue Sender on page 14-14
Sending a Message Using a Queue Sender with Default Send Options on page 14-15
Sending Messages Using a Queue Sender by Specifying Send Options on page 14-17
Two Ways of Creating a Queue Browser for JMS Message Queues on page 14-20
Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages on page 14-21
Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages, Locking Messages while Browsing on page 14-23
Two Ways of Creating a Queue Browser for Oracle Object Type (ADT) Messages Queues on page 14-25
Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages on page 14-26
Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages, Locking Messages While Browsing on page 14-28
Browsing Messages Using a Queue Browser on page 14-29
Two Ways of Creating a Queue Receiver on page 14-31
Creating a Queue Receiver for Queues of Standard JMS Type Messages on page 14-32x
Creating a Queue Receiver for Queues of Oracle Object Type (ADT) Messages on page 14-34
Creating a Queue Connection with an Open OracleOCIConnection Pool on page 14-36

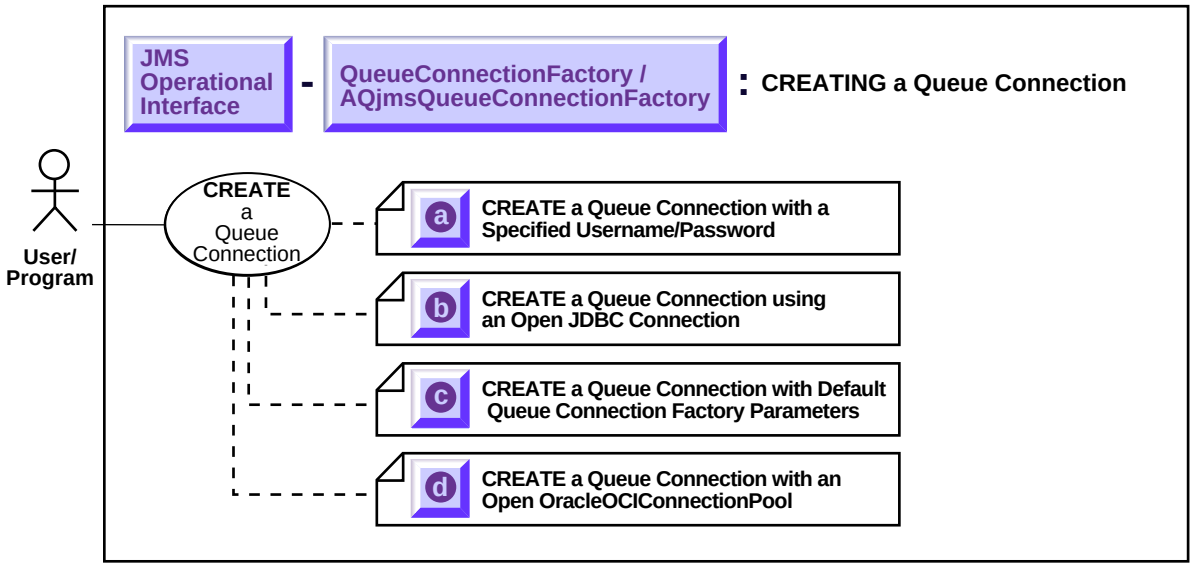
Use Case Model Diagram: Operational Interface (Point-to-Point)

Figure 14–1 Use Case Model Diagram: Operational Interface (Point-to-Point)



Four Ways of Creating a Queue Connection

Figure 14–2 Use Case Diagram: Multiple Ways to Create a Queue Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

This section explains three ways to create a queue connection:

a. Creating a Queue Connection with Username / Password on page 14-5

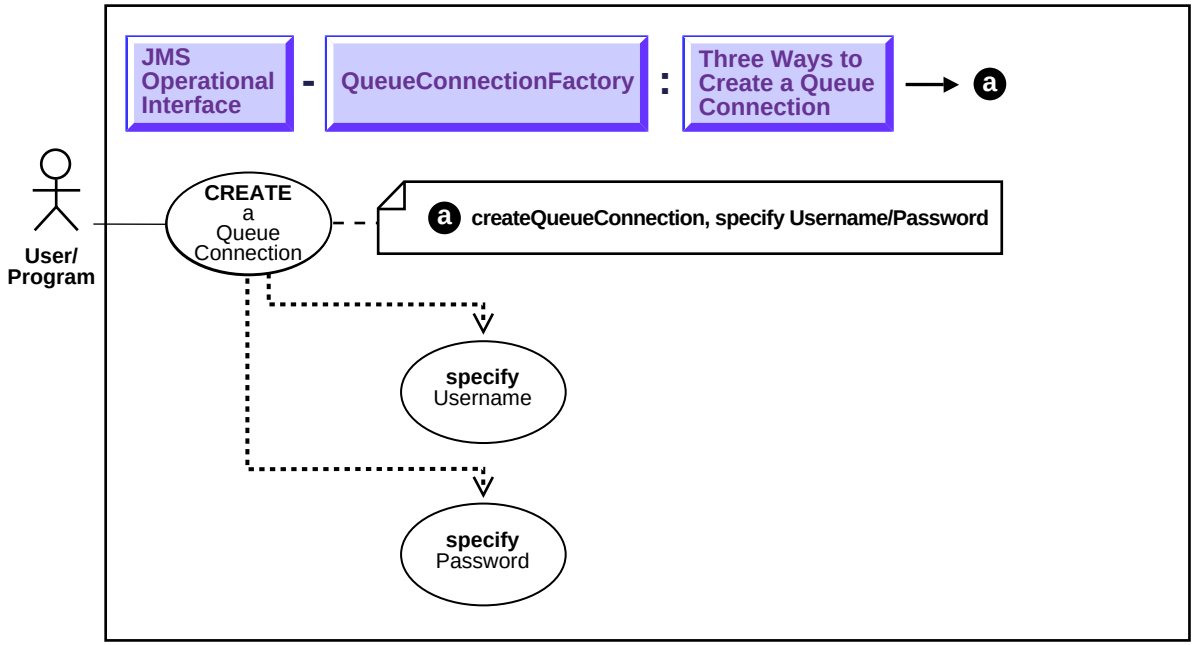
b. Creating a Queue Connection with Open JDBC Connection on page 14-6

c. Creating a Queue Connection with Default Connection Factory Parameters on page 14-8

d. Creating a Queue Connection with an Open OracleOCIConnection Pool on page 14-9

Creating a Queue Connection with Username/Password

Figure 14–3 Use Case Diagram: Create a Queue Connection with Username/Password



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue connection with username/password.

Usage Notes

Not applicable.

Syntax

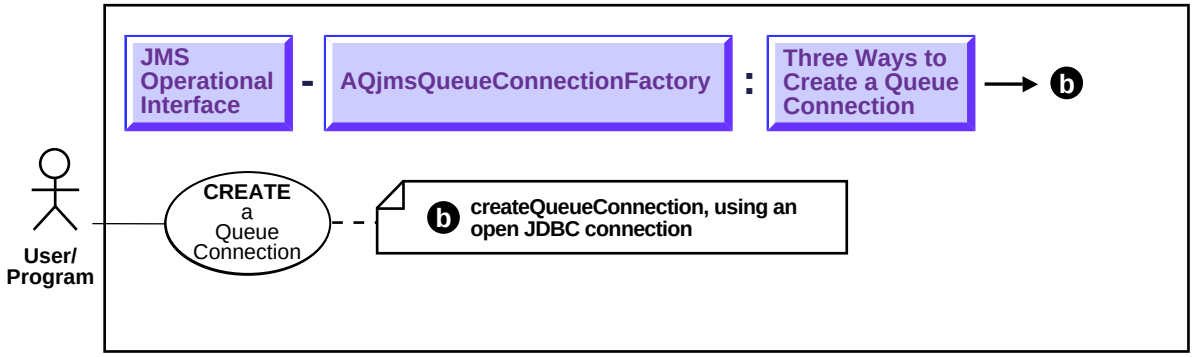
- Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsQueueConnectionFactory.createQueueConnection

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 Open JDBC Connection

Figure 14–4 Use Case Diagram: Create a Queue Connection with Open JDBC Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue connection with open JDBC connection.

Usage Notes

This is a static method.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueConnectionFactory.createQueueConnection

Example

Example 1

This method may be used if the user wants to use an existing JDBC connection (say from a connection pool) for JMS operations. In this case JMS will 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);
```

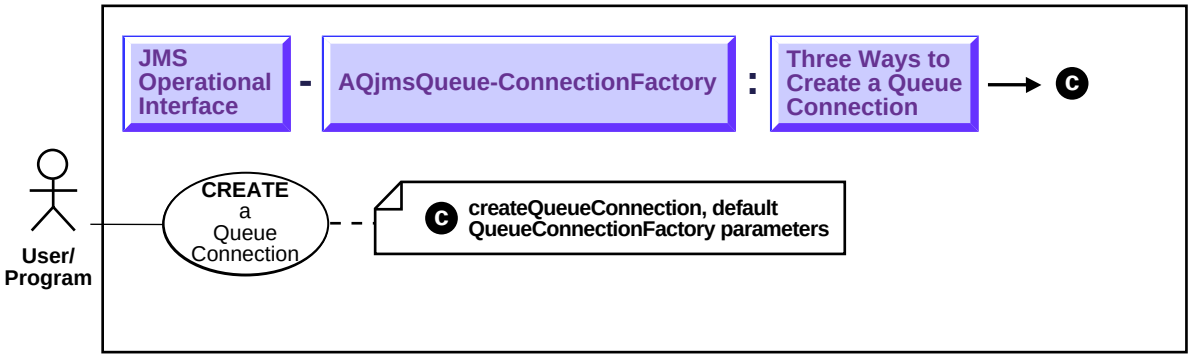
Example 2

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

Figure 14–5 Use Case Diagram: Create a Connection with Default Connection Factory Parameters



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue connection with default connection factory parameters.

Usage Notes

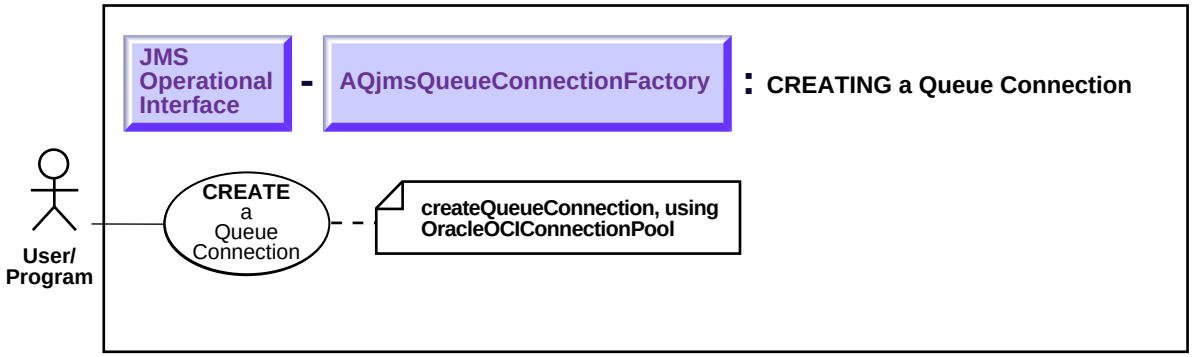
The QueueConnectionFactory properties must contain a default username and password: otherwise, this method will throw a JMSException.

Syntax

Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueConnectionFactory.createQueueConnection

Creating a Queue Connection with an Open OracleOCIConnection Pool

Figure 14–6 Use Case Diagram: Create a Queue Connection with an Open OracleOCIConnectionPool



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue connection with an open OracleOCIConnectionPool.

Usage notes

This is a static method.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueConnectionFactory.createQueueConnection

Example

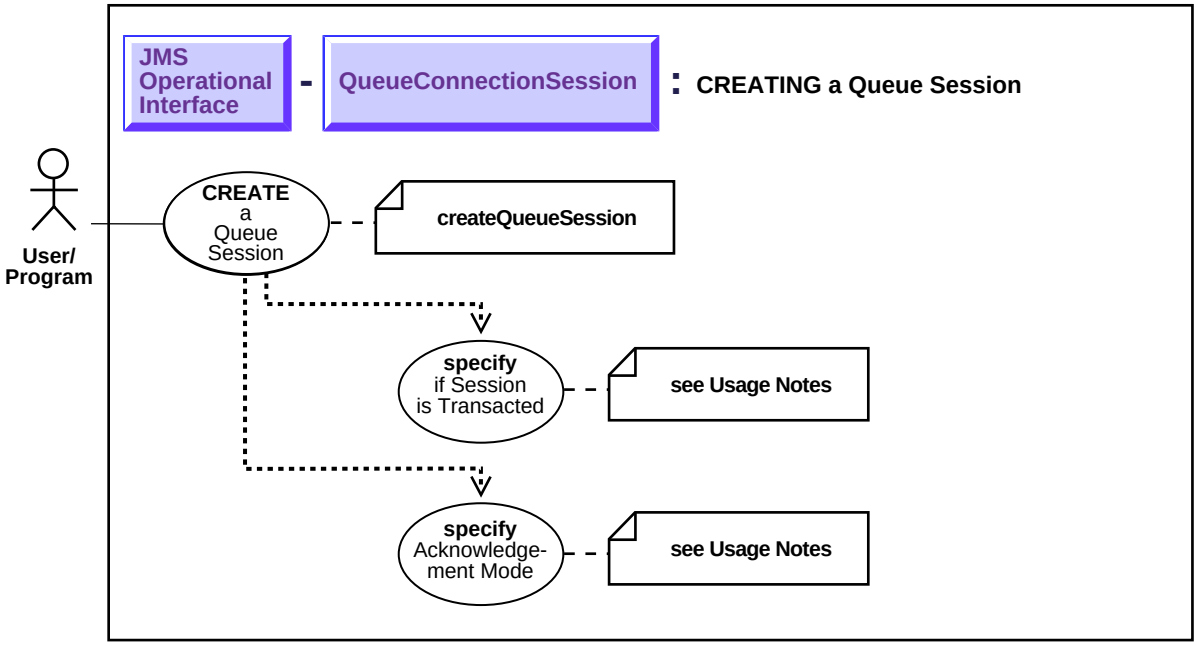
This method may be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS will

not open an new `OracleOCIConnectionPool` instance, but instead use the supplied `OracleOCIConnectionPool` instance to create the JMS QueueConnection object.

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(cpool);
```

Creating a Queue Session

Figure 14–7 Use Case Diagram: Create a Queue Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue session.

Usage Notes

In the current version, only transacted sessions are supported.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsConnection.createQueueSession

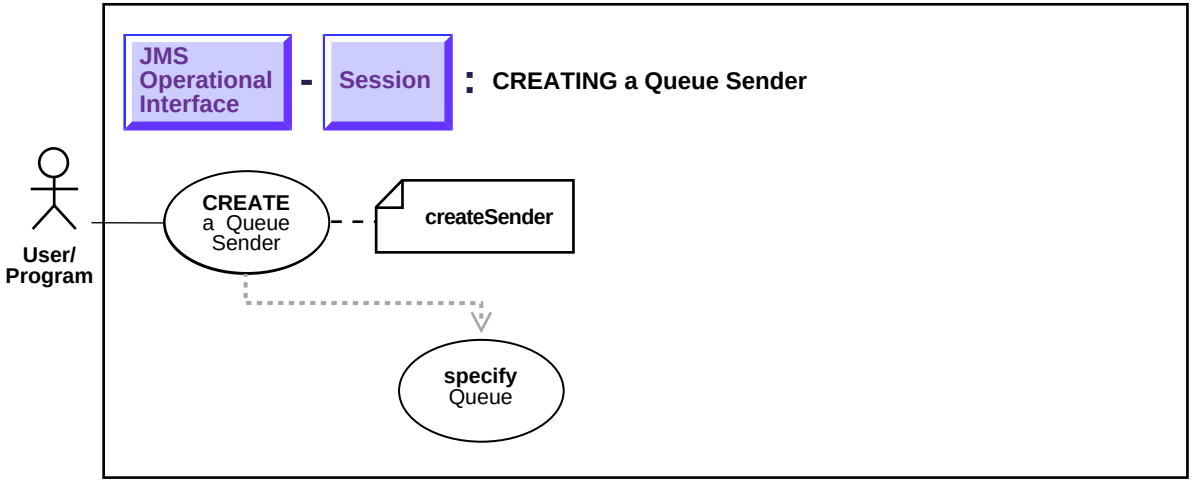
Example

In the current version, only transacted sessions are supported.

```
QueueConnection qc_conn;
QueueSession  q_sess = qc_conn.createQueueSession(true, 0);
```

Creating a Queue Sender

Figure 14–8 Use Case Diagram: Create a Queue Sender



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue sender.

Usage Notes

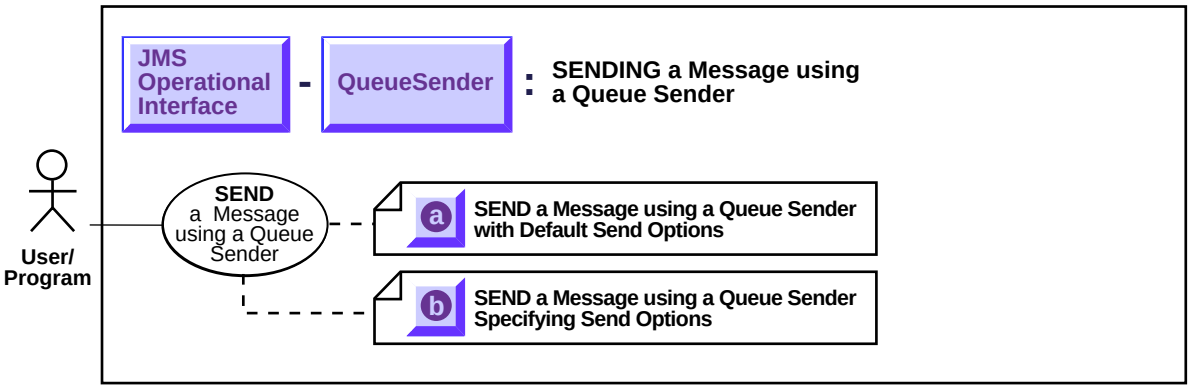
If a sender is created without a default Queue, then the destination Queue will have to be specified on every send operation.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createSender

Two Ways of Sending Messages Using a Queue Sender

Figure 14–9 Use Case Diagram: Two Ways to Send Messages Using a Queue Sender



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

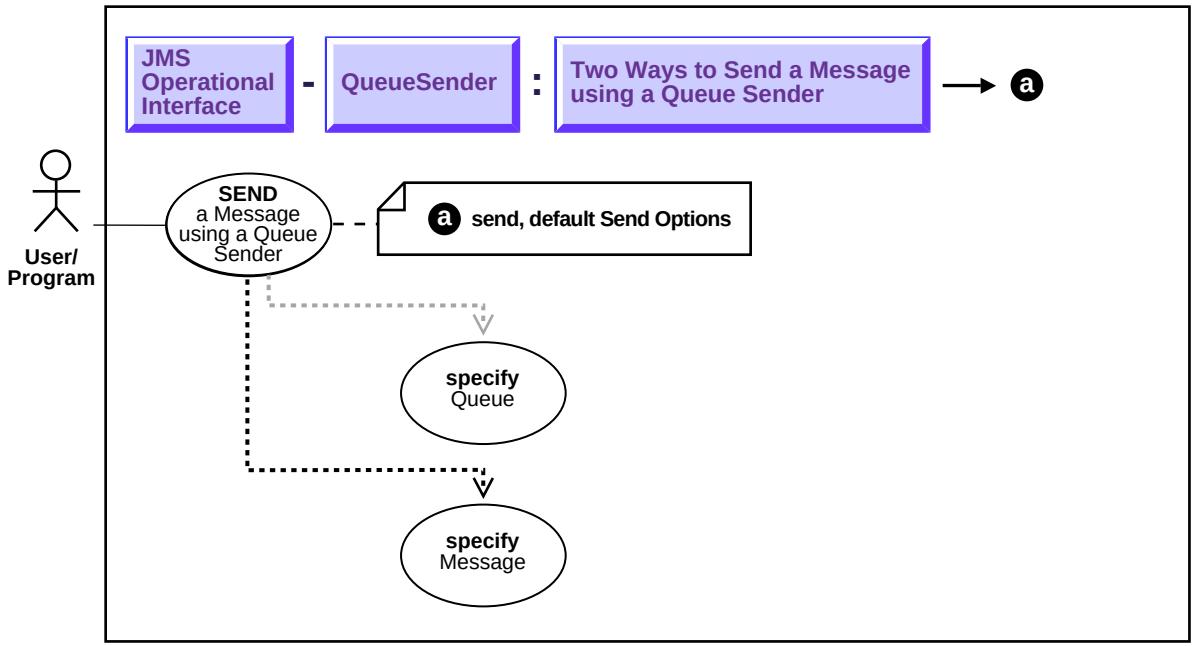
This section explains two ways to send messages using a queue sender:

a. Sending a Message Using a Queue Sender with Default Send Options on page 14-15

b. Sending Messages Using a Queue Sender by Specifying Send Options on page 14-17

Sending a Message Using a Queue Sender with Default Send Options

Figure 14–10 Use Case Diagram: Send a Message Using a Queue Sender with Default Send Options



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Send a message using a queue sender with default send options.

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 will override 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).

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueSender.send

Example

Example1

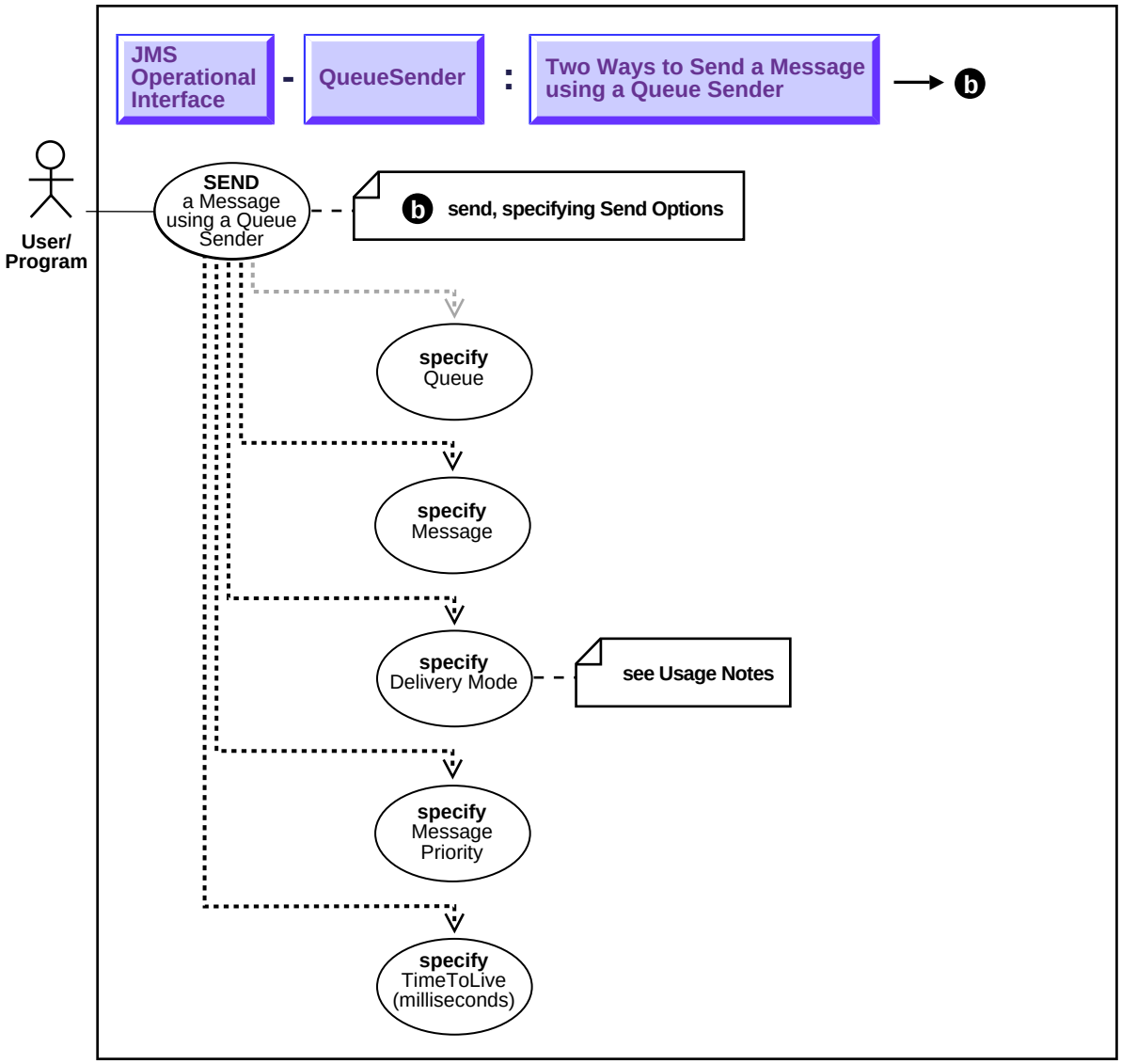
```
/* Create a sender to send messages to any queue */
QueueSession  jms_sess;
QueueSender  sender1;
TextMessage  message;
sender1 = jms_sess.createSender(null);
sender1.send(queue, message);
```

Example2

```
/* 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 Queue Sender by Specifying Send Options

Figure 14–11 Use Case Diagram: Send Messages using a Queue Sender by Specifying Send Options



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Send messages using a queue sender by specifying send options.

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 will override 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.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueSender.send

Example

Example1

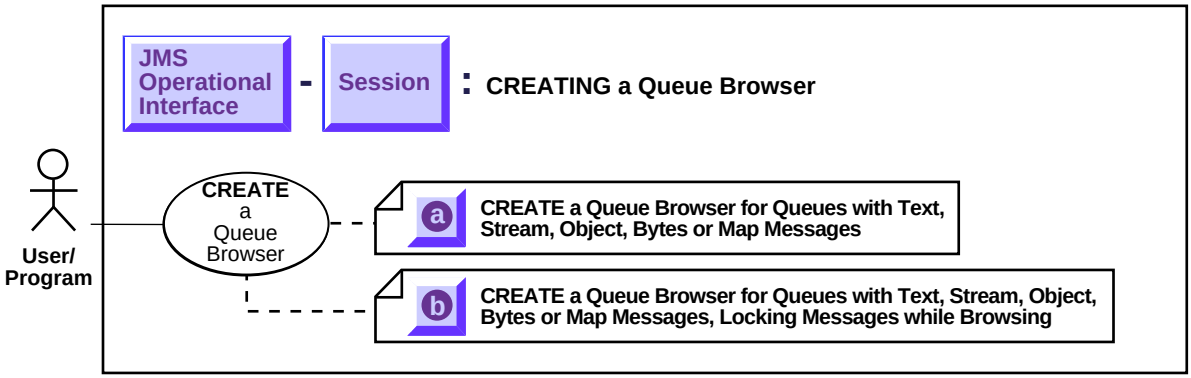
```
/* Create a sender to send messages to any queue */
/* Send a message to new_orders_que with priority 2 and timetoLive 100000
   milliseconds */
QueueSession  jms_sess;
QueueSender  sender1;
TextMessage mesg;
Queue   new_orders_que
sender1 = jms_sess.createSender(null);
sender1.send(new_orders_que, mesg, DeliveryMode.PERSISTENT, 2, 100000);
```

Example2

```
/* 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);
```

Two Ways of Creating a Queue Browser for JMS Message Queues

Figure 14–12 Use Case Diagram: Two Ways to Create a Queue Browser for Queues of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

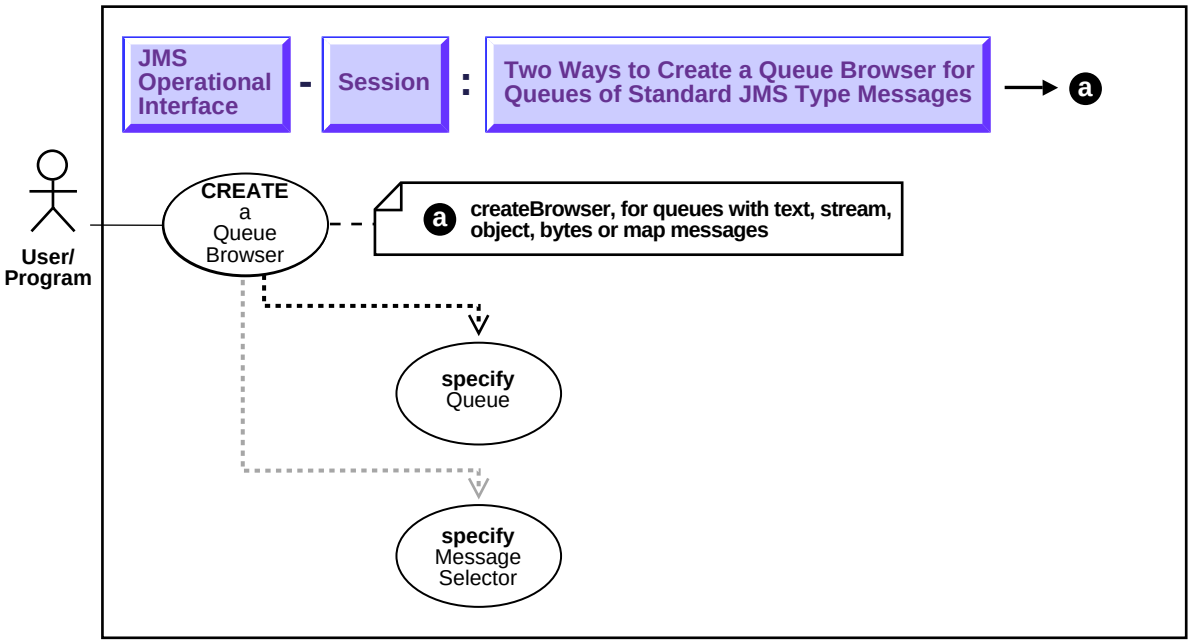
This section explains two ways to create a queue browser for JMS message queues:

a. Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages on page 14-21

b. Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages, Locking Messages while Browsing on page 14-23

Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages

Figure 14–13 Use Case Diagram: Create a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue browser for queues with text, stream, objects, bytes or map messages.

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.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

Example1

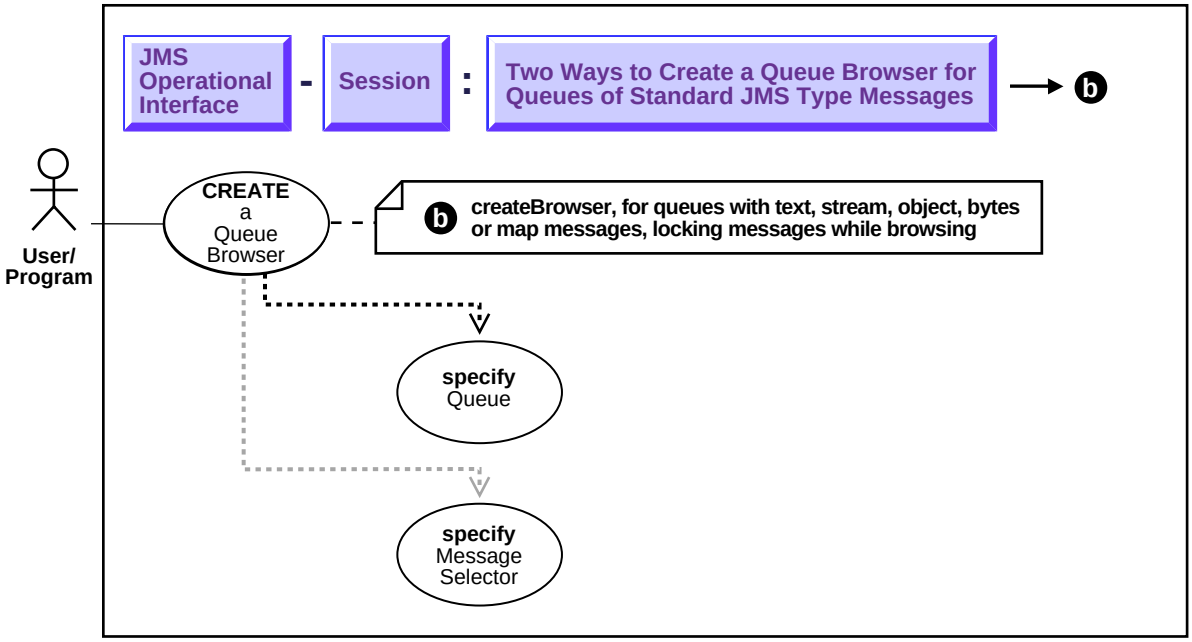
```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;

browser = jms_session.createBrowser(queue);
```

Example2

```
/* Create a browser for queues with a specified selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;

/* create a Browser to look at messages with correlationID = RUSH  */
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");
```

Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages, Locking Messages while Browsing

Figure 14–14 Use Case Diagram: Create a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages, Locking Messages while Browsing



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue browser for queues with text, stream, objects, bytes or map messages, locking messages while browsing.

Usage Notes

If locked parameter is specified as true, messages are locked as they are browsed. Hence these messages cannot be removed by other consumers until the browsing session ends the transaction

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

Example1

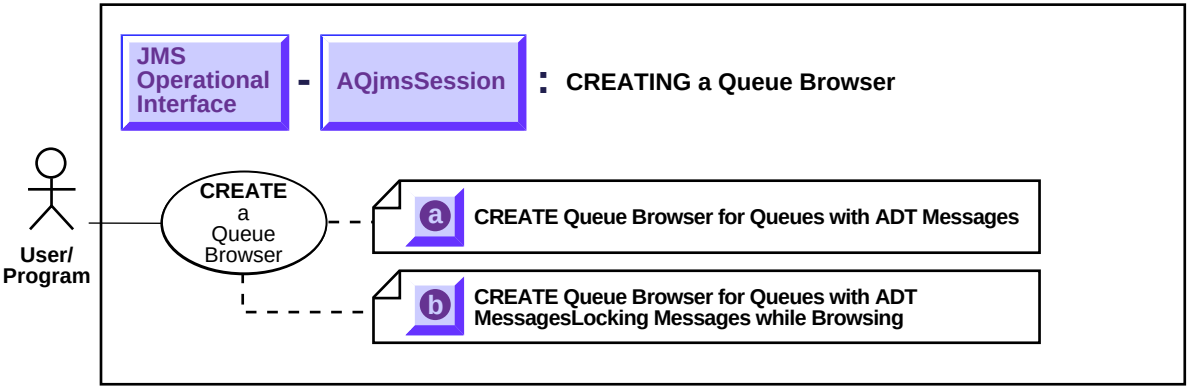
```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;

browser = jms_session.createBrowser(queue, null, true);
```

Example2

```
/* 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);
```

Two Ways of Creating a Queue Browser for Oracle Object Type (ADT) Messages Queues

Figure 14–15 Use Case Diagram: Two Ways to Create a Queue Browser for Queues of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

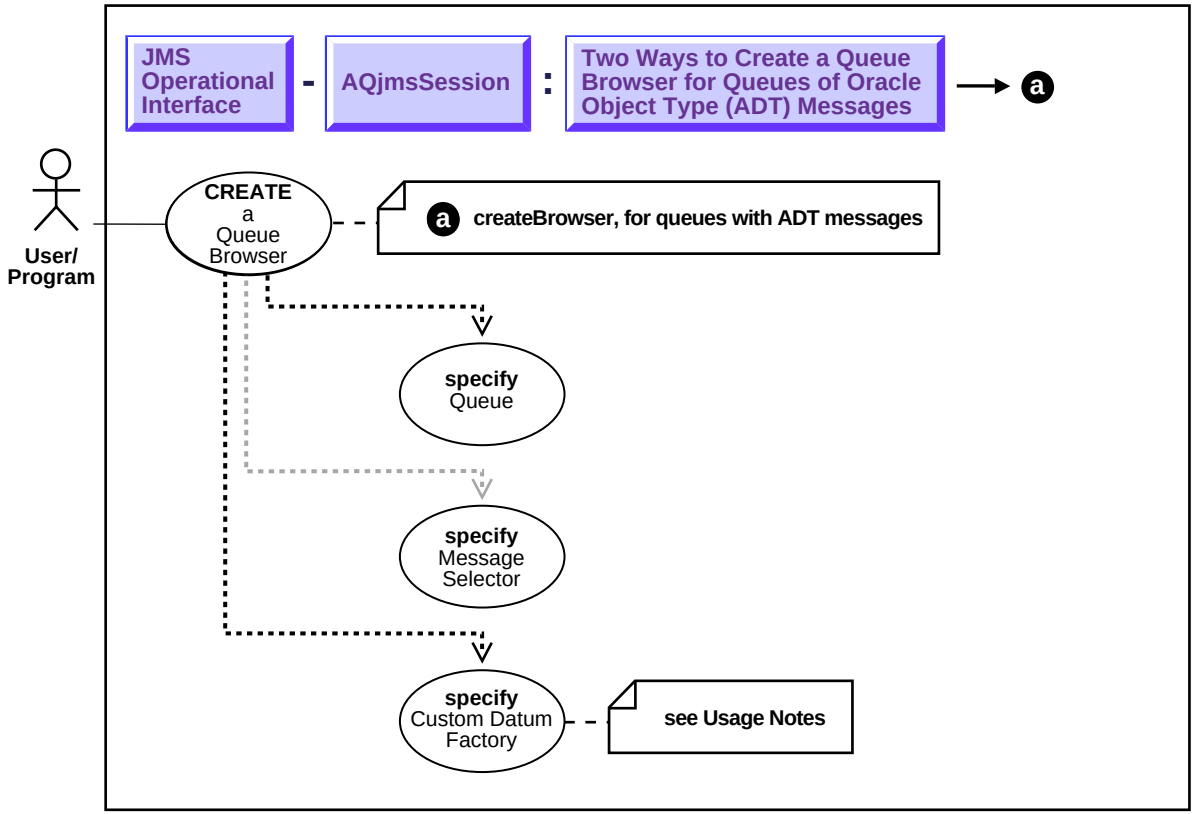
This section explains two ways to create browser for Oracle object type (ADT) messages queues:

a. Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages on page 14-26

b. Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages, Locking Messages While Browsing on page 14-28

Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages

Figure 14–16 Use Case Diagram: Create a Queue Browser for Queues of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue browser for queues of Oracle object type (ADT) messages.

Usage Notes

For queues containing `AdtMessages` the selector for 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`

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

The CustomDatum factory for a particular java class that maps to the SQL ADT payload can be obtained via 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 ADT 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());
```

Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages, Locking Messages While Browsing

To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue browser for queues of Oracle object type (ADT) messages, locking messages while browsing.

Usage Notes

Not applicable.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

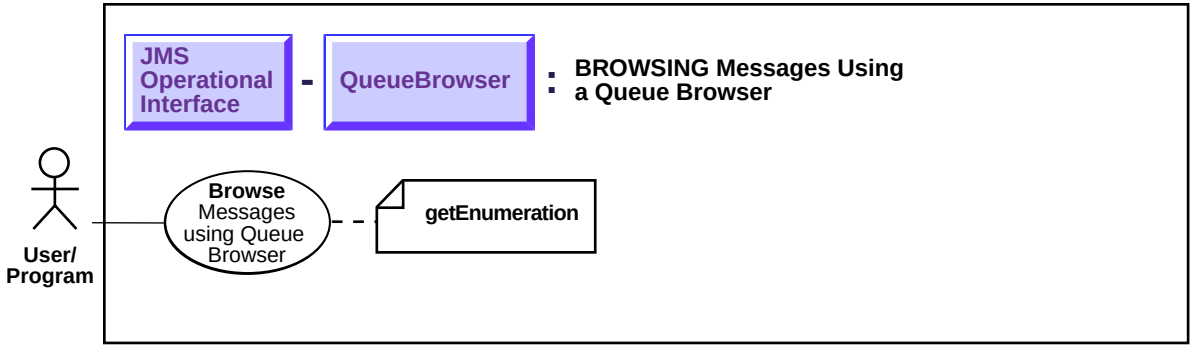
Example

```
 /* 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);
```

Browsing Messages Using a Queue Browser

Figure 14–17 Use Case Diagram: Browsing Messages Using a Queue Browser



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Browse messages using a queue browser.

Usage Notes

Use methods in java.util.Enumeration to go through the list of messages.

Syntax

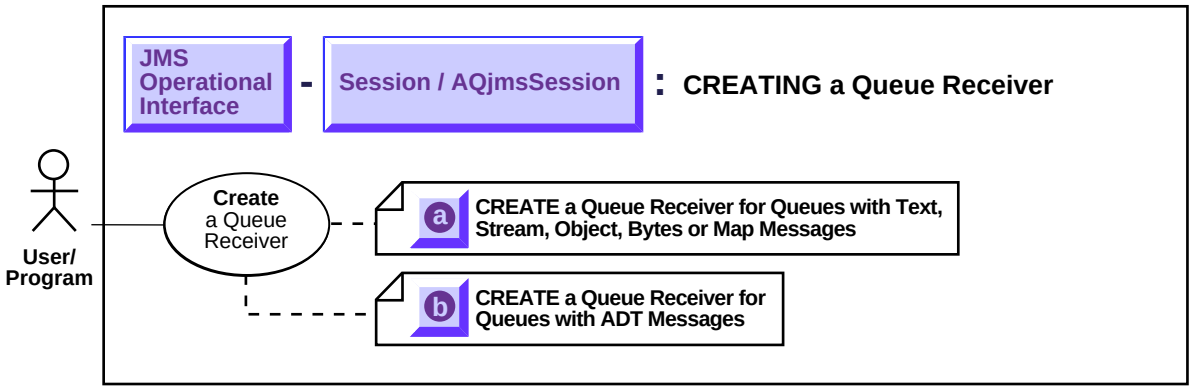
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueBrowser

Example

```
/* Create a browser for queues with a specified selector */
public void browse_rush_orders(QueueSession jms_session)
{
   QueueBrowser    browser;
   Queue           queue;
   ObjectMessage   obj_message
   BolOrder        new_order;
   Enumeration     messages;

   /* get a handle to the new_orders queue */
   queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");

   /* create a Browser to look at RUSH orders */
   browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");

   /* Browse through the messages */
   for (messages = browser.elements() ; message.hasMoreElements() ;)
   {
     obj_message = (ObjectMessage)message.nextElement();
   }
}
```

Two Ways of Creating a Queue Receiver

Figure 14–18 Use Case Diagram: Two Ways to Create a QueueReceiver



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

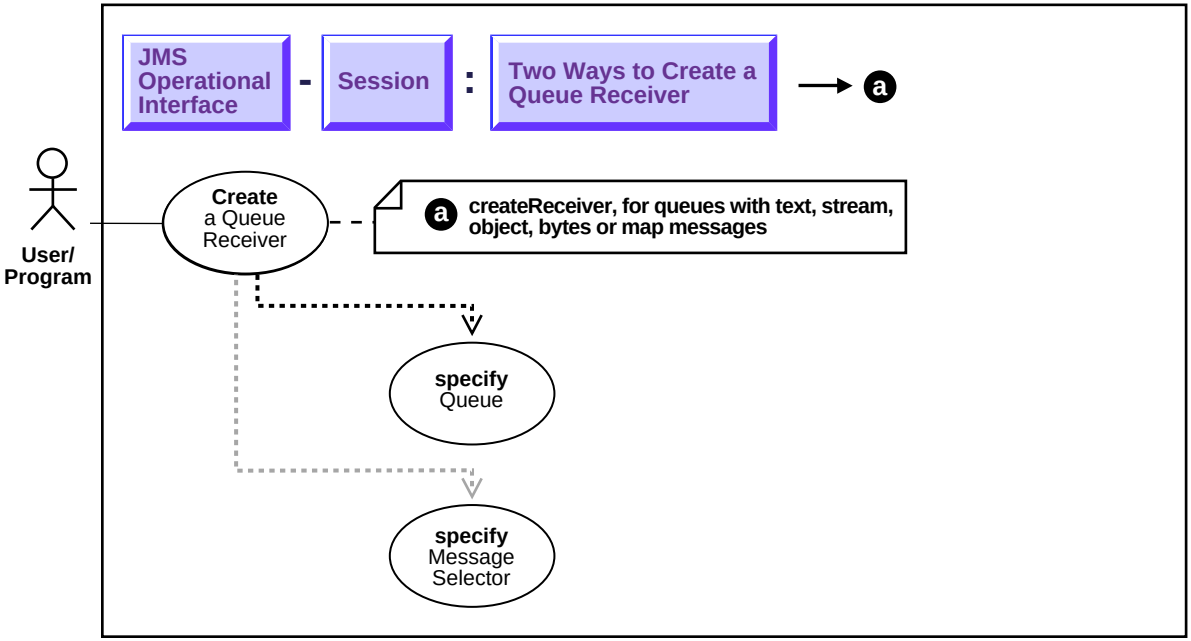
This section explains two ways to create a queue receiver:

a. Creating a Queue Receiver for Queues of Standard JMS Type Messages on page 14-32

b. Creating a Queue Receiver for Queues of Oracle Object Type (ADT) Messages on page 14-34

Creating a Queue Receiver for Queues of Standard JMS Type Messages

Figure 14–19 Use Case Diagram: Create a Queue Receiver for Queues of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue receiver for queues of standard JMS type messages.

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
- 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:"

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createReceiver

Example

Example1

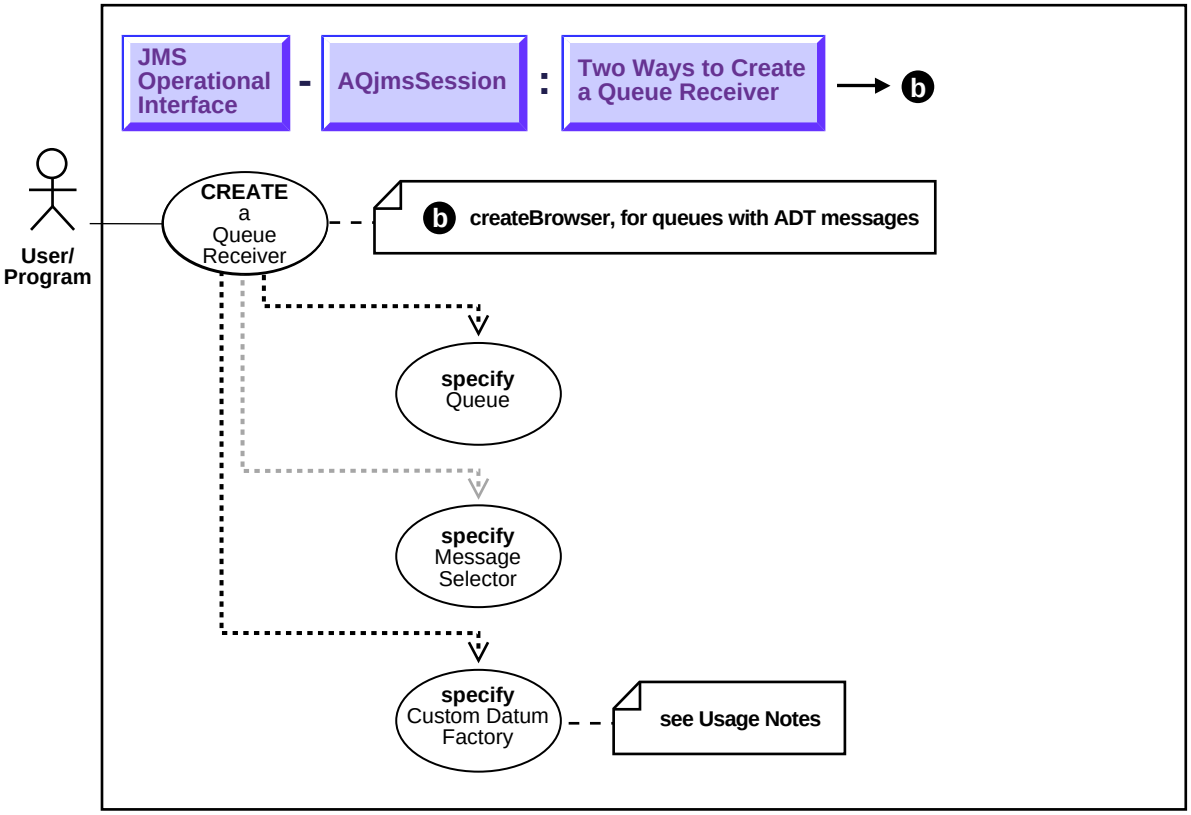
```
/* Create a receiver without a selector */
QueueSession    jms_session
QueueReceiver   receiver;
Queue           queue;

receiver = jms_session.createReceiver(queue);
```

Example2

```
/* Create a receiver for queues with a specified selector */
QueueSession    jms_session;
QueueReceiver   receiver;
Queue           queue;

/* create a Receiver to receive messages with correlationID starting with EXP
*/
browser = jms_session.createReceiver(queue, "JMSCorrelationID LIKE 'EXP%'");
```

Creating a Queue Receiver for Queues of Oracle Object Type (ADT) Messages

Figure 14–20 Use Case Diagram: Create a Queue Receiver for Queues of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue receiver for queues of Oracle object type (ADT) messages.

Usage Notes

The CustomDatum factory for a particular java class that maps to the SQL ADT payload can be obtained via the `getFactory` static method.

For queues containing `AdtMessages` the selector for 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

  `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`

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createReceiver

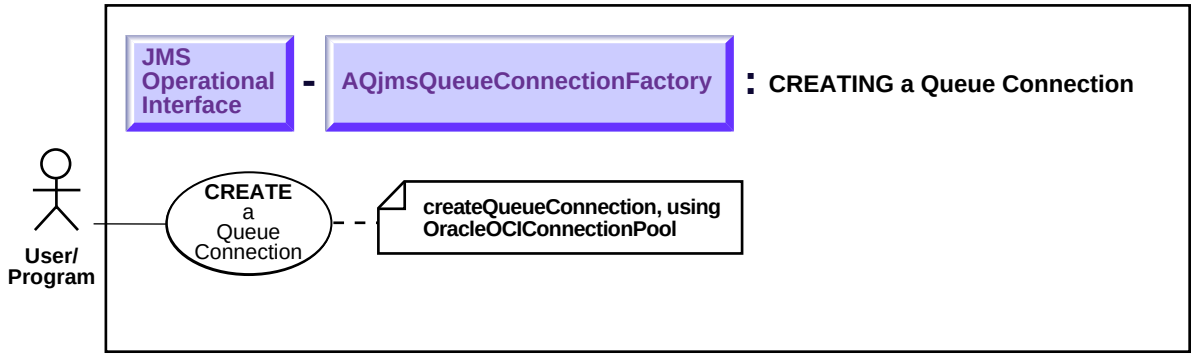
Example

Assume the Queue - test_queue has payload of type SCOTT.EMPLOYEE and the java class that is generated by Jpublisher for this ADT 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());
```

Creating a Queue Connection with an Open OracleOCIConnection Pool

Figure 14–21 Use Case Diagram: Create a Queue Connection with an Open OracleOCIConnectionPool



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a queue connection with an open OracleOCIConnectionPool.

Usage notes

This is a static method.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsQueueConnectionFactory.createQueueConnection

Example

This method may be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS will not open an new `OracleOCIConnectionPool` instance, but instead use the supplied `OracleOCIConnectionPool` instance to create the JMS QueueConnection object.

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(cpool);
Connection db_conn;     /* previously opened JDBC connection */
 QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(db_
conn);
```

15

JMS Operational Interface: Basic Operations (Publish-Subscribe)

In this chapter we describe the operational interface to Oracle Advanced Queuing in terms of use cases. That is, we discuss each operation (such as "Publish a Message") as a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: Operational Interface — Basic Operations" on page 14-2).

A summary figure, "Use Case Diagram: Operational Interface — Basic Operations", locates all the use cases in single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case that interests you by clicking on the relevant use case title.

The individual use cases are themselves laid out as follows:

Each use case is laid out as follows:

- *Use case figure*. A figure that depicts the use case.
- *Purpose*. The purpose of this use case.
- *Usage Notes*. Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment that illustrate the use case.

Use Case Model: JMS Operational Interface — Basic Operations (Publish-Subscribe)

Table 15–1 Use Case Model: Operational Interface — Basic Operations (Publish-Subscribe)

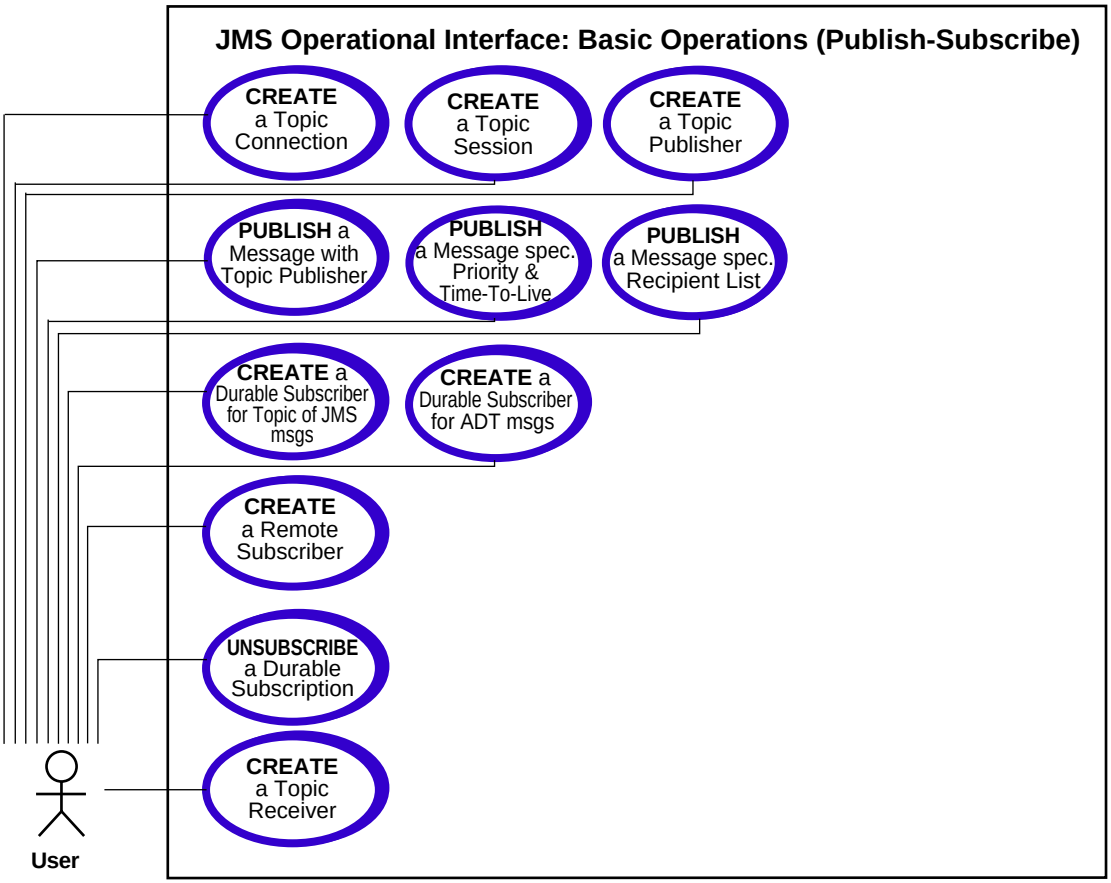
Use Case
Four Ways of Creating a Topic Connection on page 15-4ì
Creating a Topic Connection with Username/Password on page 15-6
Creating a Topic Connection with Open JDBC Connection on page 15-8
Creating a Topic Connection with Default Connection Factory Parameters on page 15-10
Creating a Topic Connection with an Open OracleOCIConnectionPool on page 15-11
Creating a Topic Session on page 15-13
Creating a Topic Publisher on page 15-14
Four Ways of Publishing Messages Using a Topic Publisher on page 15-16
Publishing a Message with Minimal Specification on page 15-17
Publishing a Message Specifying Correlation and Delay on page 15-20
Publishing a Message Specifying Priority and Time-To-Live on page 15-23
Publishing Messages Specifying a Recipient List Overriding Topic Subscribers on page 15-25
Two Ways of Creating a Durable Subscriber for a Topic of Standard JMS Type Messages on page 15-28
Creating a Durable Subscriber for a JMS Topic without Selector on page 15-29
Creating a Durable Subscriber for a JMS Topic with Selector on page 15-31
Two Ways of Creating a Durable Subscriber for a Topic of Oracle Object Type (ADT) Messages on page 15-34
Creating a Durable Subscriber for an ADT Topic without Selector on page 15-35
Creating a Durable Subscriber for an ADT Topic with Selector on page 15-37
Two Ways of Creating a Remote Subscriber on page 15-40
Creating a Remote Subscriber for Topics of JMS Messages on page 15-41
Creating a Remote Subscriber for Topics of Oracle Object Type (ADT) Messages on page 15-44
Two Ways of Unsubscribing a Durable Subscription on page 15-47
Unsubscribing a Durable Subscription for a Local Subscriber on page 15-48
Unsubscribing a Durable Subscription for a Remote Subscriber on page 15-50

Table 15–1 (Cont.) Use Case Model: Operational Interface — Basic Operations (Publish-Subscribe)

Use Case
Two Ways of Creating a Topic Receiver on page 15-52
Creating a Topic Receiver for a Topic of Standard JMS Type Messages on page 15-53
Creating a Topic Receiver for a Topic of Oracle Object Type (ADT) Messages on page 15-55
Two Ways of Creating a Topic Browser for JMS Message Queues on page 15-57
Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages on page 15-58
Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages, Locking Messages While Browsing on page 15-60
Two Ways of Creating a Topic Browser for Oracle Object Type (ADT) Message Topics on page 15-62
Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages on page 15-63
Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages, Locking Messages While Browsing on page 15-66
Browsing Messages Using a Topic Browser on page 15-68

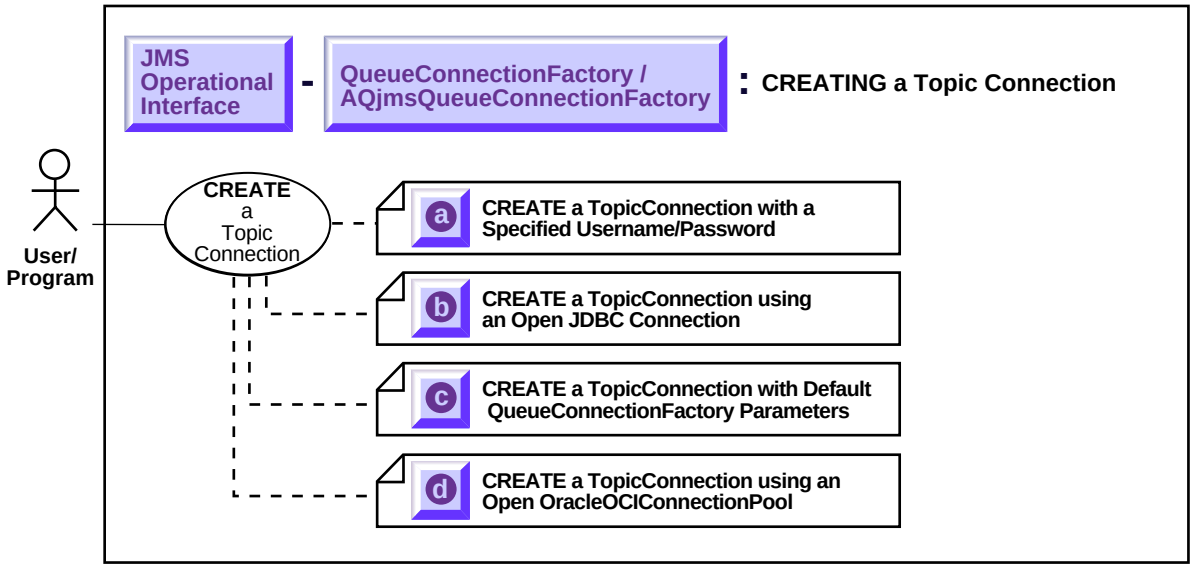
Use Case Model Diagram: Operational Interface — Basic Operations (Publish-Subscribe)

Figure 15–1 Use Case Diagram: Publish-Subscribe - Operational Interface



Four Ways of Creating a Topic Connection

Figure 15–2 Use Case Diagram: Publish-Subscribe - Four Ways to Create a Topic Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

This section explains multiple ways to create a topic connection:

a. Creating a Topic Connection with Username / Password on page 15-6

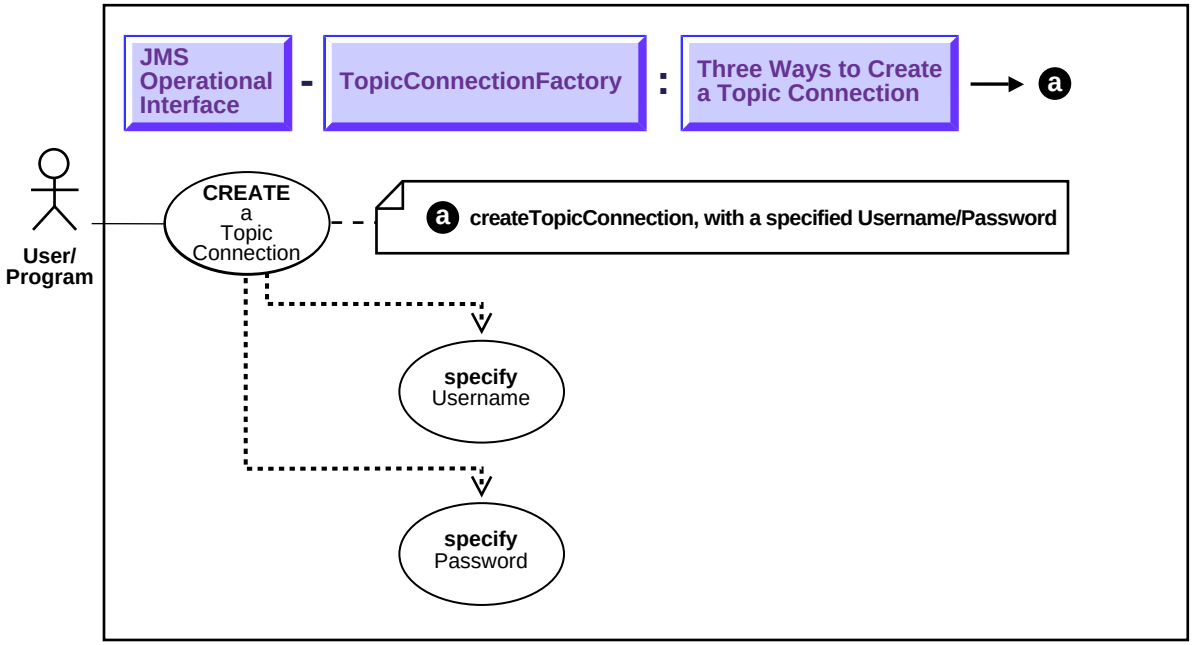
b. Creating a Topic Connection with Open JDBC Connection on page 15-8

c. Creating a Topic Connection with Default Connection Factory Parameters on page 15-10

d. Creating a Topic Connection with an Open OracleOCIConnectionPool on page 15-11

Creating a Topic Connection with Username/Password

Figure 15–3 Use Case Diagram: Publish-Subscribe - Create a Topic Connection with Username/Password



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic connection with username/password

Usage Notes

Not applicable.

Syntax

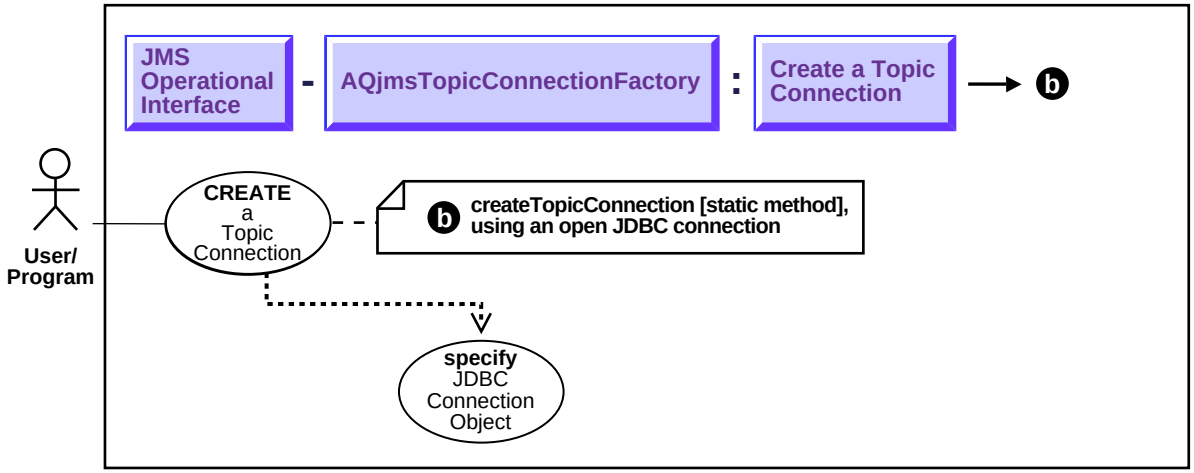
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicConnectionFactory.createTopicConnection

Example

```
 TopicConnectionFactory tc_fact =
AQjmsFactory.getTopicConnectionFactory("sun123", "oratest", 5521, "thin");
/* Create a topic connection using a username/password */
TopicConnection tc_conn = tc_fact.createTopicConnection("jmsuser", "jmsuser");
```

Creating a Topic Connection with Open JDBC Connection

Figure 15–4 Use Case Diagram: Publish-Subscribe - Create a Topic Connection with Open JDBC Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic connection with open JDBC connection.

Usage Notes

Not applicable.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicConnectionFactory.createTopicConnection

Example

Example 1

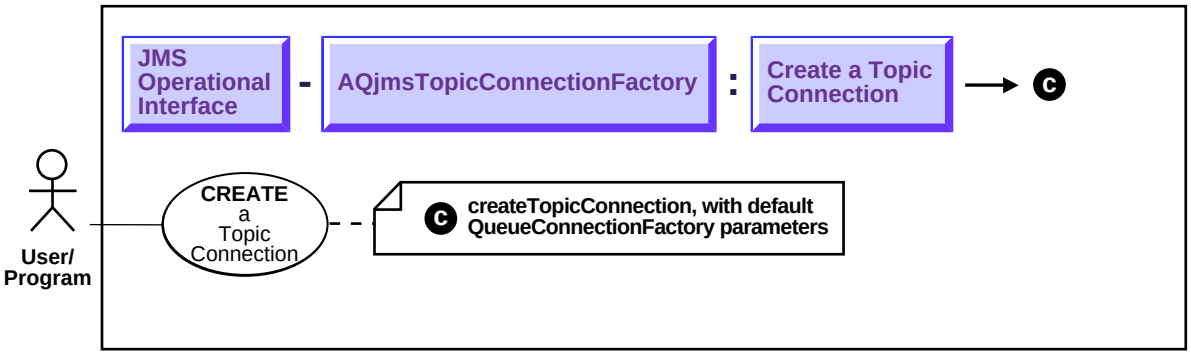
```
Connection db_conn;   /*previously opened JDBC connection */
TopicConnection tc_conn = AQjmsTopicConnectionFactory.createTopicConnection(db_
conn);
```

Example 2

```
OracleDriver ora = new OracleDriver();
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(ora.defaultConnection());
```

Creating a Topic Connection with Default Connection Factory Parameters

Figure 15–5 Use Case Diagram: Publish-Subscribe - Create a Topic Connection with Default Connection Factory Parameters



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic connection with default connection factory parameters.

Usage Notes

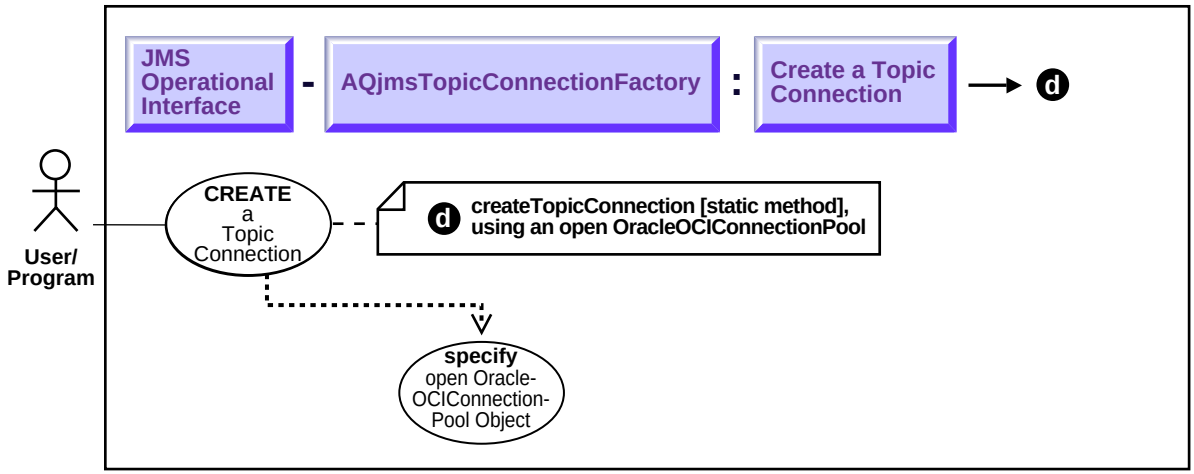
Not applicable.

Syntax

- Java (JDBC): *X*,oracle.jms, AQjmsTopicConnectionFactory.createTopicConnection

Creating a Topic Connection with an Open OracleOCIConnectionPool

Figure 15–6 Use Case Diagram: Publish-Subscribe - Create a Topic Connection with an Open OracleOCIConnectionPool



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic connection with an open `OracleOCIConnectionPool`.

Usage notes

This is a static method.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicConnectionFactory.createTopicConnection

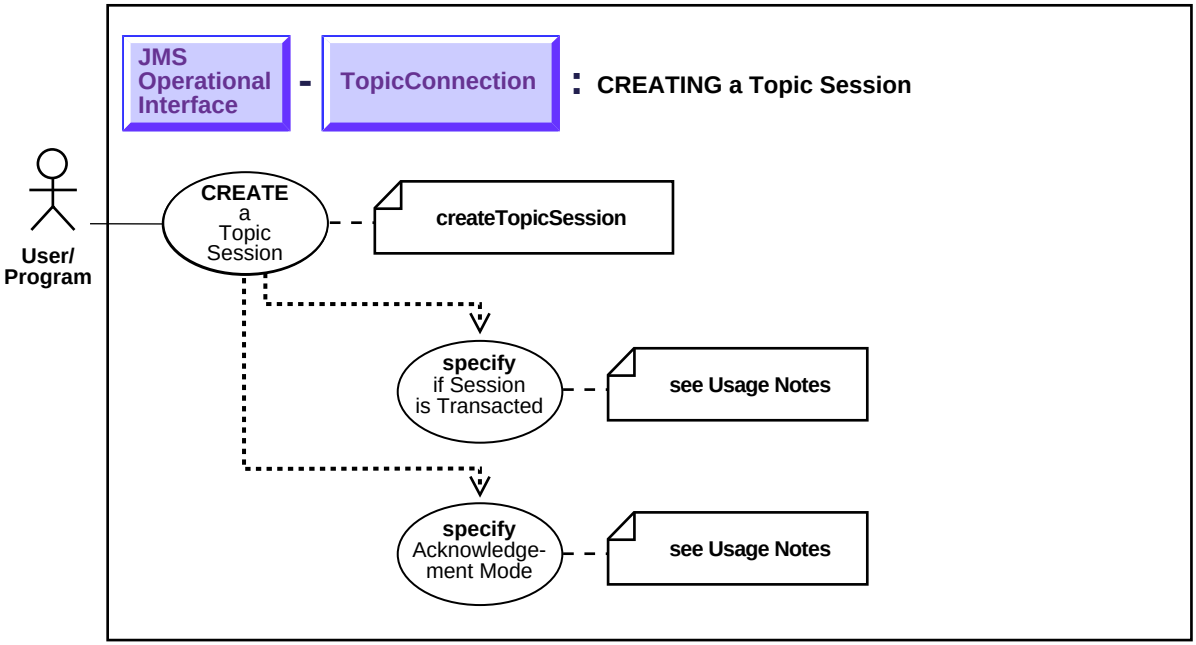
Example

This method may be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS will 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 Topic Session

Figure 15–7 Use Case Diagram: Publish-Subscribe - Create a Topic Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic session.

Usage Notes

Not applicable.

Syntax

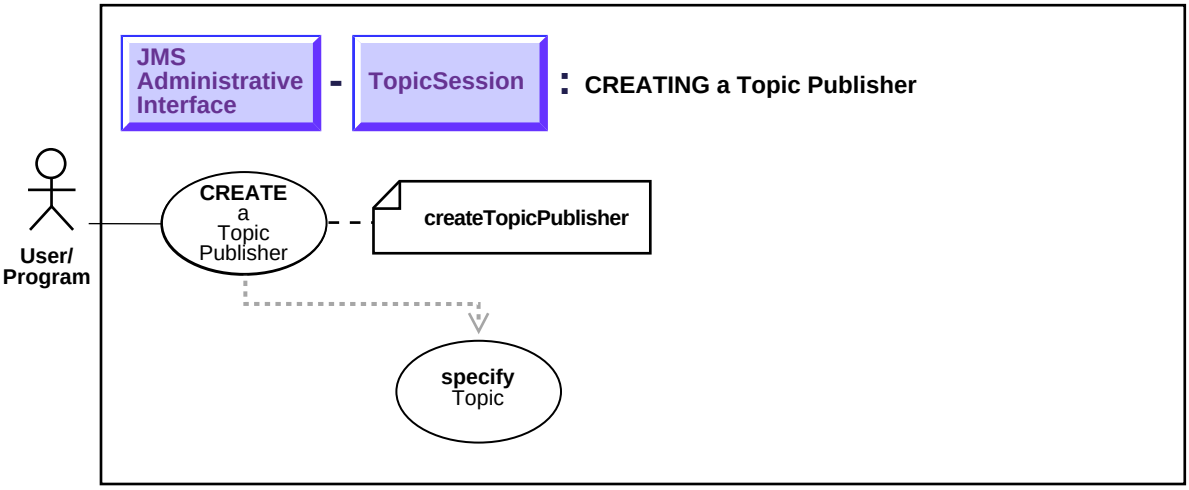
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsConnection.createTopicSession

Example

```
TopicConnection tc_conn;
TopicSession t_sess = tc_conn.createTopicSession(true,0);
```

Creating a Topic Publisher

Figure 15–8 Use Case Diagram: Publish-Subscribe - Create a Topic Publisher



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic publisher.

Usage Notes

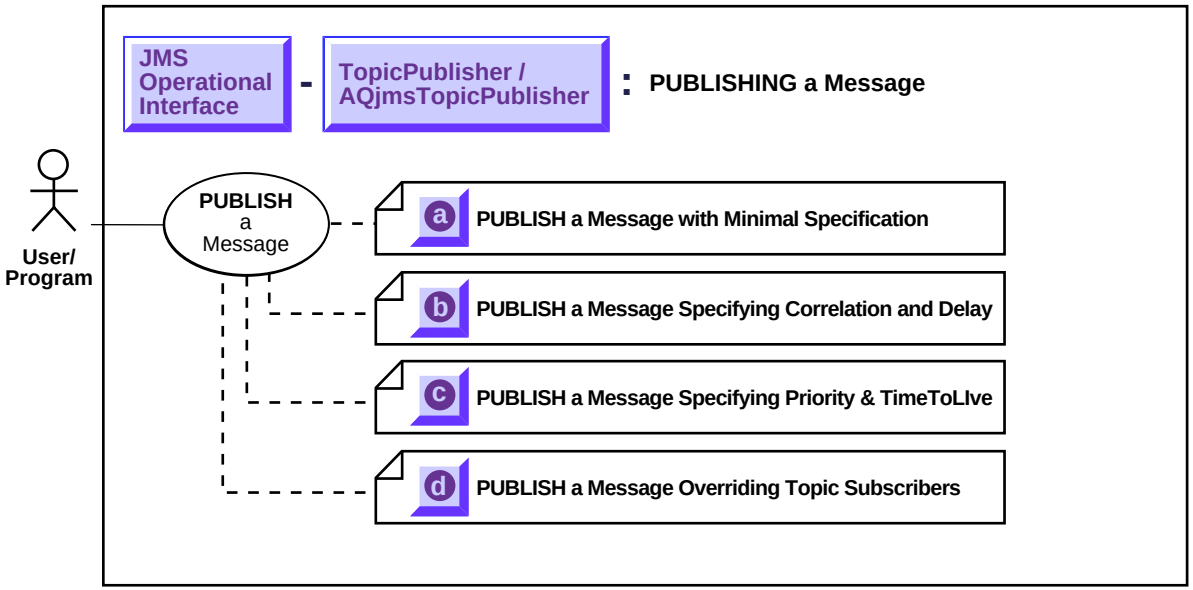
Not applicable.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference,*oracle.jms, AQjmsSession.createPublisher

Four Ways of Publishing Messages Using a Topic Publisher

Figure 15–9 Use Case Diagram: Publish-Subscribe - Four Ways to Publish Messages Using a Topic Publisher



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

This section explains four ways to publish messages using a topic publisher:

a. Publishing a Message with Minimal Specification on page 15-17

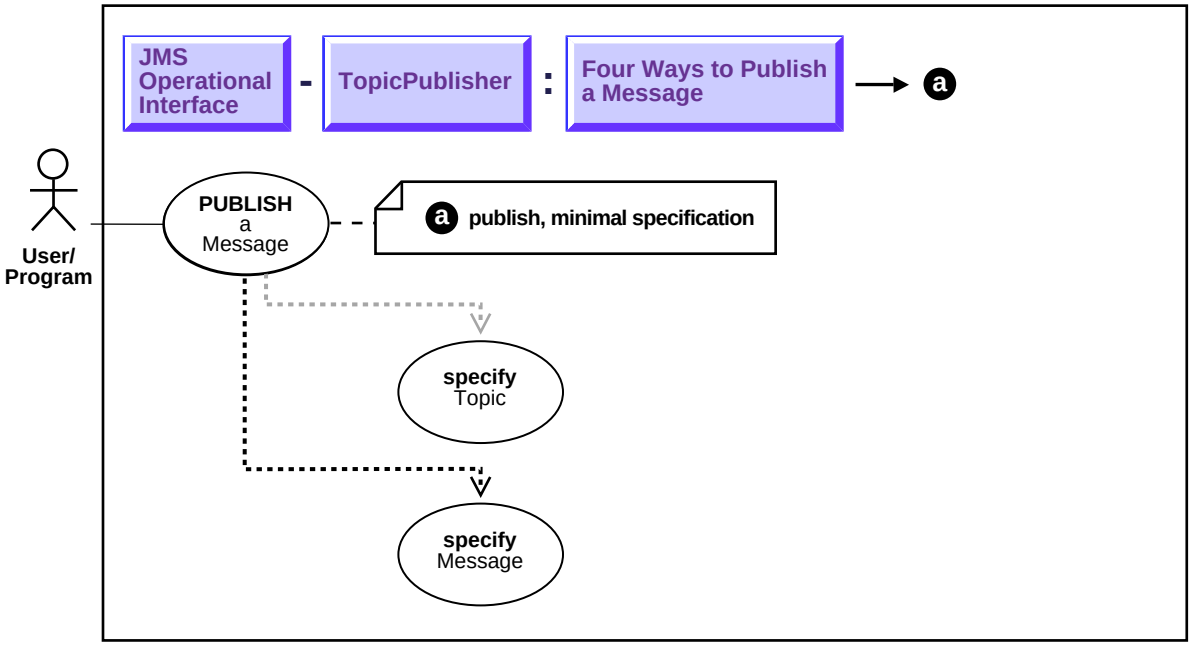
b. Publishing a Message Specifying Correlation and Delay on page 15-20

c. Publishing a Message Specifying Priority and Time-To-Live on page 15-23

d. Publishing Messages Specifying a Recipient List Overriding Topic Subscribers on page 15-25

Publishing a Message with Minimal Specification

Figure 15–10 Use Case Diagram: Publish-Subscribe - Publish a Message with Minimal Specification



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Publish a message with minimal specification.

Usage Notes

If the Topic Publisher 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 will override 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).

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsTopicPublisher.publish

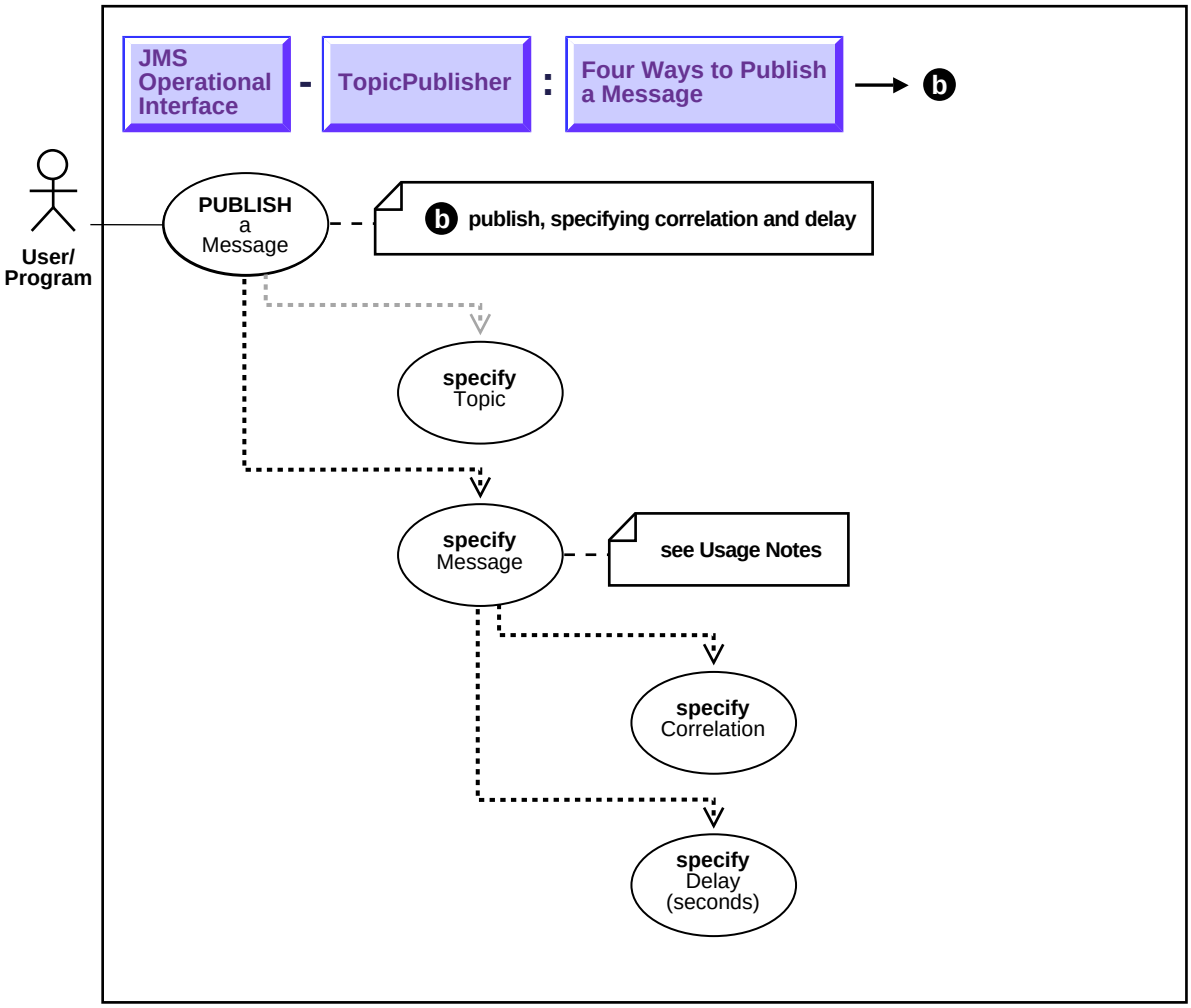
Example

```
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 topic publisher */
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 topic session */
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);
```

Publishing a Message Specifying Correlation and Delay

Figure 15–11 Use Case Diagram: Publish-Subscribe - Publish a Message Specifying Correlation and Delay



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Publish a message specifying correlation and delay.

Usage Notes

The publisher can set the message properties like delay and correlation before publishing.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicPublisher.publish()

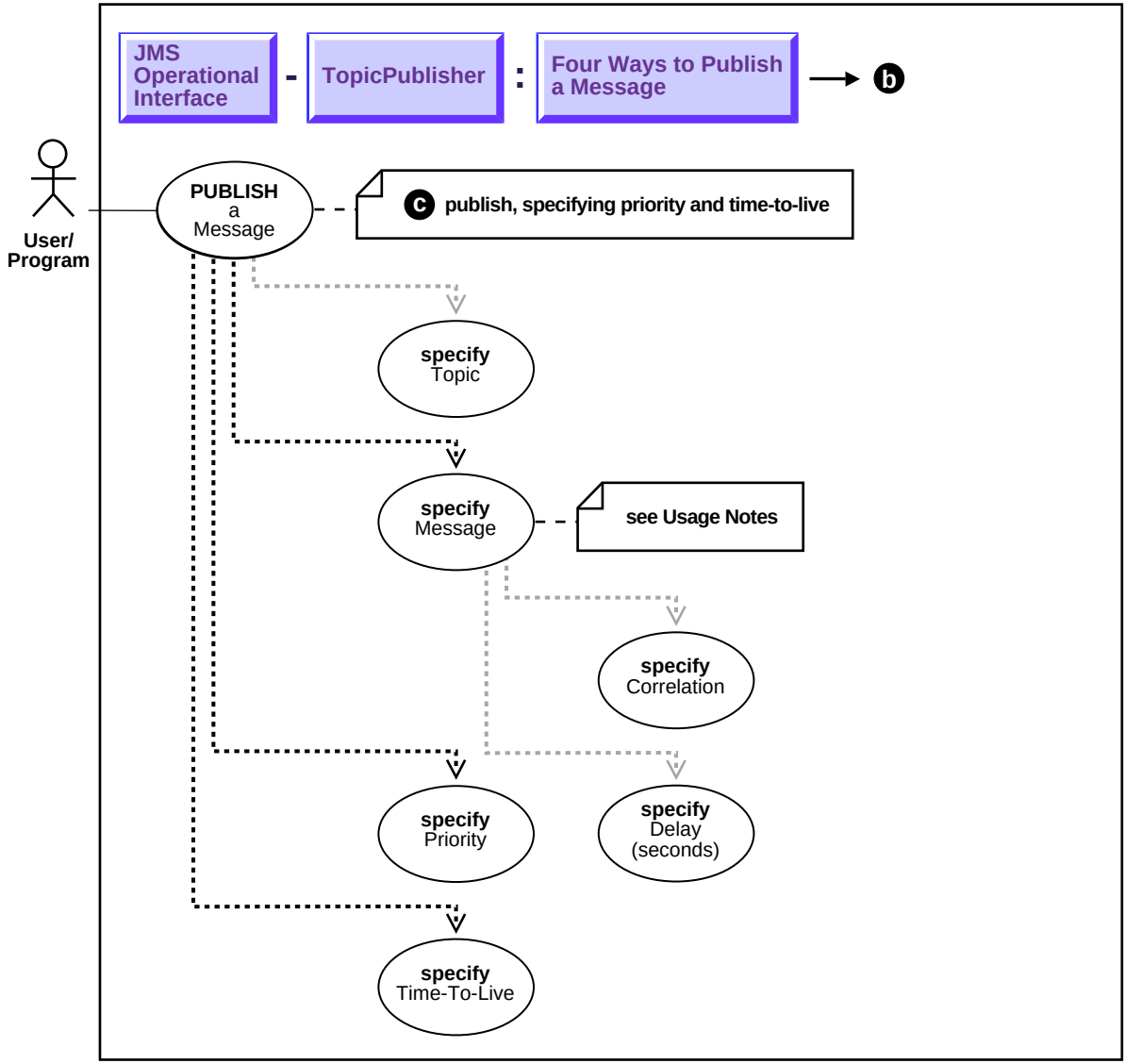
Example

```
 Example 1 - publish specifying delay, correlation

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);
```

Publishing a Message Specifying Priority and Time-To-Live

Figure 15–12 Use Case Diagram: Publish-Subscribe - Publish Messages Specifying Priority and Time-To-Live



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Publish a message specifying priority and time-to-live.

Usage Notes

The priority, and timeToLive of the message can be specified with the publish call. The only delivery mode supported for this release is PERSISTENT.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicPublisher.publish

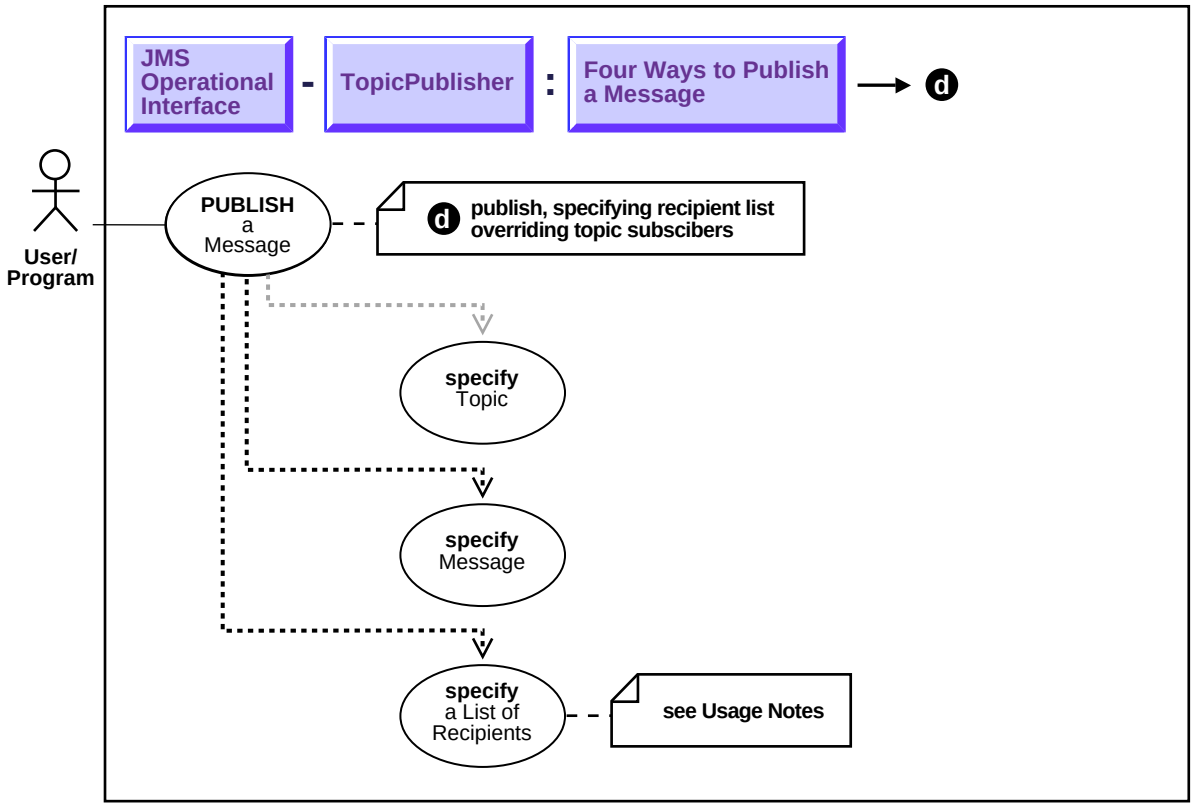
Example

```
 Example 1 - publish specifying priority, timeToLive

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);
```

Publishing Messages Specifying a Recipient List Overriding Topic Subscribers

Figure 15–13 Use Case Diagram: Publish-Subscribe - Publish A Messages Specifying a Recipient List Overriding Topic Subscribers



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Publish a messages specifying a recipient list overriding topic subscribers.

Usage Notes

The subscription list of the topic can be overridden by specifying the recipient list with the publish call.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsTopicPublisher.publish

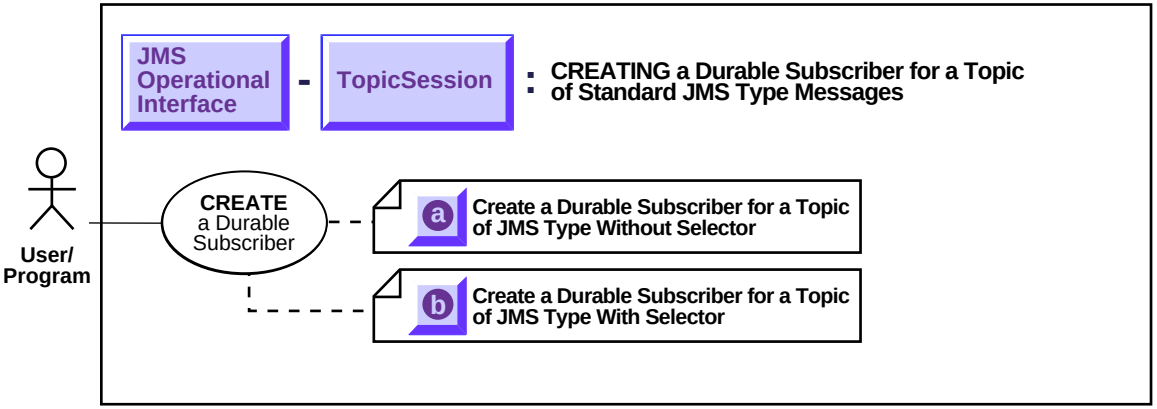
Example

```
 Example 1 - publish specifying priority, 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);
```

Two Ways of Creating a Durable Subscriber for a Topic of Standard JMS Type Messages

Figure 15–14 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Durable Subscriber for a Topic of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

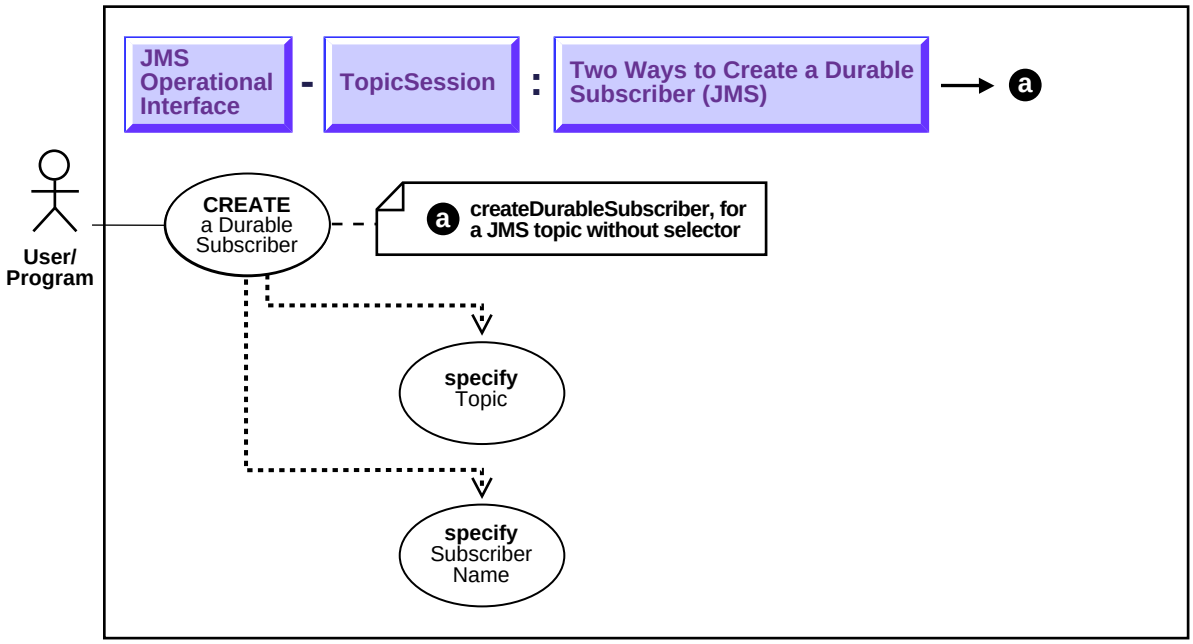
This section explains two ways to create a durable subscriber for a topic of standard JMS type messages:

a. Creating a Durable Subscriber for a JMS Topic without Selector on page 15-29

b. Creating a Durable Subscriber for a JMS Topic with Selector on page 15-31

Creating a Durable Subscriber for a JMS Topic without Selector

Figure 15–15 Use Case Diagram: Publish-Subscribe - Create a Durable Subscriber for JMS Topic without Selector



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a durable subscriber for a JMS topic without selector.

Usage Notes

The subscriber name and JMS topic need to be specified to create a durable subscriber. An unsubscribe call is needed to end the subscription to the topic.

Syntax

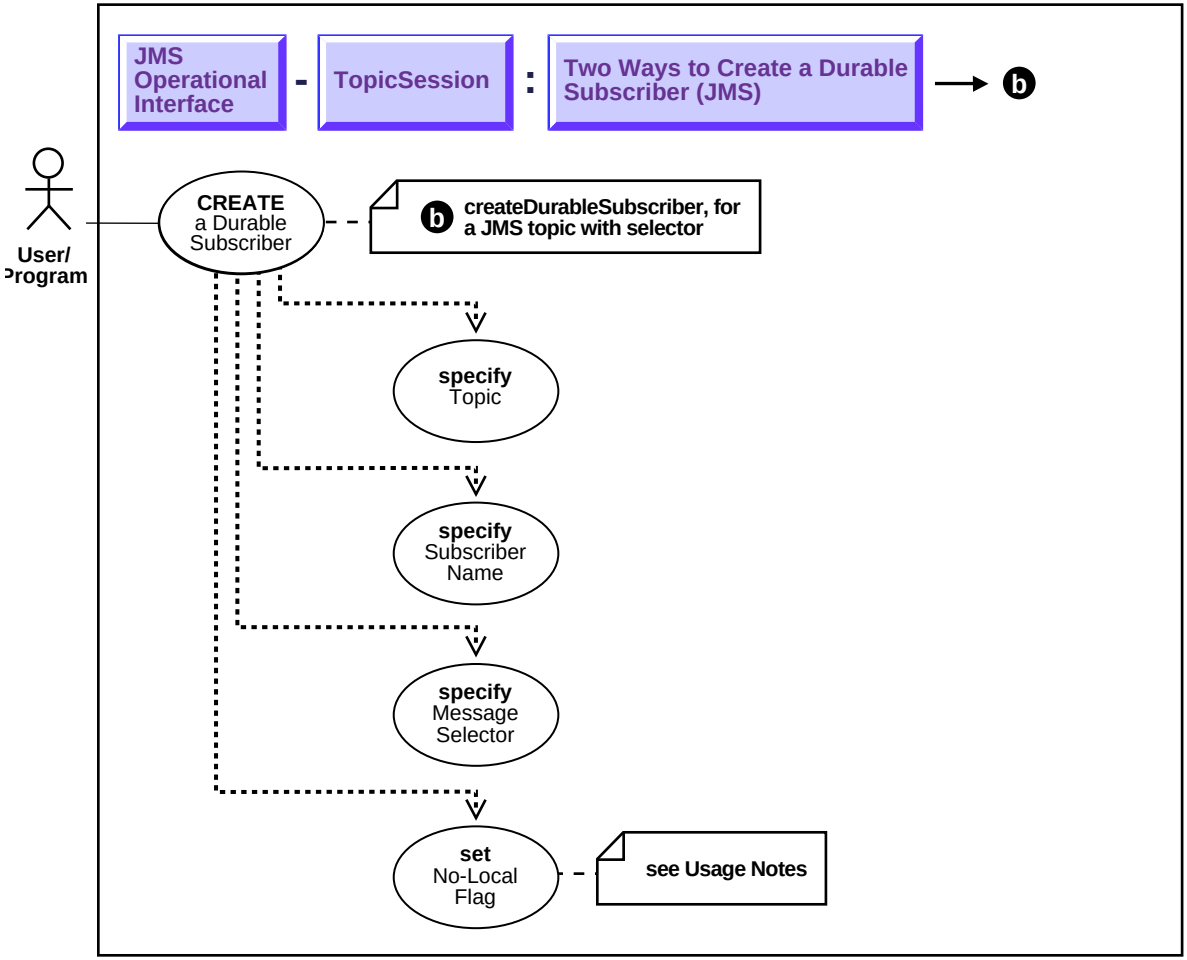
Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.CreateDurableSubscriber

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

Figure 15–16 Use Case Diagram: Publish-Subscribe - Create a Durable Subscriber for a JMS Topic with Selector



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a durable subscriber for a jms topic with selector.

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 may be null. The selector can contain any SQL92 expression that has a combination of one or more of the following:

- JMS Message header fields or properties: JMSPriority (int), JMSCorrelationID (string), JMSType (string), JMSXUserID (string), JMSXAppID (string), JMSXGroupID (string) JMSXGroupSeq (int)

  For example:

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

- 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.

Syntax

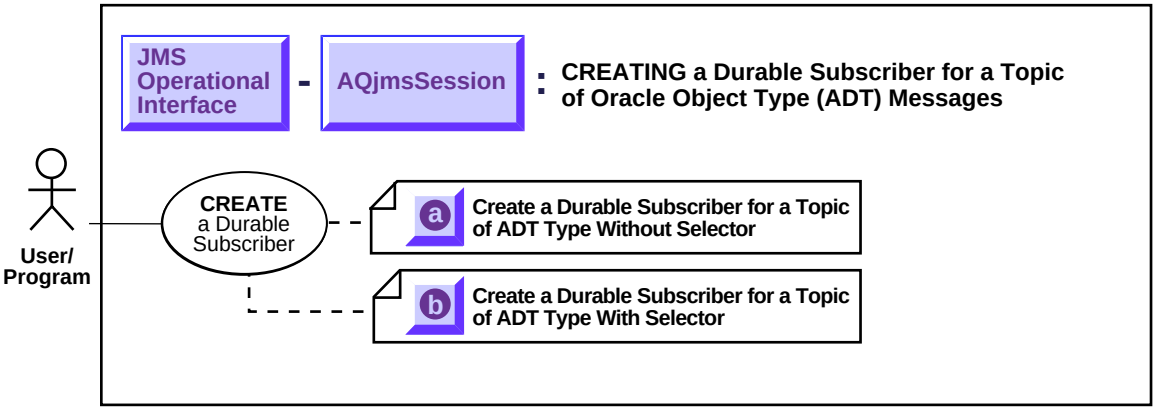
- Java (JDBC): *Oracle9i Supplied Java Packages Reference,*oracle.jms, AQjmsTopicPublisher.publish

Example

```
Example 1 - subscribe specifying selector

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);
```

Two Ways of Creating a Durable Subscriber for a Topic of Oracle Object Type (ADT) Messages

Figure 15–17 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Durable Subscriber for a Topic of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

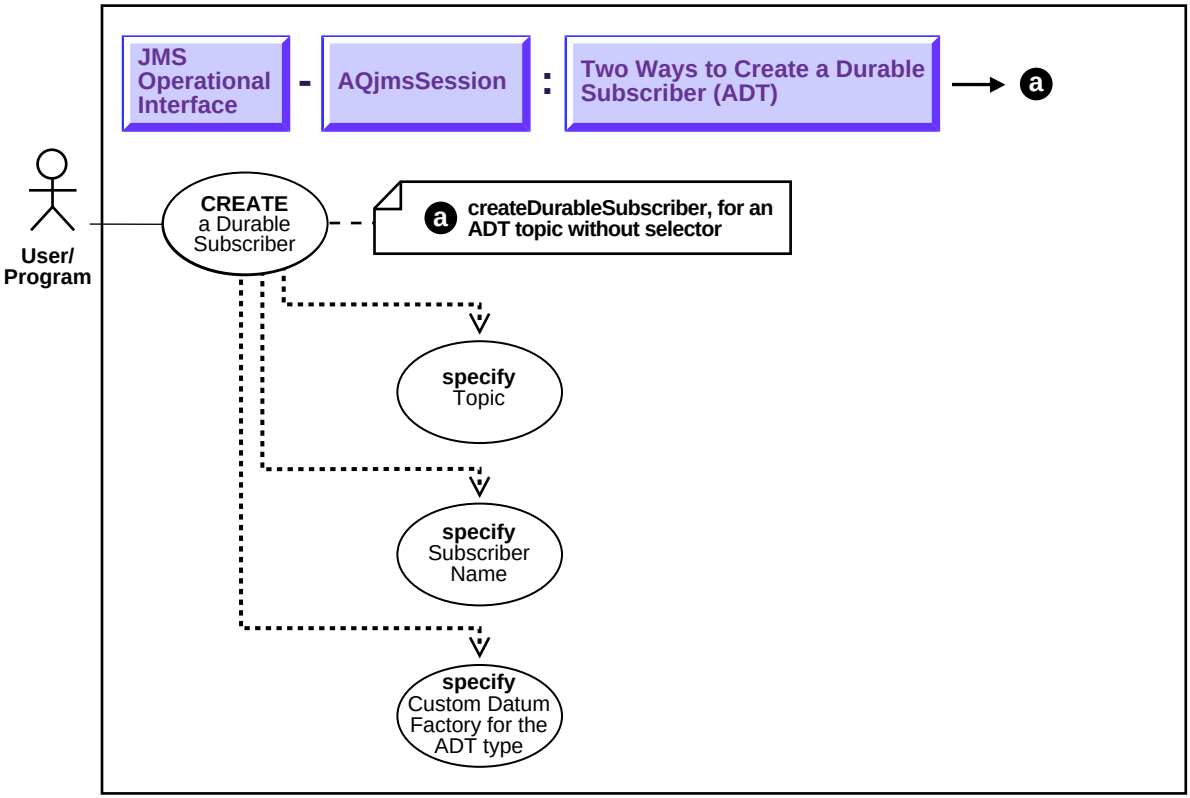
This section explains two ways to create a durable subscriber for a topic of oracle object type (ADT) messages

a. Creating a Durable Subscriber for an ADT Topic without Selector on page 15-35

b. Creating a Durable Subscriber for an ADT Topic with Selector on page 15-37

Creating a Durable Subscriber for an ADT Topic without Selector

Figure 15–18 Use Case Diagram: Publish-Subscribe - Create a Durable Subscriber for an ADT Topic without Selector



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a durable subscriber for an adt topic without selector.

Usage Notes

To create a durable subscriber for a Topic of Oracle Object type, the client needs to specify the CustomDatumFactory for the Oracle Object Type in addition to the Topic and subscriber name.

Syntax

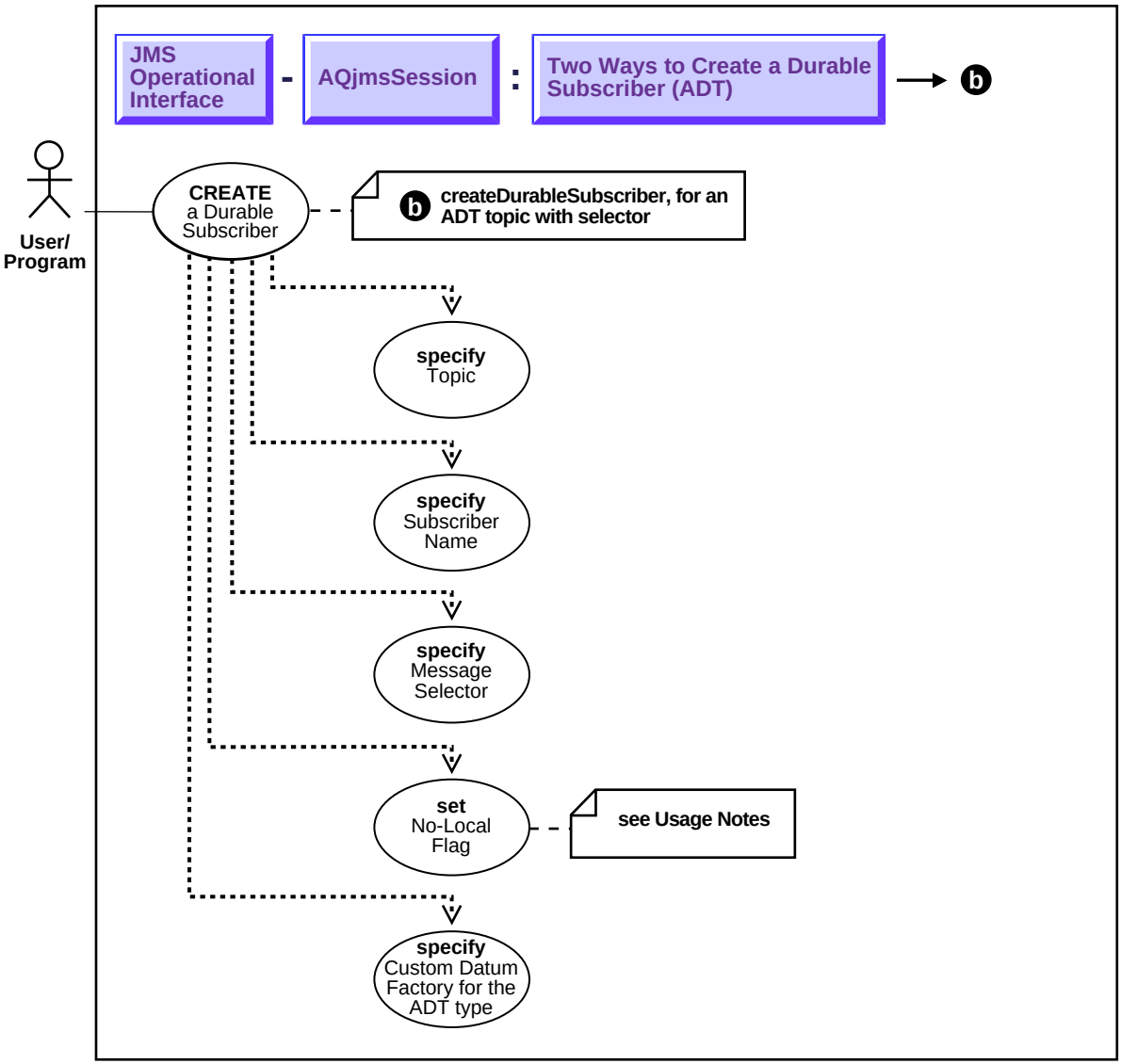
- Java (JDBC): *Oracle9i Supplied Java Packages Reference,*oracle.jms, AQjmsSession.createDurableSubscriber

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 ADT Topic with Selector

Figure 15–19 Use Case Diagram: Publish-Subscribe - Create a Durable Subscriber for an ADT Topic with Selector



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a durable subscriber for an adt topic with selector.

Usage Notes

To create a durable subscriber for a Topic of Oracle Object type, the client needs to specify the CustomDatumFactory for the Oracle Object Type in addition to the Topic and subscriber name.

Optionally, a message selector may be specified. Only messages matching the selector will be delivered to the subscriber.

ADT 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 ADT 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 AQ rules syntax

a. Selector on priority or correlation is specified as follows

For example.:- priority > 3 AND corrid = 'Fiction'

b. Selector on message payload is specified as follows. The attribute

name must be prefixed with tab.user_data.

For example:-

tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000

Syntax

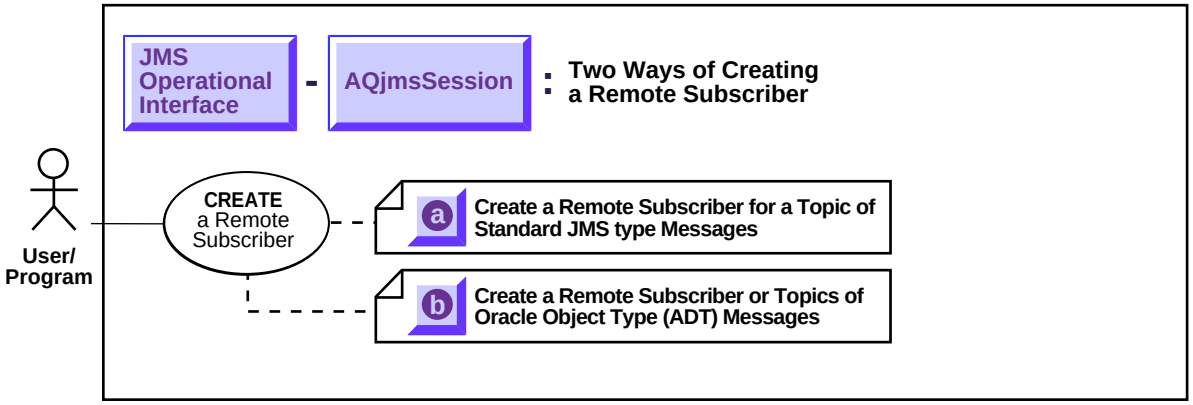
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.createDurableSubscriber

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 the correct CustomDatumFactory  and selector
*/
subscriber1 = jms_sess.createDurableSubscriber(shipped_orders,
"WesternShipping", " priority > 1 and tab.user_data.region like 'WESTERN %'",
false, ADTMessage.getFactory());
```

Two Ways of Creating a Remote Subscriber

Figure 15–20 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Remote Subscriber



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

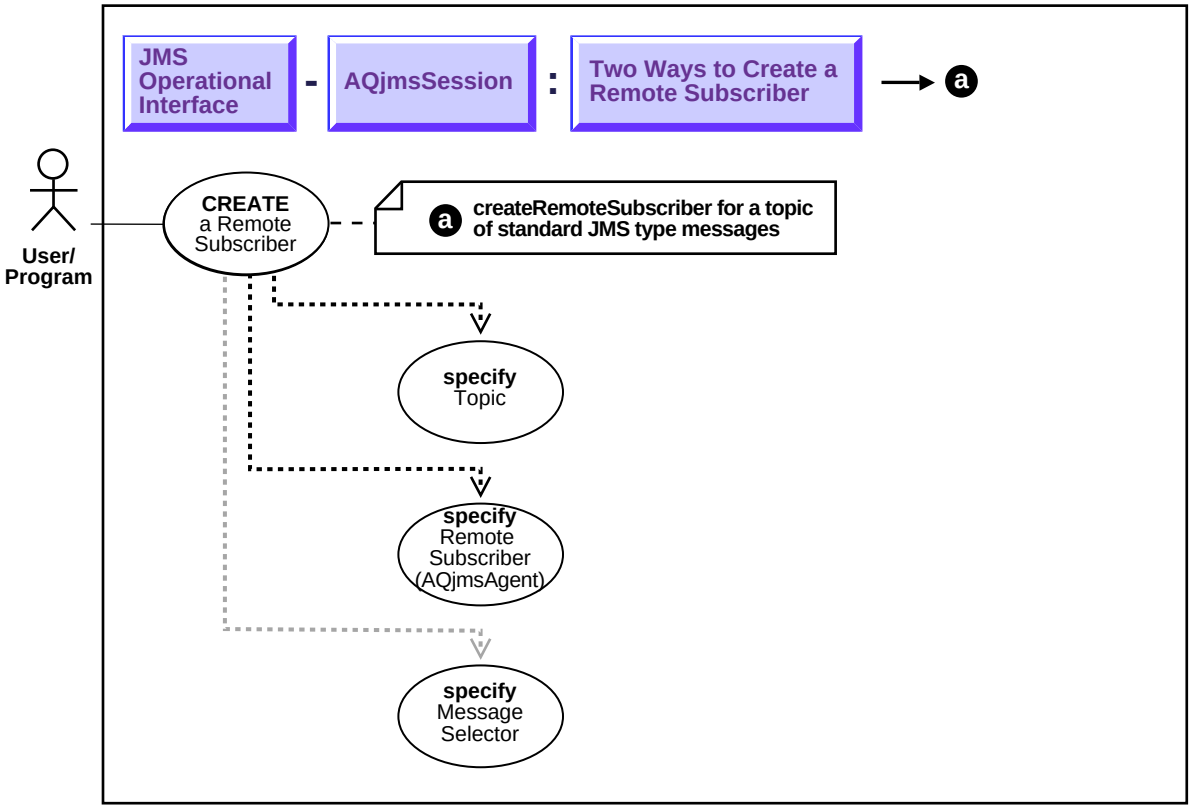
This section explains two ways to create a subscriber for topics of standard JMS type messages

a. Creating a Remote Subscriber for Topics of JMS Messages on page 15-41

b. Creating a Remote Subscriber for Topics of Oracle Object Type (ADT) Messages on page 15-44

Creating a Remote Subscriber for Topics of JMS Messages

Figure 15–21 Use Case Diagram: Publish-Subscribe - Create a Remote Subscriber for Topics of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a remote subscriber for topics of jms messages without selector.

Usage Notes

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 may 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.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference,*oracle.jms, AQjmsSession.createRemoteSubscriber

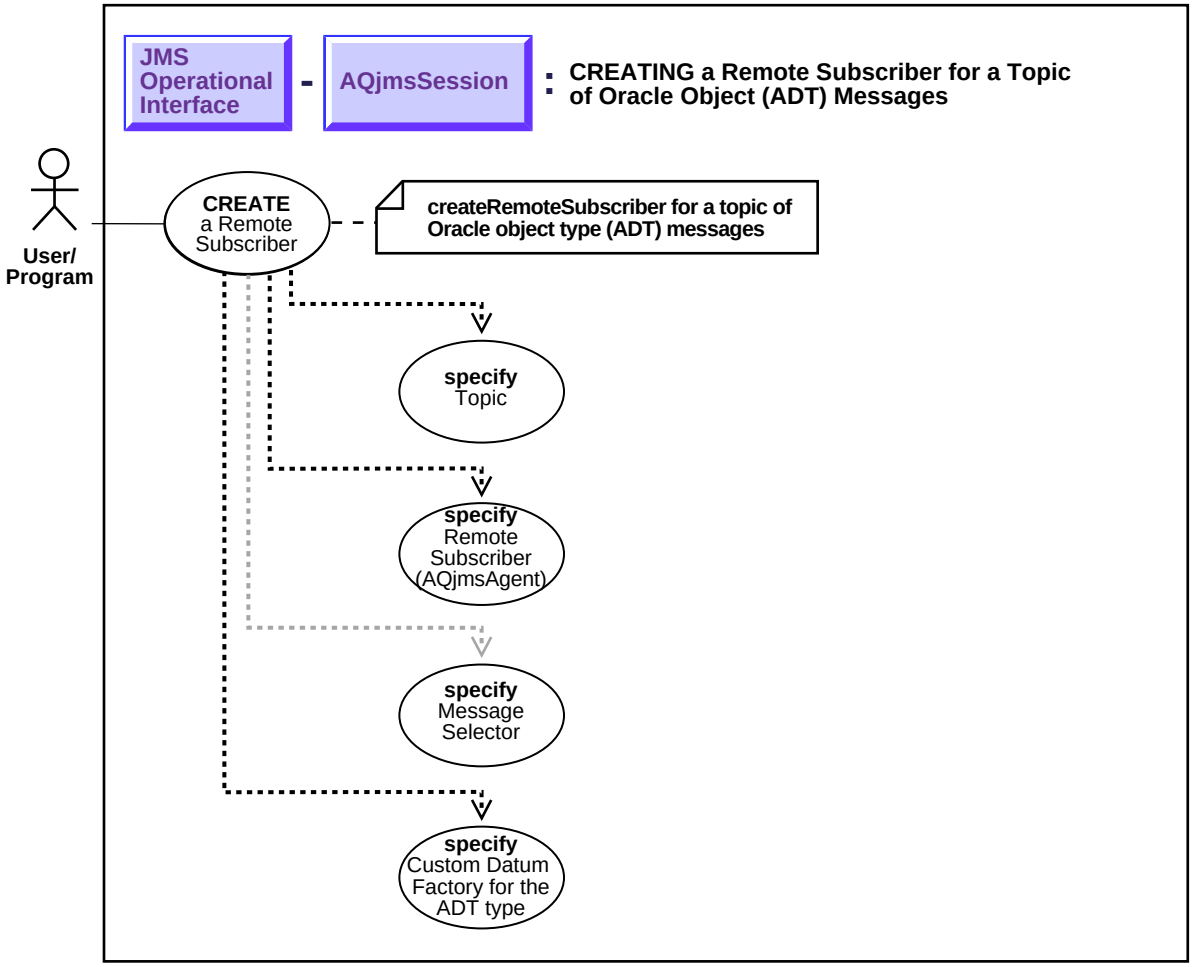
Example

```
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 (ADT) Messages

Figure 15–22 Use Case Diagram: Publish-Subscribe - Create a Remote Subscriber for Topics of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a remote subscriber for topics of oracle object type (adt) messages.

Usage Notes

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 may 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 that same as that for createDurableSubscriber with Topics of ADT type messages. The message selector may be null.

Syntax

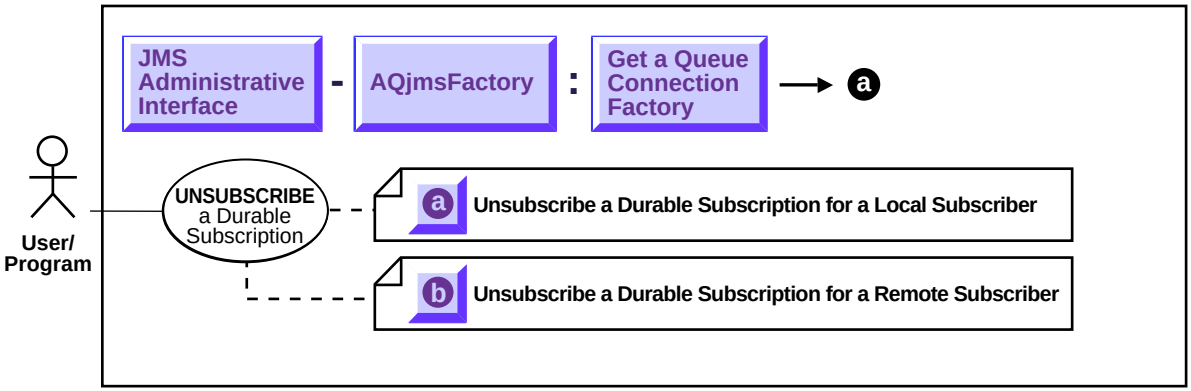
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.createRemoteSubscriber

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 Topic session */
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);
```

Two Ways of Unsubscribing a Durable Subscription

Figure 15–23 Use Case Diagram: Publish-Subscribe - Two Ways to Unsubscribe a Durable Subscription



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

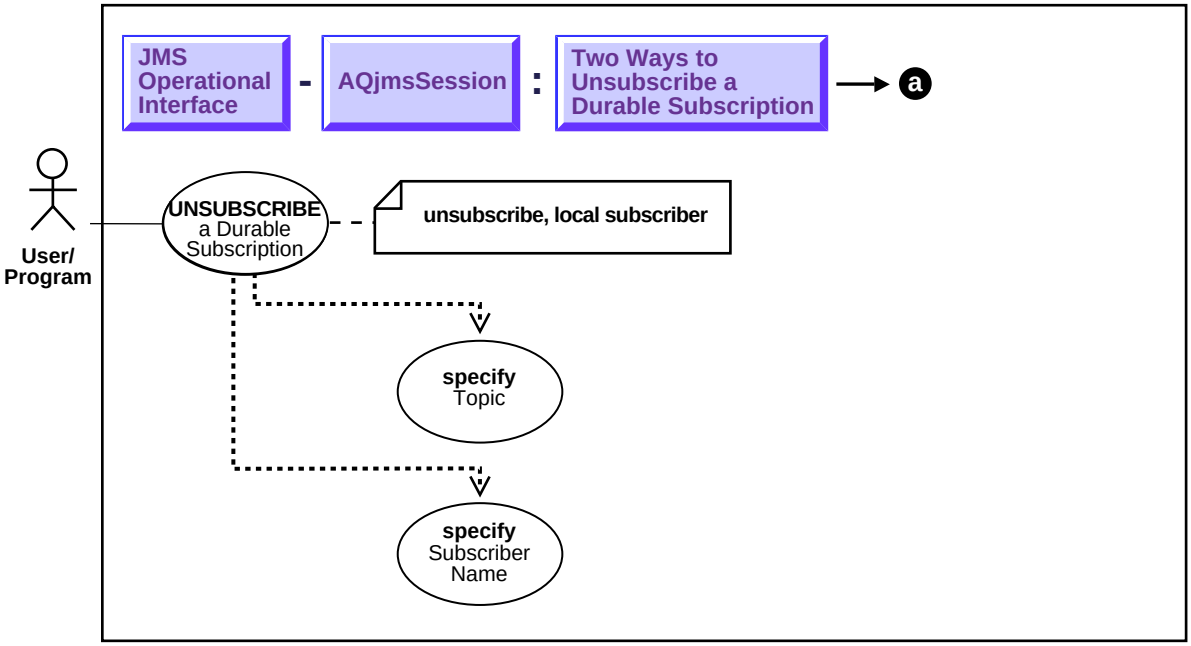
This section explains two ways to unsubscribe a durable subscription.

a. Unsubscribing a Durable Subscription for a Local Subscriber on page 15-48

b. Unsubscribing a Durable Subscription for a Remote Subscriber on page 15-50

Unsubscribing a Durable Subscription for a Local Subscriber

Figure 15–24 Use Case Diagram: Publish-Subscribe - Unsubscribe a Durable Subscription for a Local Subscriber



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Unsubscribe a durable subscription for a local subscriber.

Usage Notes

Unsubscribe a durable subscription that has been created by a client on the specified topic.

Syntax

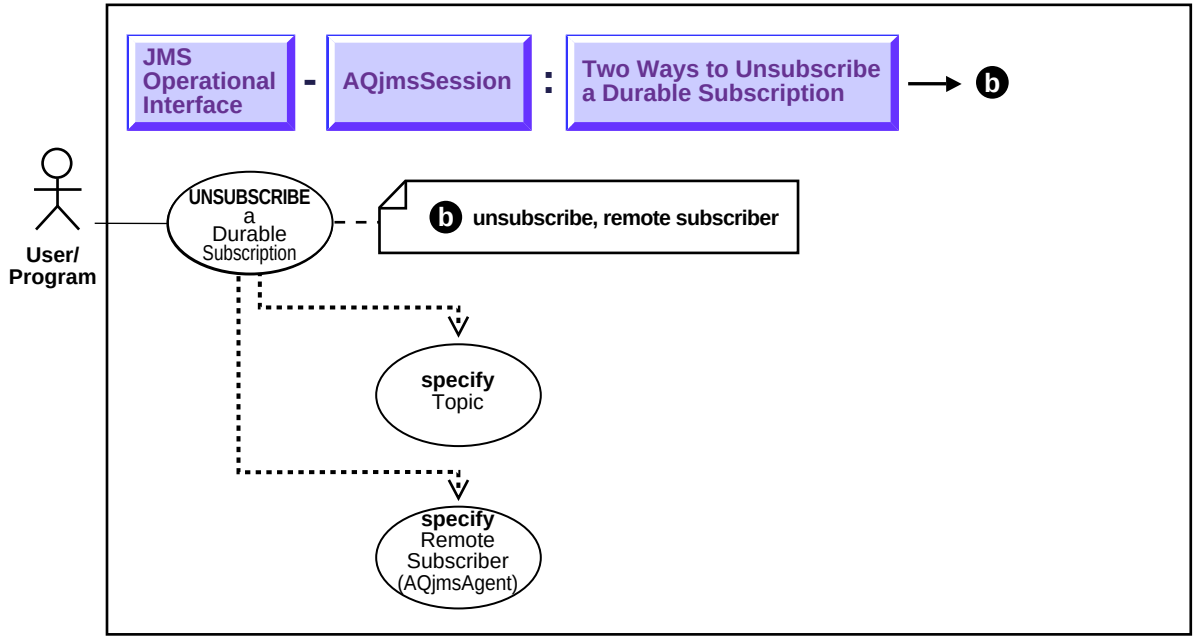
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.unsubscribe

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

Figure 15–25 Use Case Diagram: Publish-Subscribe - Unsubscribe a Durable Subscription for a Remote Subscriber



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Unsubscribe a durable subscription for a remote subscriber.

Usage Notes

Not applicable.

Syntax

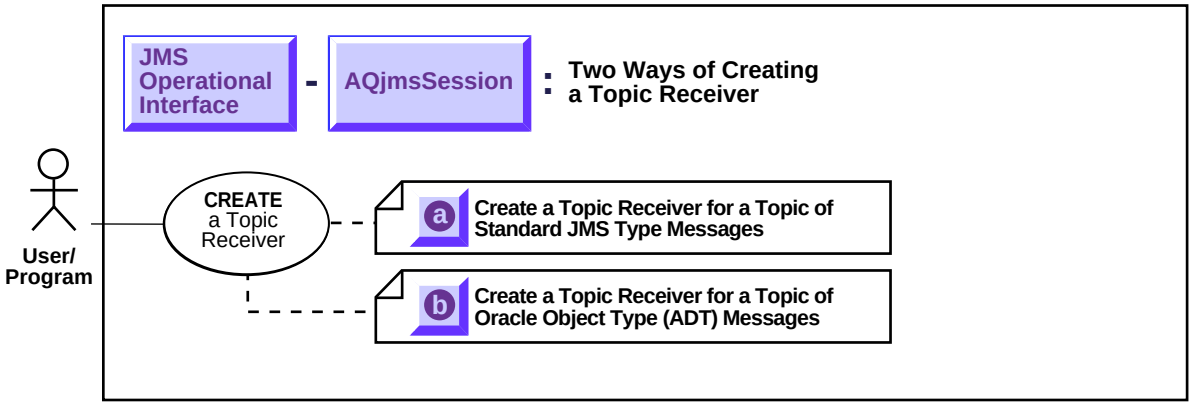
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.unsubscribe

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);
```

Two Ways of Creating a Topic Receiver

Figure 15–26 Use Case Diagram: Publish-Subscribe - Two Ways to Create a Topic Receiver



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

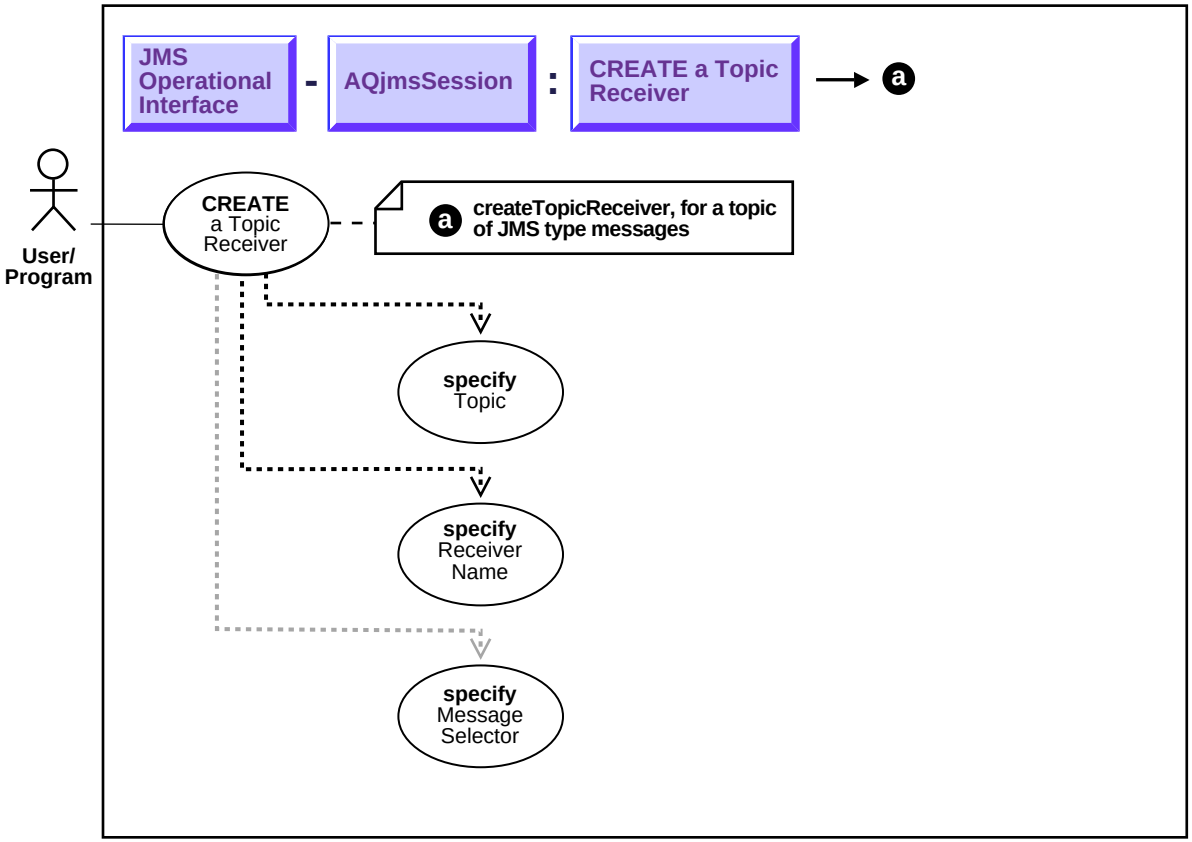
This section explains two ways to create a topic receiver.

a. Creating a Topic Receiver for a Topic of Standard JMS Type Messages on page 15-53

b. Creating a Topic Receiver for a Topic of Oracle Object Type (ADT) Messages on page 15-55

Creating a Topic Receiver for a Topic of Standard JMS Type Messages

Figure 15–27 Use Case Diagram: Publish-Subscribe - Create a Topic Receiver for a Topic of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic receiver for a topic of standard jms type messages.

Usage Notes

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, it will receive 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.

Syntax

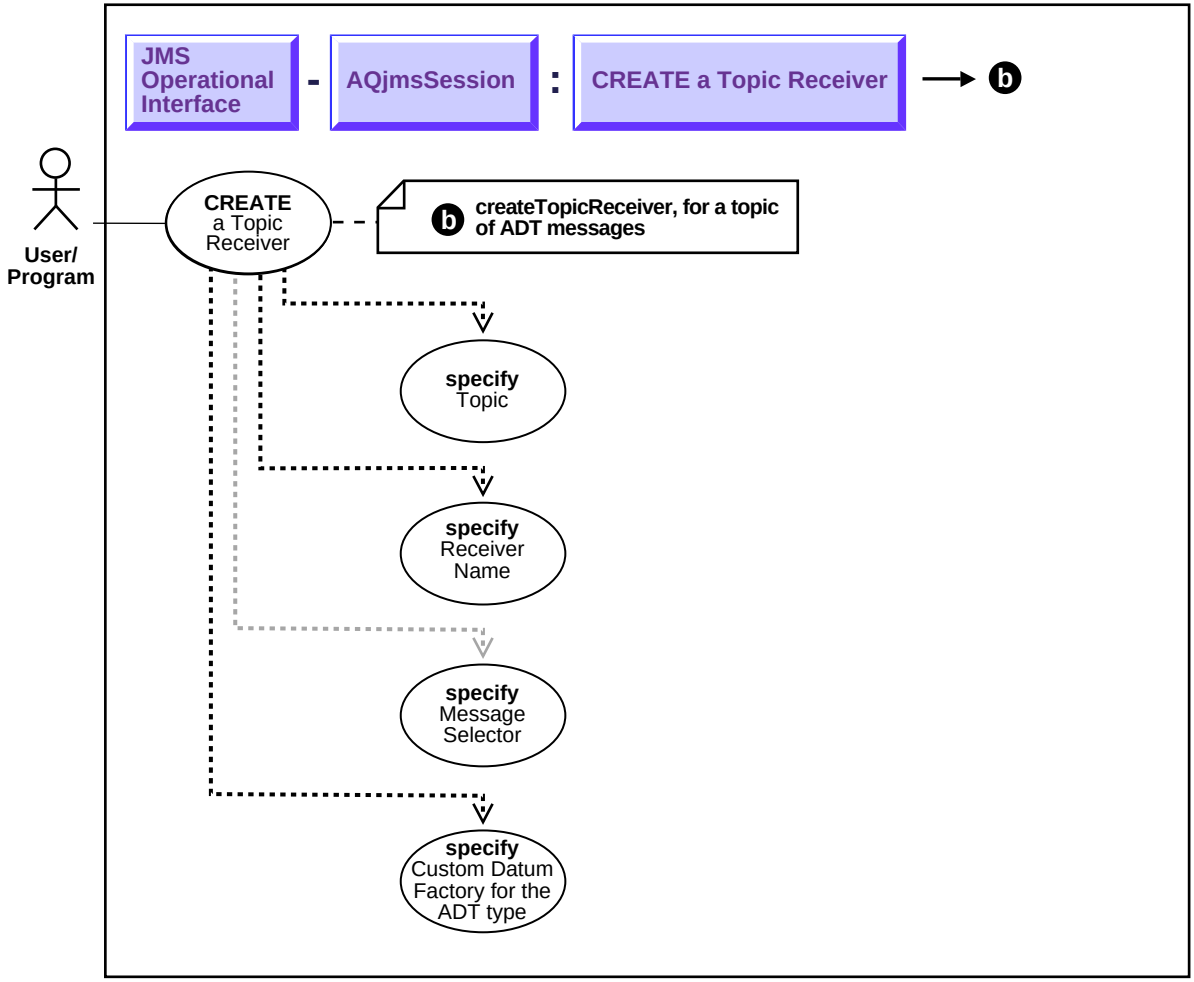
- Java (JDBC): *Oracle9i Supplied Java Packages Reference,*oracle.jms, AQjmsSession.createTopicReceiver

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 Topic Receiver for a Topic of Oracle Object Type (ADT) Messages

Figure 15–28 Use Case Diagram: Publish-Subscribe - Create a Topic Receiver for a Topic of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic receiver for a topic of adt messages with selector.

Usage Notes

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, it will receive 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 ADT messages.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*,oracle.jms, AQjmsSession.createTopicReceiver

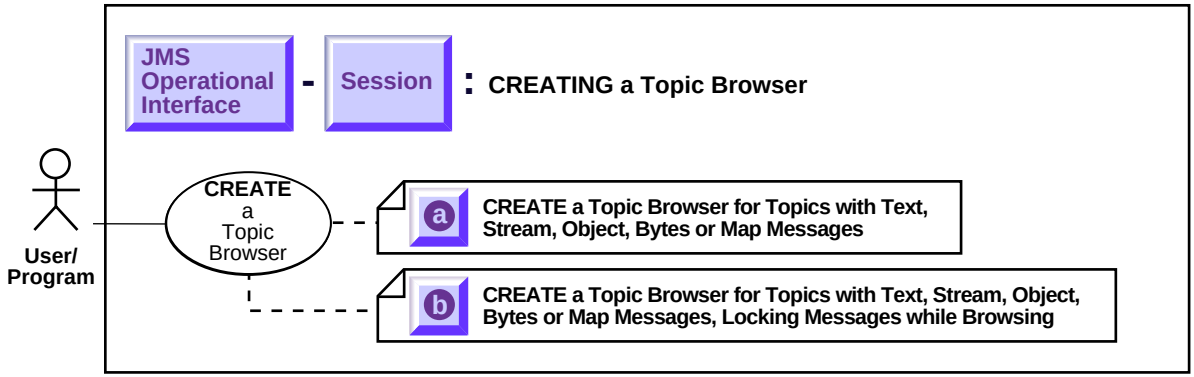
Example

```
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);
```

Two Ways of Creating a Topic Browser for JMS Message Queues

Figure 15–29 Use Case Diagram: Two Ways to Create a Topic Browser for Topics of Standard JMS Type Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

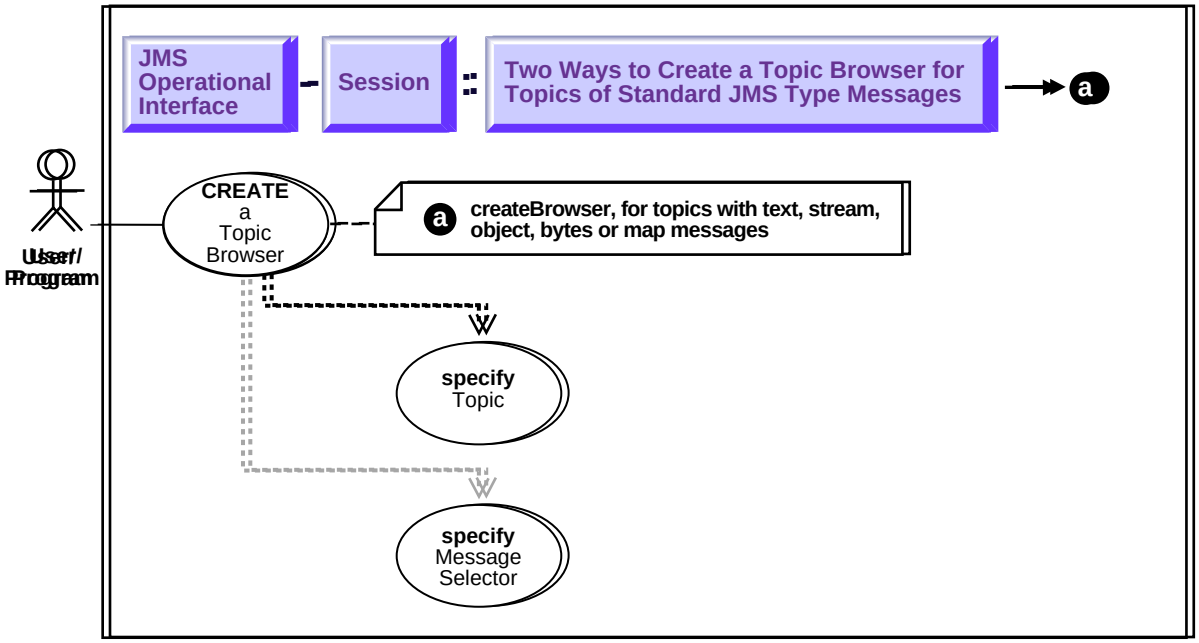
This section explains two ways to create a topic browser for JMS message topics:

a. Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages on page 15-58

b. Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages, Locking Messages While Browsing on page 15-60

Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages

Figure 15–30 Use Case Diagram: Create a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic browser for topics with text, stream, objects, bytes, or map messages.

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.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

Example 1

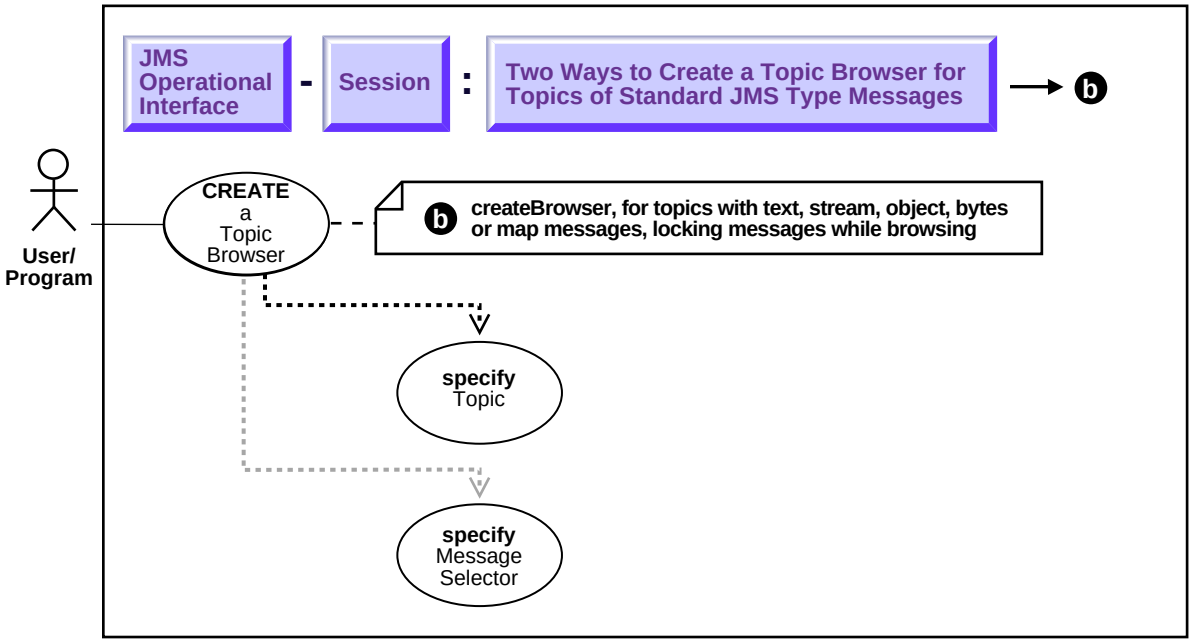
```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;

browser = ((AQjmsSession) jms_session).createBrowser(topic, "SUBS1");
```

Example2

```
/* Create a browser for topics with a specified selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;

/* create a Browser to look at messages with correlationID = RUSH  */
browser = ((AQjmsSession) jms_session).createBrowser(topic, "SUBS1",
    "JMSCorrelationID = 'RUSH'");
```

Creating a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages, Locking Messages While Browsing

Figure 15–31 Use Case Diagram: Create a Topic Browser for Topics with Text, Stream, Objects, Bytes or Map Messages, Locking Messages While Browsing



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic browser for topics with text, stream, objects, bytes or map messages, locking messages while browsing.

Usage Notes

If a locked parameter is specified as true, messages are locked as they are browsed. Hence these messages cannot be removed by other consumers until the browsing session ends the transaction.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

Example 1

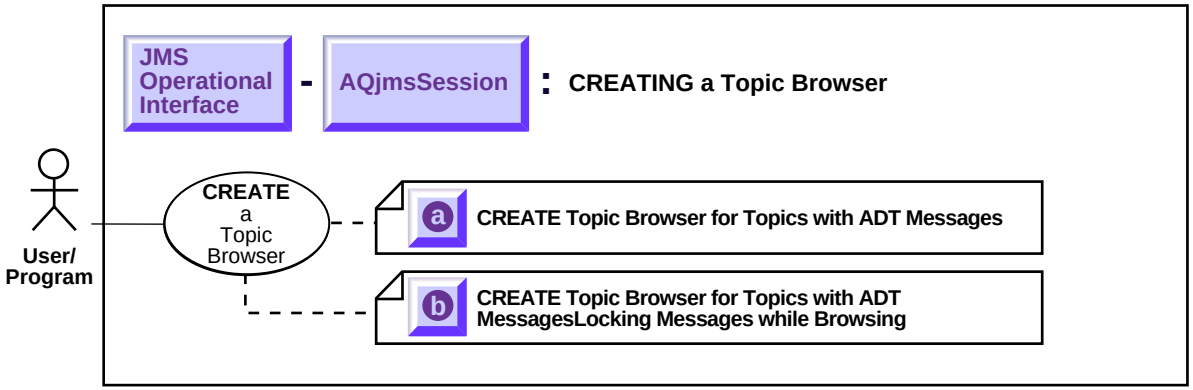
```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;

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);
```

Two Ways of Creating a Topic Browser for Oracle Object Type (ADT) Message Topics

Figure 15–32 Use Case Diagram: Two Ways to Create a Topic Browser for Topics of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

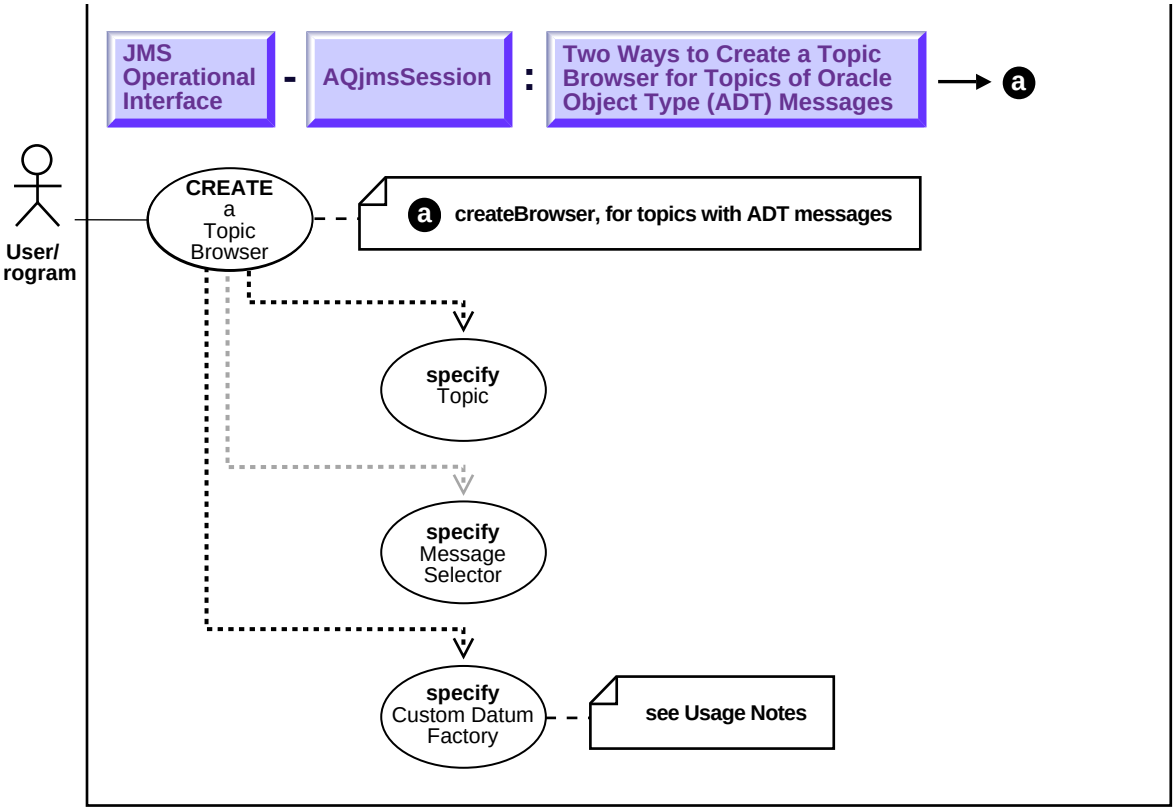
This section explains two ways to create a browser for Oracle object type (ADT) messages topics:

a. Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages on page 15-63

b. Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages, Locking Messages While Browsing on page 15-66

Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages

Figure 15–33 Use Case Diagram: Create a Topic Browser for Topics of Oracle Object Type (ADT) Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic browser for topics of Oracle object type (ADT) messages.

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`

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

Example

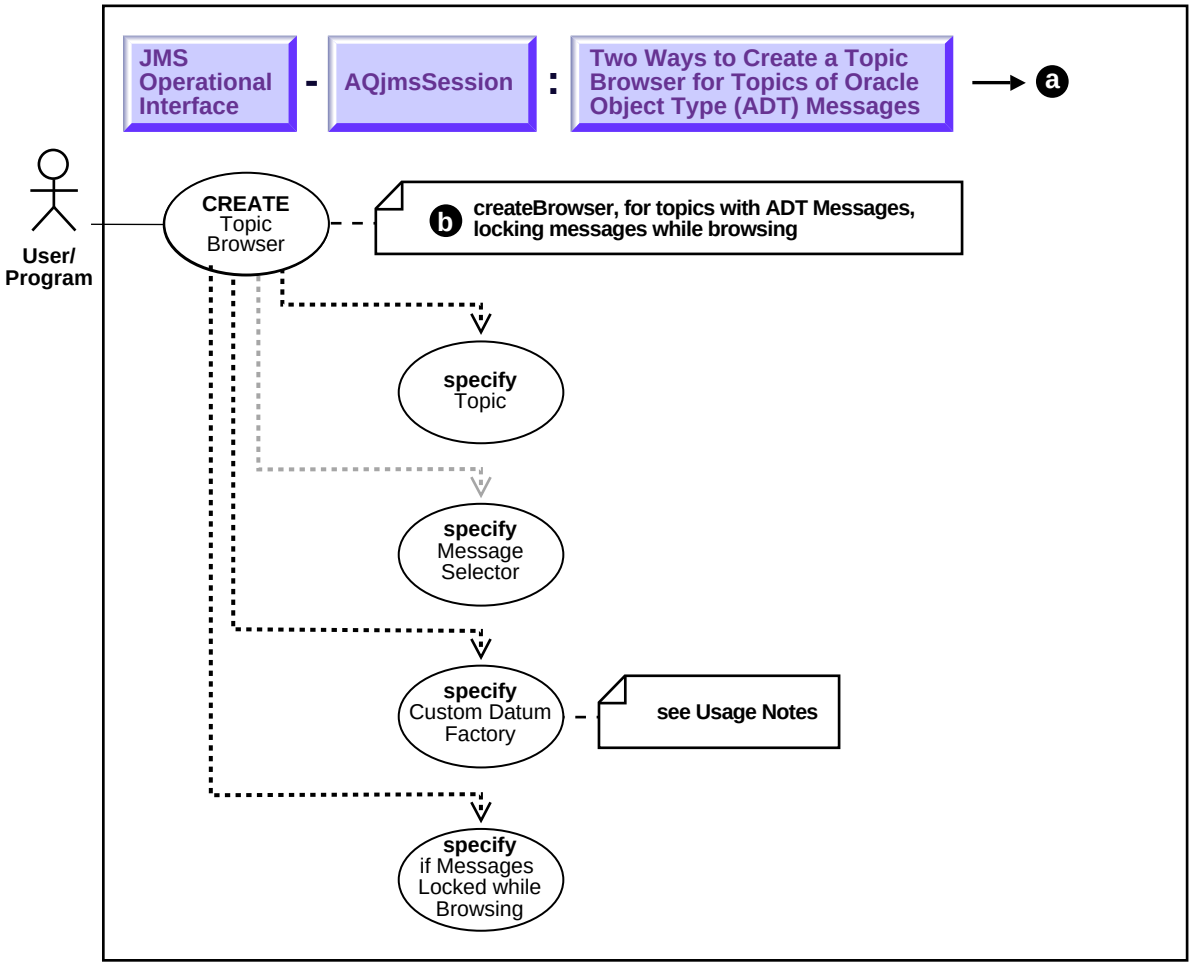
The CustomDatum factory for a particular Java class that maps to the SQL ADT payload can be obtained via the `getFactory` static method. Assume the `Topic - test_topic` has payload of type `SCOTT.EMPLOYEE` and the Java class that is generated by Jpublisher for this ADT is called `Employee`. The Employee 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());
```

Creating a Topic Browser for Topics of Oracle Object Type (ADT) Messages, Locking Messages While Browsing

Figure 15–34 Use Case Diagram: Create a Topic Browser for Topics of Oracle Object Type (ADT) Messages, Locking Messages while Browsing



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a topic browser for topics of Oracle object type (ADT) messages, locking messages while browsing.

Usage Notes

Not applicable.

Syntax

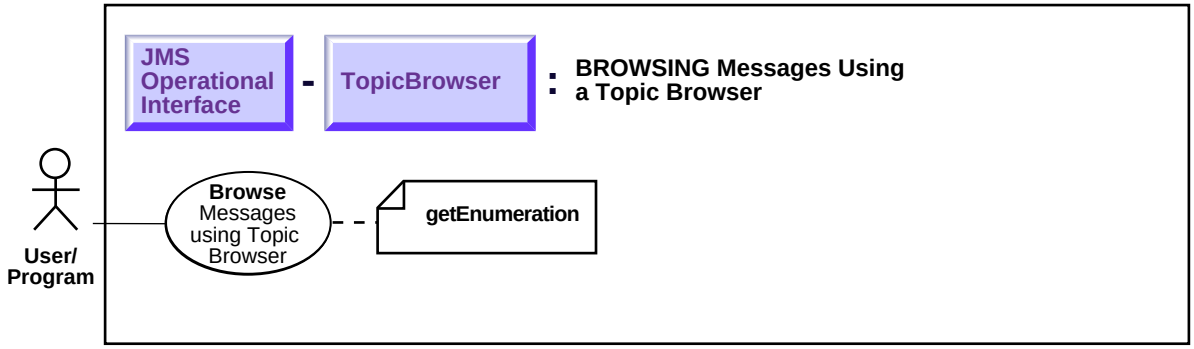
- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBrowser

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 Topic Browser

Figure 15–35 Use Case Diagram: Browse Messages Using a Topic Browser



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Browse messages using a topic browser.

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.

Syntax

- Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, TopicBrowser, AQjmsTopicBrowser

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();
}
```

16

JMS Operational Interface: Basic Operations (Shared Interfaces)

In this chapter we describe the operational interface to Oracle Advanced Queuing in terms of use cases. That is, we discuss each operation (such as "Enqueue a Message") as a use case by that name. The table listing all the use cases is provided at the head of the chapter (see " Use Case Model: Operational Interface — Basic Operations (Shared Interfaces)" on page 16-2).

A summary figure, "Use Case Diagram: Operational Interface — Basic Operations", locates all the use cases in a single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case that interests you by clicking on the relevant use case title.

Each use case is laid out as follows:

- *Use case figure.* A figure that depicts the use case.
- *Purpose.* The purpose of this use case.
- *Usage Notes.* Guidelines to assist implementation.
- *Syntax*. The main syntax used to perform this activity.
- *Examples*. Examples in each programmatic environment that illustrate the use case.

Use Case Model: JMS Operational Interface — Basic Operations (Shared Interfaces)

Table 16–1 Use Case Model: Operational Interface — Basic Operations (Shared Interfaces)

Use Case
Starting a JMS Connection on page 16-6
Getting the JMS Connection from a Session on page 16-8
Committing All Operations in a Transacted Session on page 16-9
Rolling Back All Operations in a Transacted Session on page 16-10
Getting the Underlying JDBC Connection from a JMS Session on page 16-12
Getting the Underlying OracleOCIConnectionPool from a JMS Connection on page 16-13
Create a Message of Different Types
Creating a Bytes Message on page 16-15
Creating a Map Message on page 16-16
Creating a Stream Message on page 16-17
Creating an Object Message on page 16-19
Creating a Text Message on page 16-21
Creating an ADT Message on page 16-23
Specifying Message Correlation ID on page 16-25
Specifying JMS Message Property on page 16-27
Specify JMS Property as Different Types
Specifying Message Property as Boolean on page 16-29
Specifying Message Property as String on page 16-31
Specifying Message Property as Int on page 16-33
Specifying Message Property as Double on page 16-35
Specifying Message Property as Float on page 16-37
Specifying Message Property as Byte on page 16-39
Specifying Message Property as Long on page 16-41
Specifying Message Property as Short on page 16-43

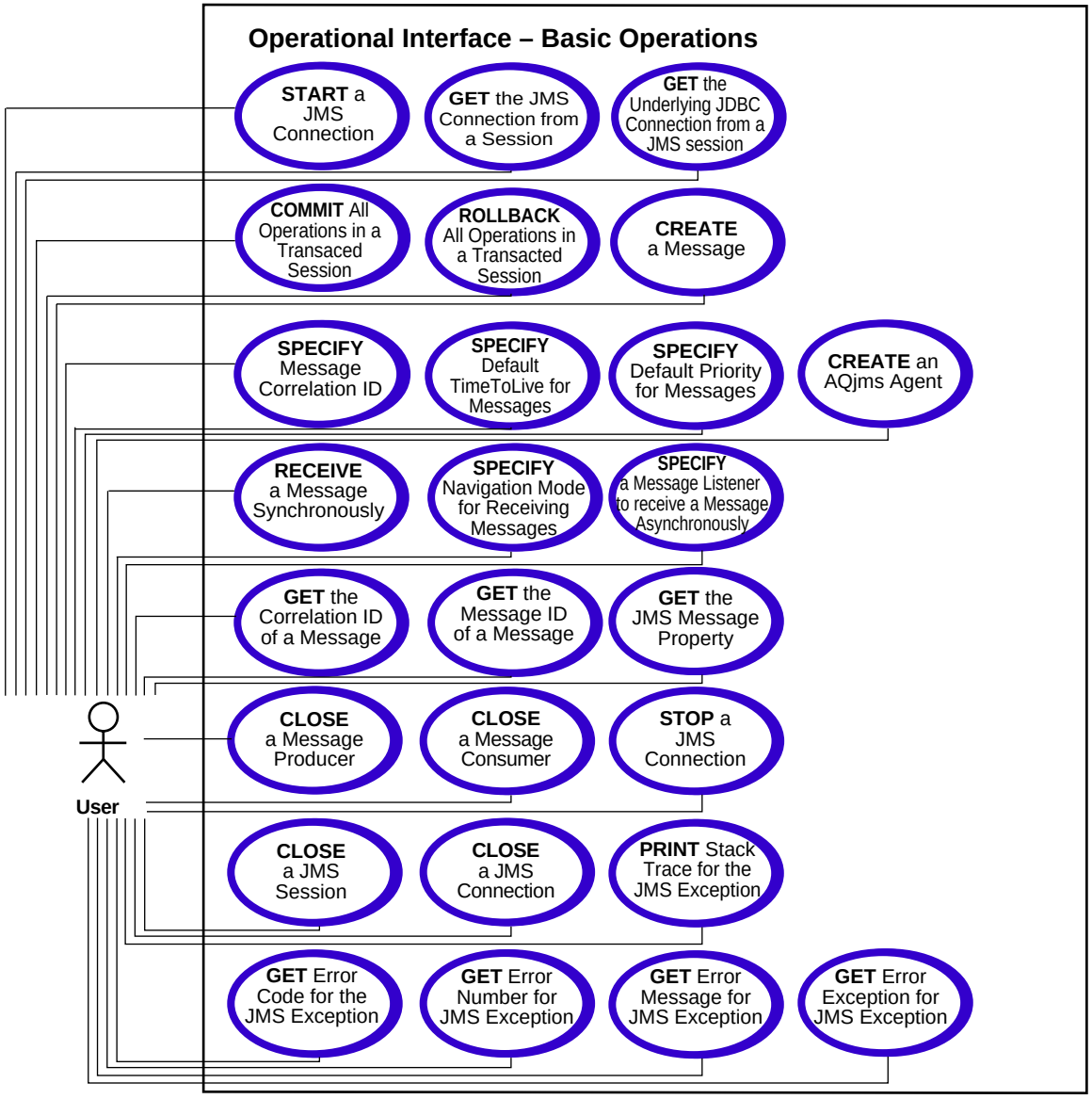
Table 16–1 (Cont.) Use Case Model: Operational Interface — Basic Operations (Shared Interfaces)

Use Case
Specifying Message Property as Object on page 16-45
Setting Default TimeToLive for All Messages Sent by a Message Producer on page 16-46
Setting Default Priority for All Messages Sent by a Message Producer on page 16-48
Creating an AQjms Agent on page 16-50
Two Ways of Receiving a Message Synchronously Using a Message Consumer on page 16-52
Receiving a Message Using a Message Consumer by Specifying Timeout on page 16-54
Receiving a Message Using a Message Consumer Without Waiting on page 16-56
Specifying the Navigation Mode for Receiving Messages on page 16-57
Two Ways of Specifying a Message Listener to Receive a Message Asynchronously on page 16-60
Specifying a Message Listener at the Message Consumer on page 16-61
Specifying a Message Listener at the Session on page 16-64
Getting the Correlation ID of a Message on page 16-65
Two Ways of Getting the Message ID of a Message on page 16-67
Getting the Message ID of a Message as Bytes on page 16-68
Getting the Message ID of a Message as a String on page 16-69
Getting the JMS Message Property on page 16-71
Get the JMS Message Property of Different Types
Getting the Message Property as a Boolean on page 16-73
Getting the Message Property as a String on page 16-74
Getting the Message Property as Int on page 16-76
Getting the Message Property as Double on page 16-78
Getting the Message Property as Float on page 16-79
Getting the Message Property as Byte on page 16-81
Getting the Message Property as Long on page 16-82
Getting the Message Property as Short on page 16-84
Getting the Message Property as Object on page 16-84
Closing a Message Producer on page 16-87

Table 16–1 (Cont.) Use Case Model: Operational Interface — Basic Operations (Shared Interfaces)

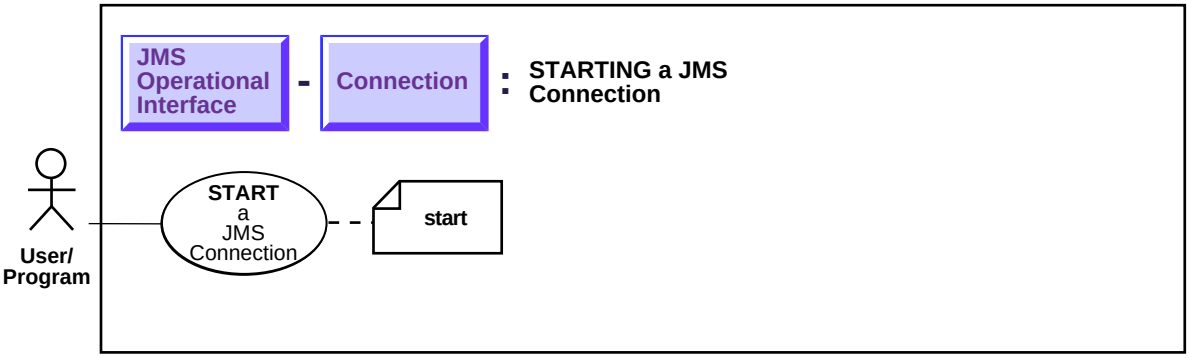
Use Case
Closing a Message Consumer on page 16-88
Stopping a JMS Connection on page 16-90
Closing a JMS Session on page 16-91
Closing a JMS Connection on page 16-93
Getting the Error Code for the JMS Exception on page 16-94
Getting the Error Number for the JMS Exception on page 16-95
Getting the Error Message for the JMS Exception on page 16-97
Getting the Exception Linked to the JMS Exception on page 16-98
Printing the Stack Trace for the JMS Exception on page 16-99
Setting the Exception Listener on page 16-101
Getting the Exception Listener on page 16-103
Setting the Ping Period for the Exception Listener on page 16-104
Getting the Ping Period for the Exception Listener on page 16-105

Figure 16–1 Use Case Model Diagram: Operational Interface



Starting a JMS Connection

Figure 16–2 Use Case Diagram: Start a JMS Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Start a JMS Connection for receiving messages.

Usage Notes

The start method is used to start (or restart) the connection's delivery of incoming messages.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms AQjmsConnection.start

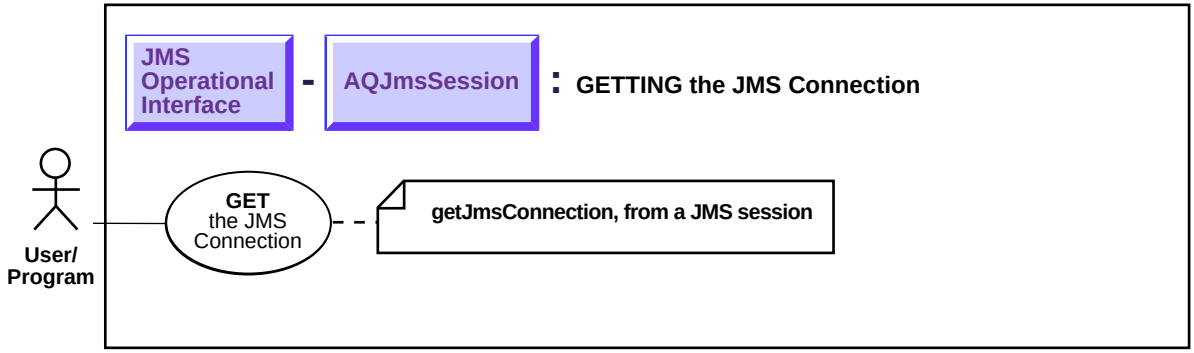
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the JMS Connection from a Session

Figure 16–3 Use Case Diagram: Get the JMS Connection from a Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the JMS Connection from a Session

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.getJmsConnection

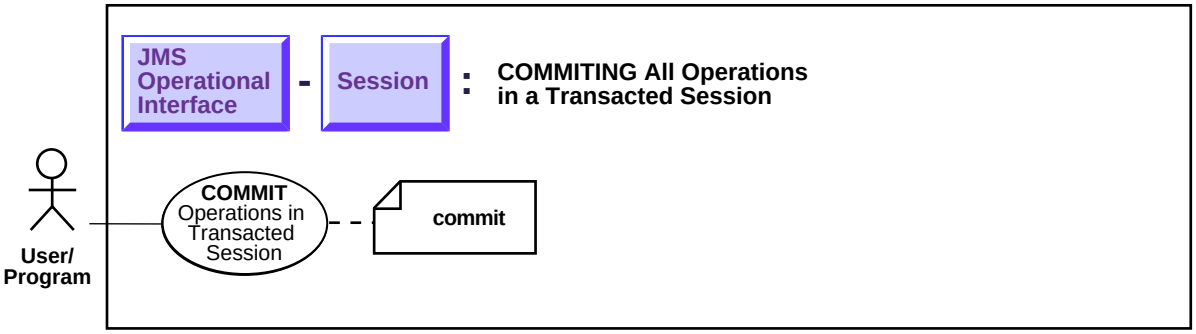
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Committing All Operations in a Transacted Session

Figure 16–4 Use Case Diagram: Commit All Operations in a Transacted Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Commit All Operations in a Transacted Session

Usage Notes

This method commits all JMS and SQL operations performed in this session.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.commit

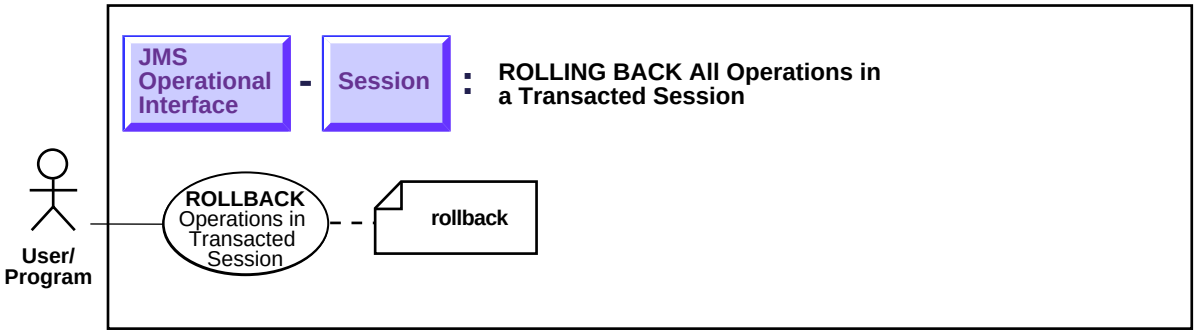
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Rolling Back All Operations in a Transacted Session

Figure 16–5 Use Case Diagram: Rollback All Operations in a Transacted Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Rollback All Operations in a Transacted Session

Usage Notes

This method aborts all JMS and SQL operations performed in this session.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.rollback

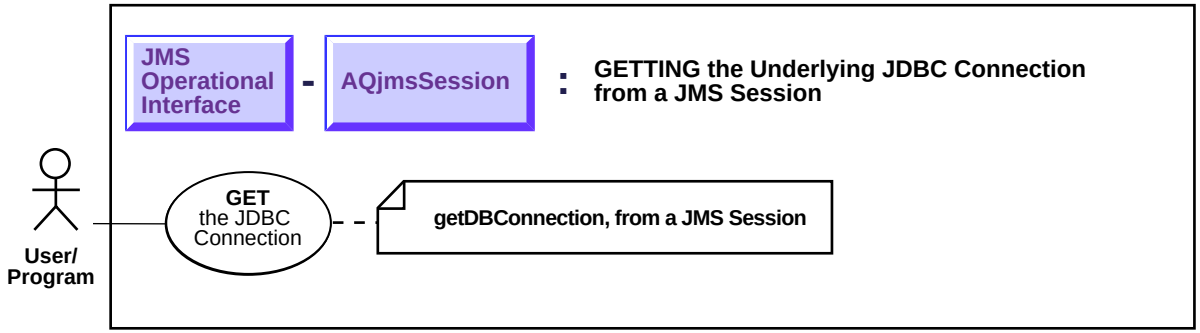
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Underlying JDBC Connection from a JMS Session

Figure 16–6 Use Case Diagram: Get the Underlying JDBC Connection from a JMS Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Underlying JDBC Connection from a JMS session

Usage Notes

This method is used to obtain the underlying JDBC connection from a JMS session. The JDBC connection may be used to perform SQL operations as part of the same transaction that the JMS operations are done.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.getDBConnection

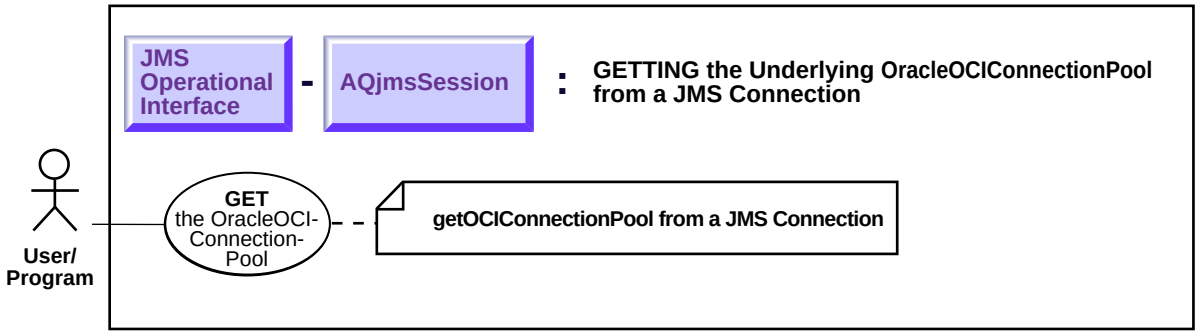
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
java.sql.Connection db_conn;
QueueSession     jms_sess;
db_conn = ((AQjmsSession)jms_sess).getDBConnection();
```

Getting the Underlying OracleOCIConnectionPool from a JMS Connection

Figure 16–7 Use Case Diagram: Get the Underlying OracleOCIConnectionPool from a JMS Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the underlying `OracleOCIConnectionPool` from a JMS connection.

Usage Notes

This method is used to obtain the underlying `OracleOCIConnectionPool` instance from a JMS connection. The settings of the `OracleOCIConnectionPool`

instance may 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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

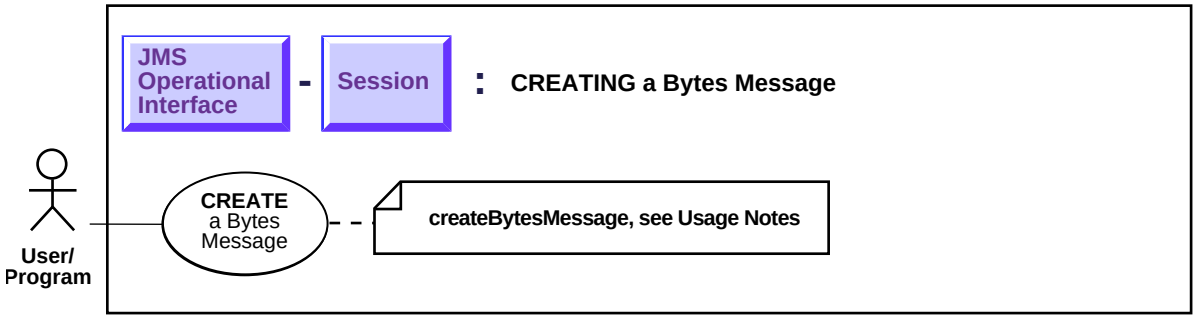
- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsConnection.getOCIConnectionPool

Examples

```
oracle.jdbc.pool.OracleOCIConnectionPool cpool;
QueueConnection jms_conn;
cpool = ((AQjmsConnection)jms_conn).getOCIConnectionPool();
```

Creating a Bytes Message

Figure 16–8 Use Case Diagram: Create a Bytes Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a Bytes Message

Usage Notes

This method can be used only if the queue table which contains the destination queue/topic has been created with payload_type SYS.AQ$_JMS_BYTES_MESSAGE

Refer to Java Packages Reference for methods used to populate a BytesMessage.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createBytesMessage

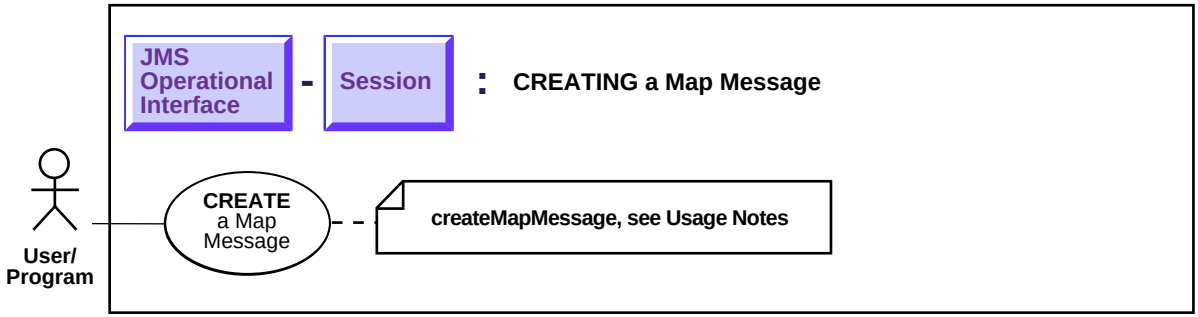
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Creating a Map Message

Figure 16–9 Use Case Diagram: Create a Map Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a Map Message

Usage Notes

This method can be used only if the queue table which contains the destination queue/topic has been created with payload_type SYS.AQ$_JMS_MAP_MESSAGE.

Refer to Java Packages Reference for methods used to populate a MapMessage.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.createMapMessage

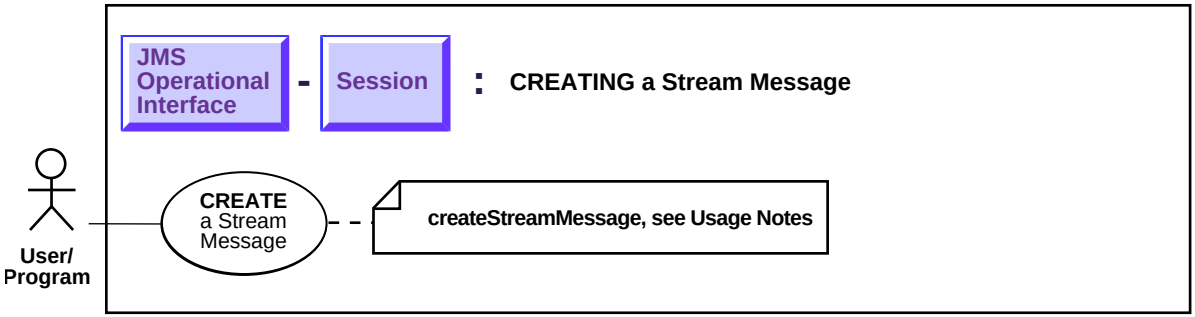
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Creating a Stream Message

Figure 16–10 Use Case Diagram: Create a Stream Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a Stream Message

Usage Notes

This method can be used only if the queue table which contains the destination queue/topic has been created with payload_type SYS.AQ$_JMS_STREAM_MESSAGE.

Refer to Java Packages Reference for methods used to populate a StreamMessage.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.createStreamMessage

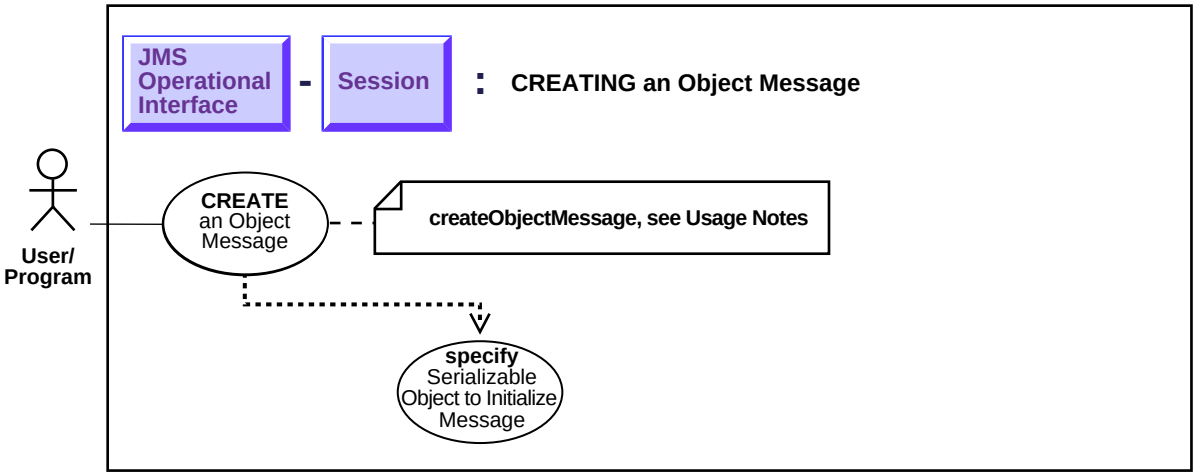
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Creating an Object Message

Figure 16–11 Use Case Diagram: Create an Object Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create an Object Message

Usage Notes

This method can be used only if the queue table which contains the destination queue/topic has been created with payload_type SYS.AQ$_JMS_OBJECT_MESSAGE.

Refer to Java Packages Reference for methods used to populate a ObjectMessage.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.createObjectMessage

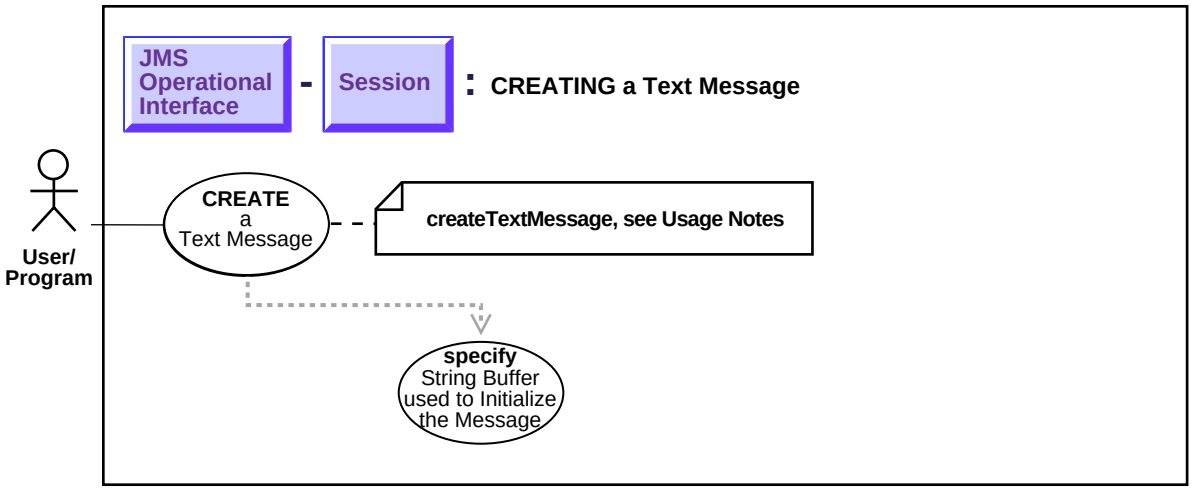
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Creating a Text Message

Figure 16–12 Use Case Diagram: Create a Text Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create a Text Message

Usage Notes

This method can be used only if the queue table which contains the destination queue/topic has been created with payload_type SYS.AQ$_JMS_TEXT_MESSAGE.

Refer to Java Packages Reference for methods used to populate a Text Message.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.createTextMessage

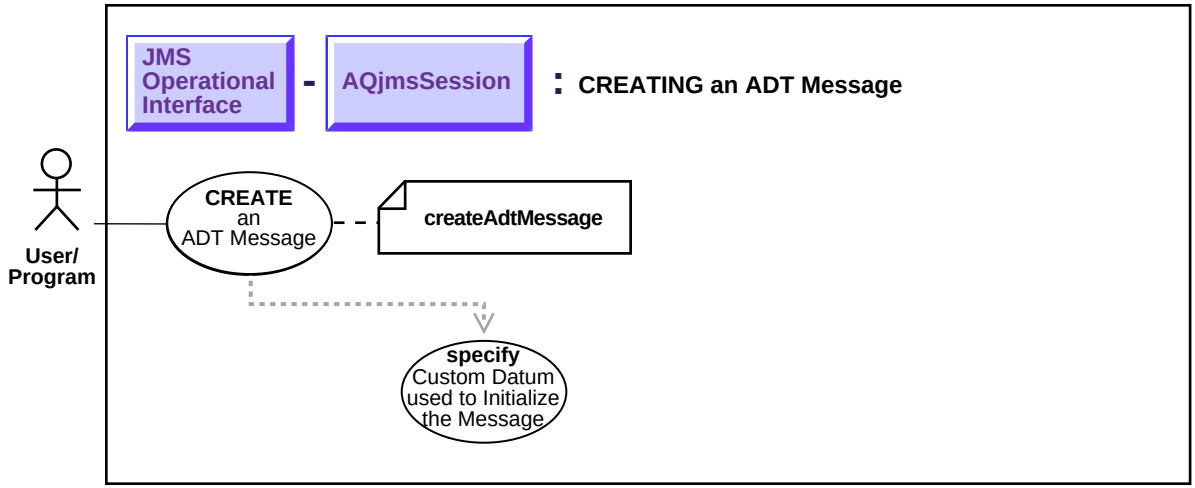
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Creating an ADT Message

Figure 16–13 Use Case Diagram: Create an ADT Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create an ADT Message

Usage Notes

This method can be used only if the queue table which contains the queue/topic has been 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 may be generated using the Jpublisher tool. Please refer to the JDBC documentation for details on CustomDatum interface and Jpublisher.

Refer to Java Packages Reference for methods used to populate a AdtMessage.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsSession.createAdtMessage

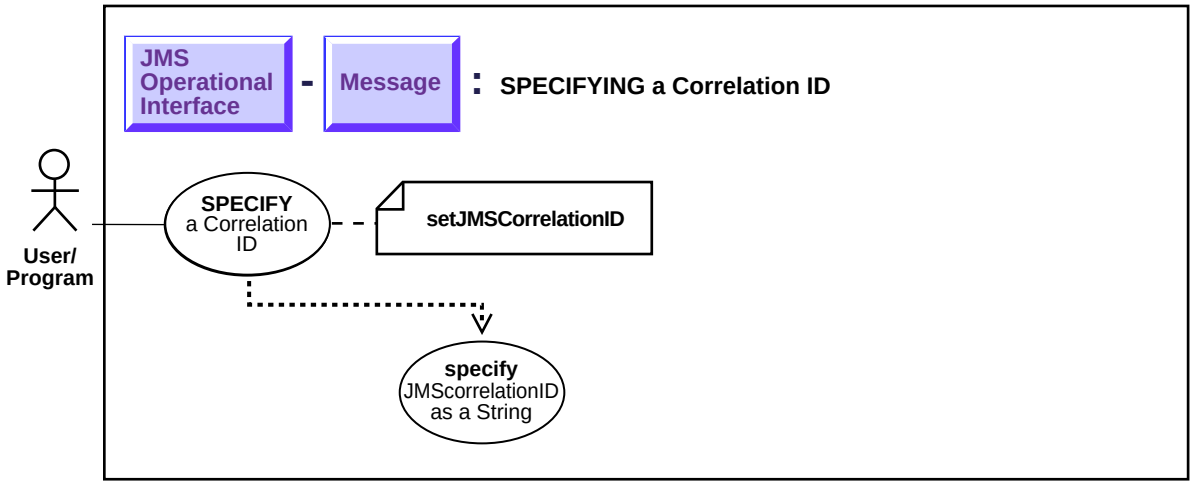
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available f.nctions in each programmatic environment.

- No example is provided with this release.

Specifying Message Correlation ID

Figure 16–14 Use Case Diagram: Specify Message Correlation ID



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify message correlation ID.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setJMSCorrelationID

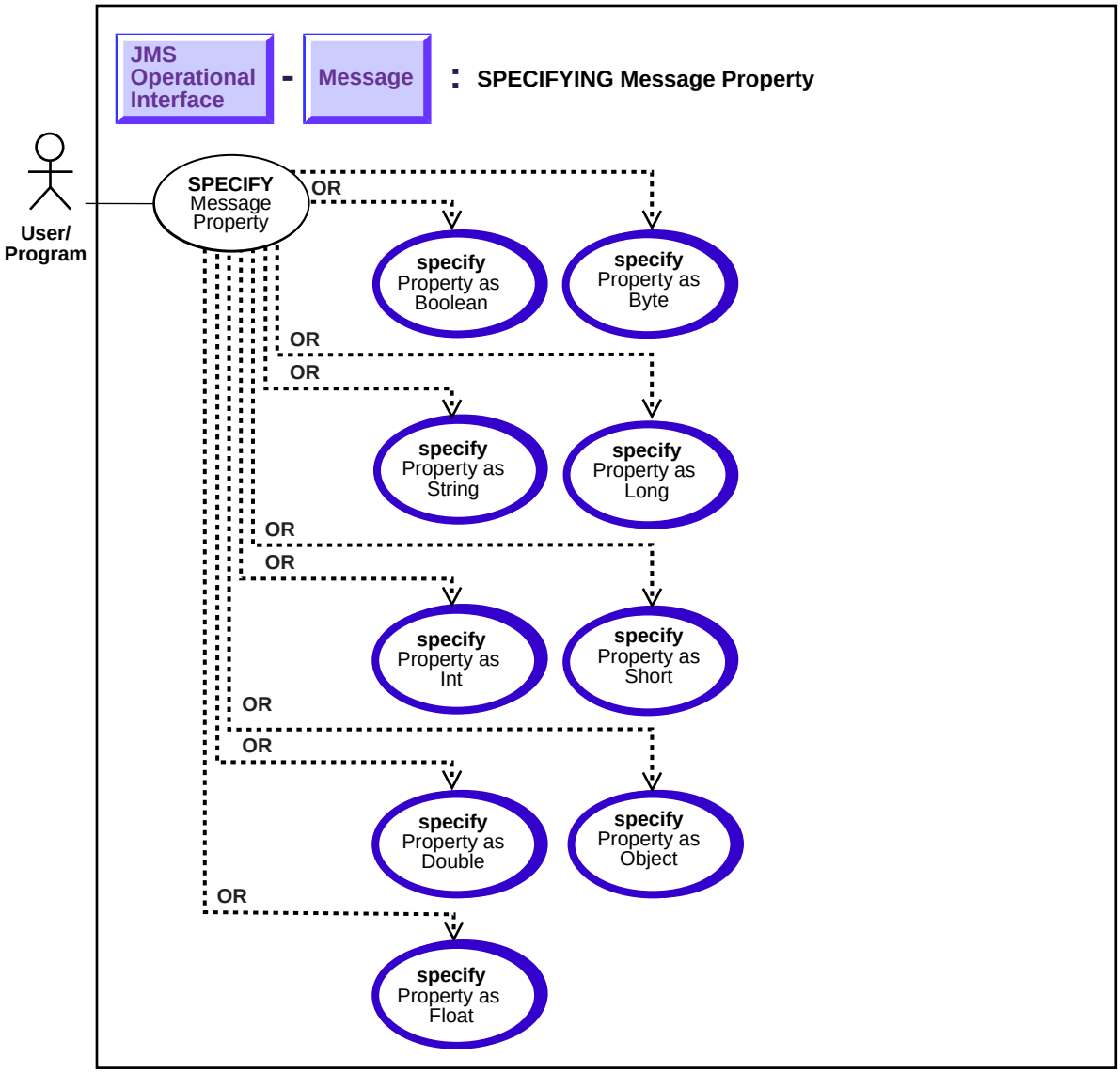
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying JMS Message Property

Figure 16–15 Use Case Diagram: Specify JMS Message Property



Usage Notes

Property names starting with JMS are provider specific. User-defined properties cannot start with JMS.

The following provider properties may be set by clients using Text, Stream, Object, Bytes or Map Message:

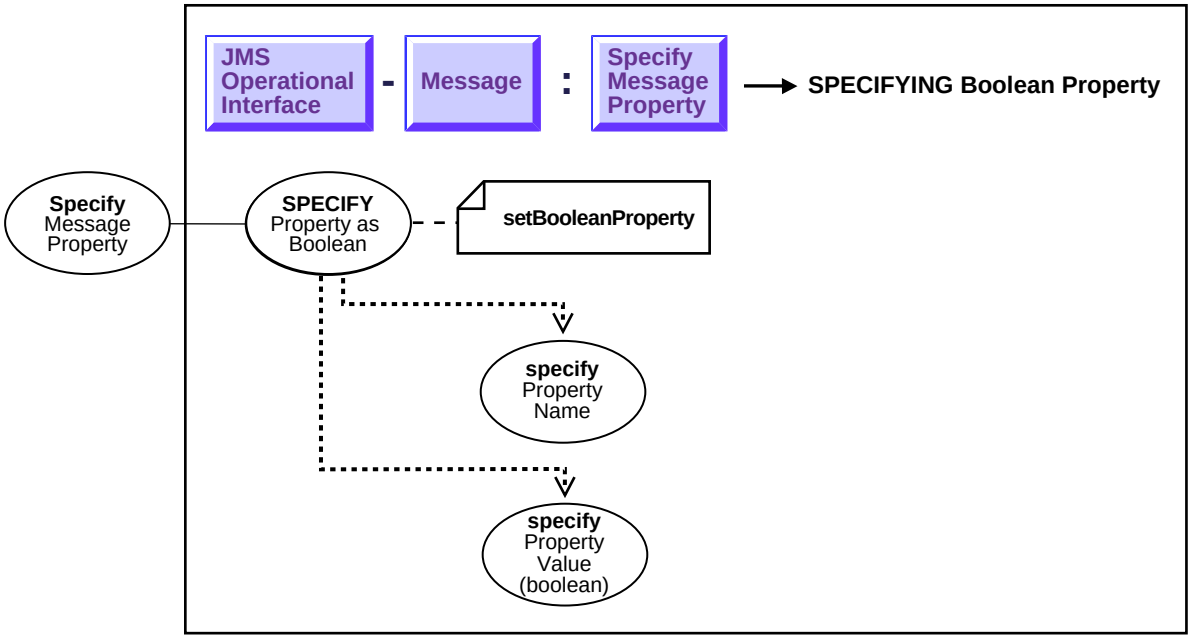
- JMSXAppID (String)
- JMSXGroupID (string)
- JMSXGroupSeq (int)
- JMS_OracleExcpQ (String) - exception queue
- JMS_OracleDelay (int) - message delay (seconds)

The following properties may be set on AdtMessage

- JMS_OracleExcpQ (String) - exception queue - specified as "<schema>.queue_name"
- JMS_OracleDelay (int) - message delay (seconds)

Specifying Message Property as Boolean

Figure 16–16 Use Case Diagram: Specify Message Property as Boolean



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Boolean

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setBooleanProperty

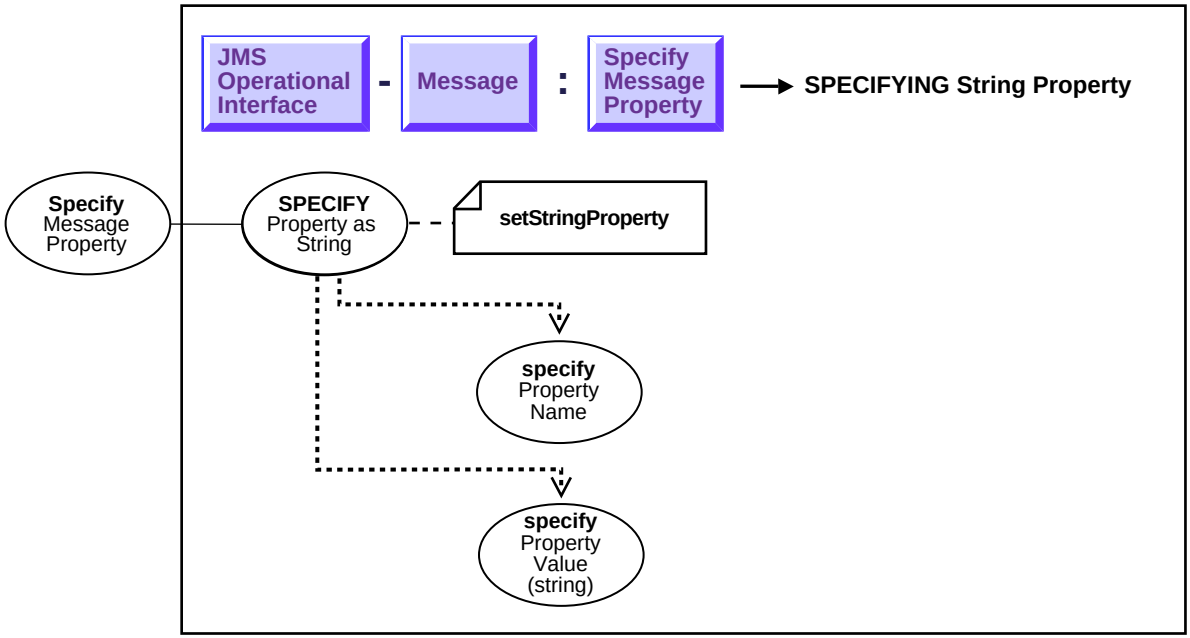
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as String

Figure 16–17 Use Case Diagram: Specify Message Property as String



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as String

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setStringProperty

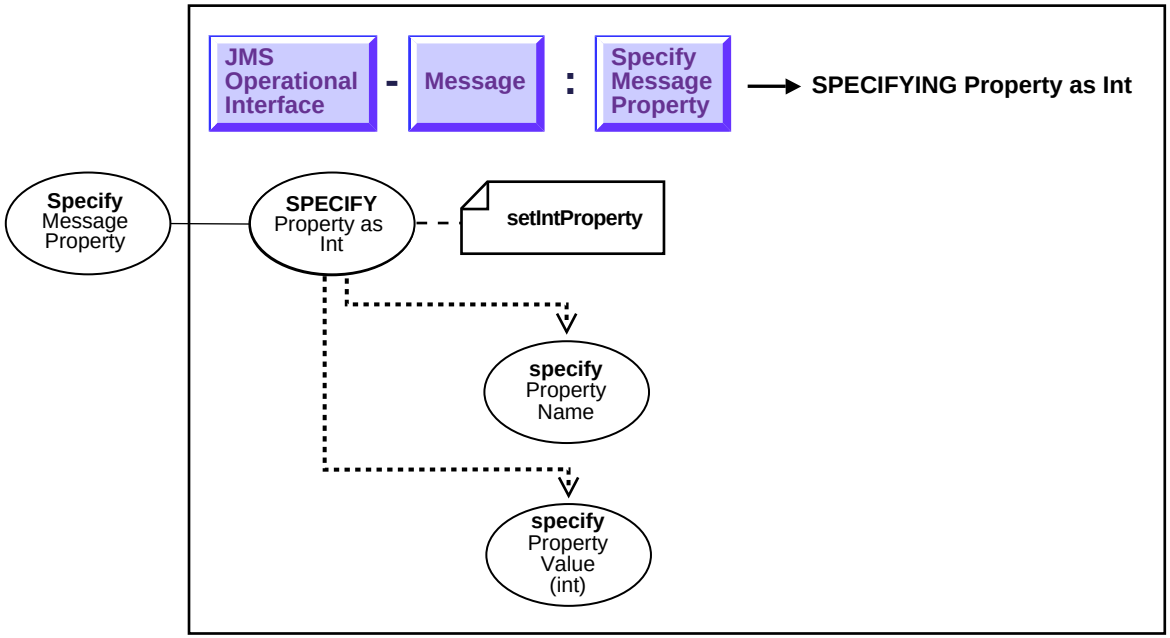
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Int

Figure 16–18 Use Case Diagram: Specify Message Property as Int



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Int

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setIntProperty

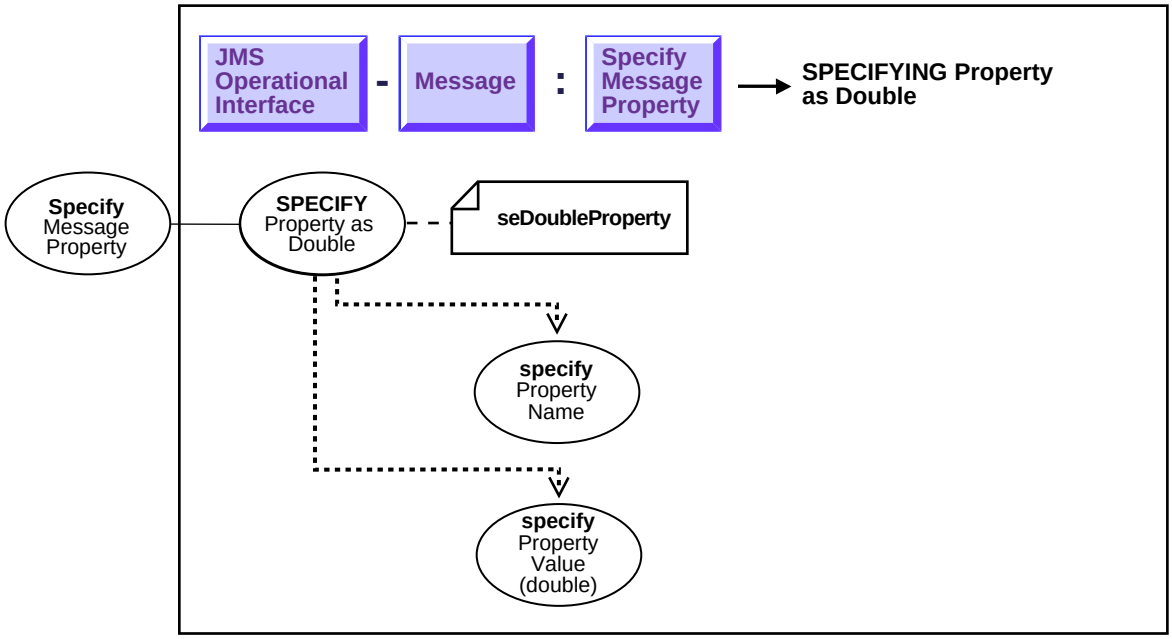
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Double

Figure 16–19 Use Case Diagram: Specify Message Property as Double



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Double

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setDoubleProperty

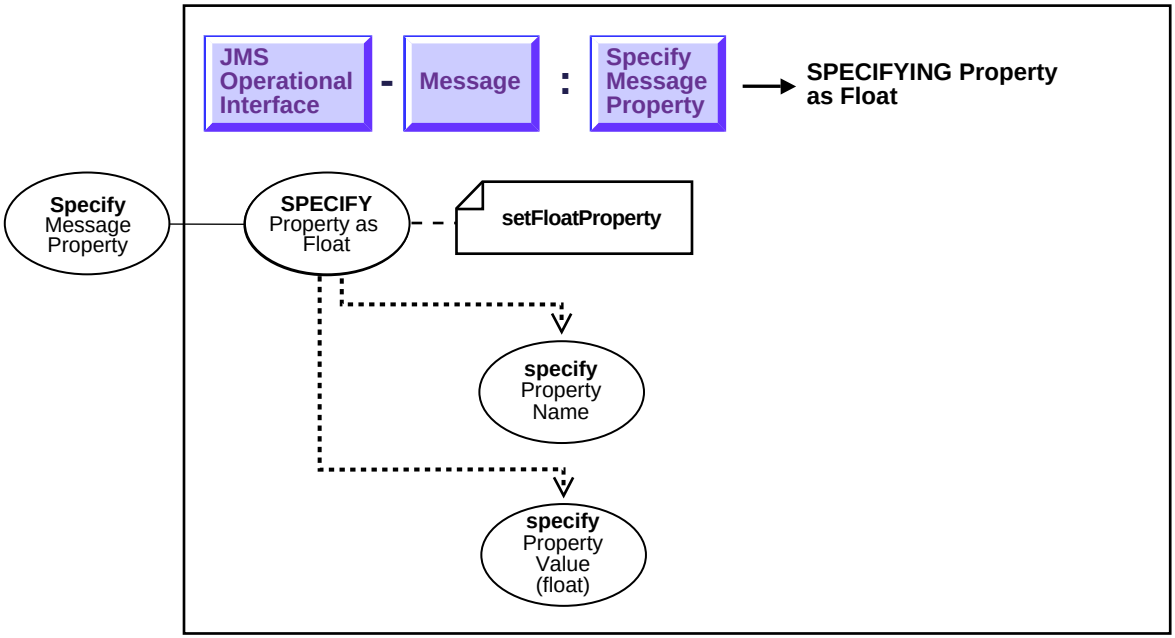
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Float

Figure 16–20 Use Case Diagram: Specify Message Property as Float



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Float

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setFloatProperty

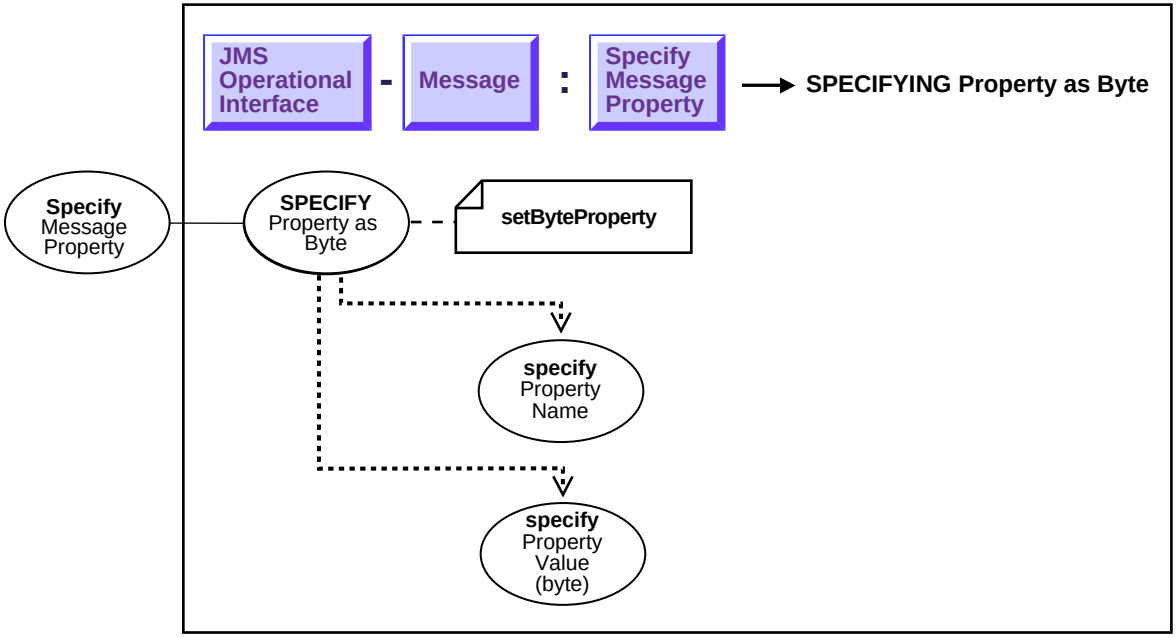
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Byte

Figure 16–21 Use Case Diagram: Specify Message Property as Byte



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.setByteProperty

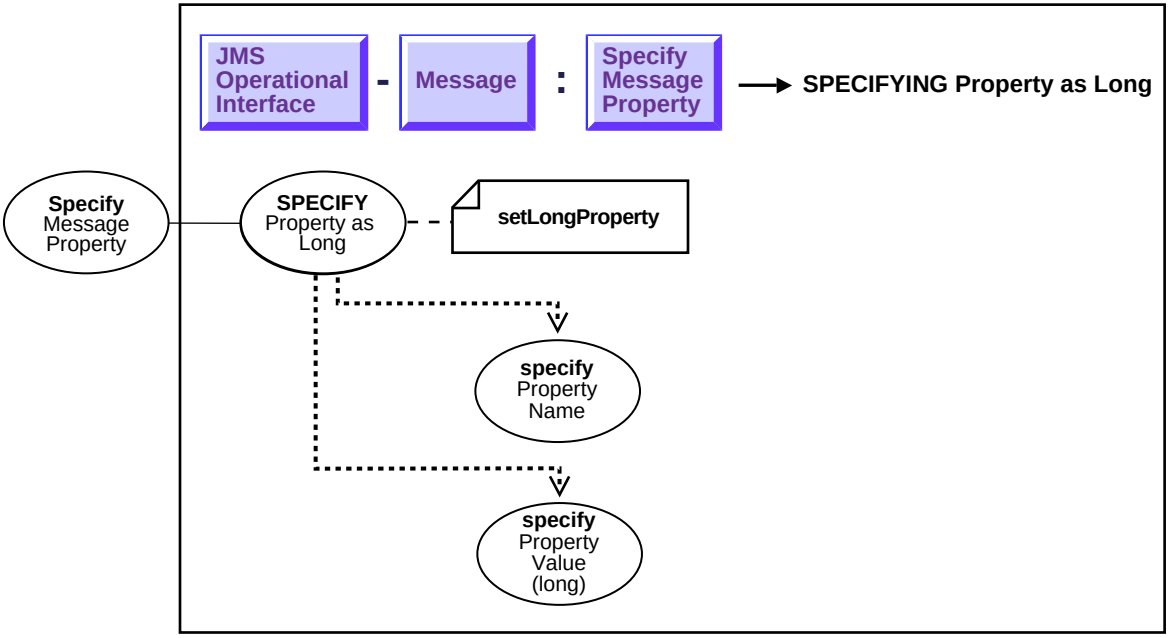
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Long

Figure 16–22 Use Case Diagram: Specify Message Property as Long



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.setLongProperty

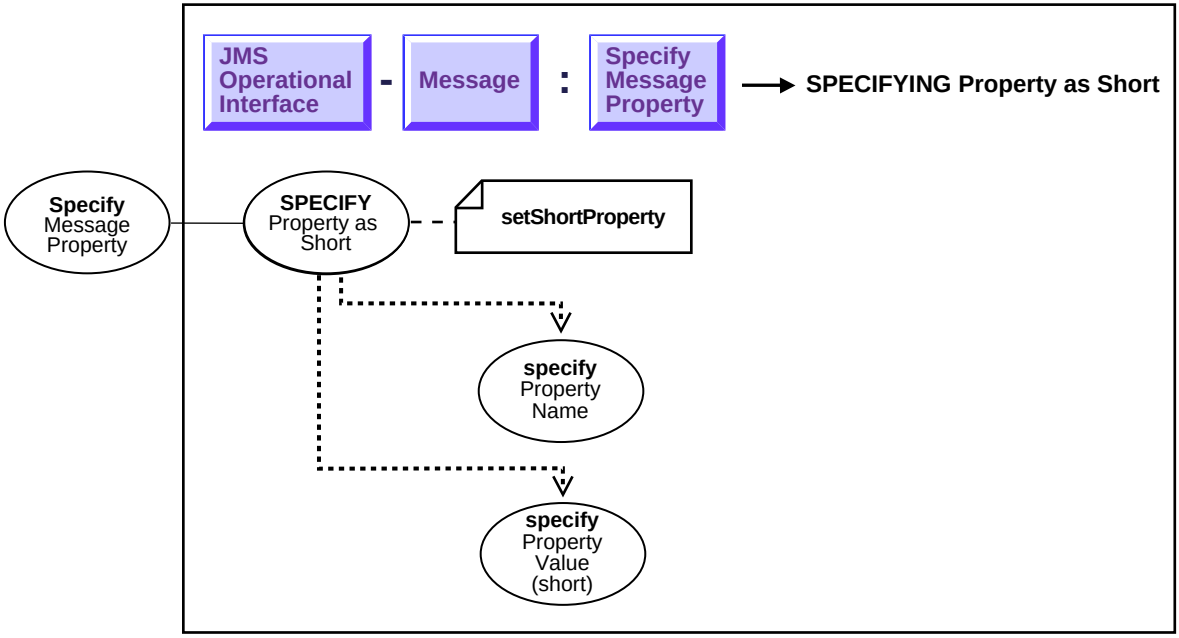
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Short

Figure 16–23 Use Case Diagram: Specify Message Property as Short



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Short

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setShortProperty

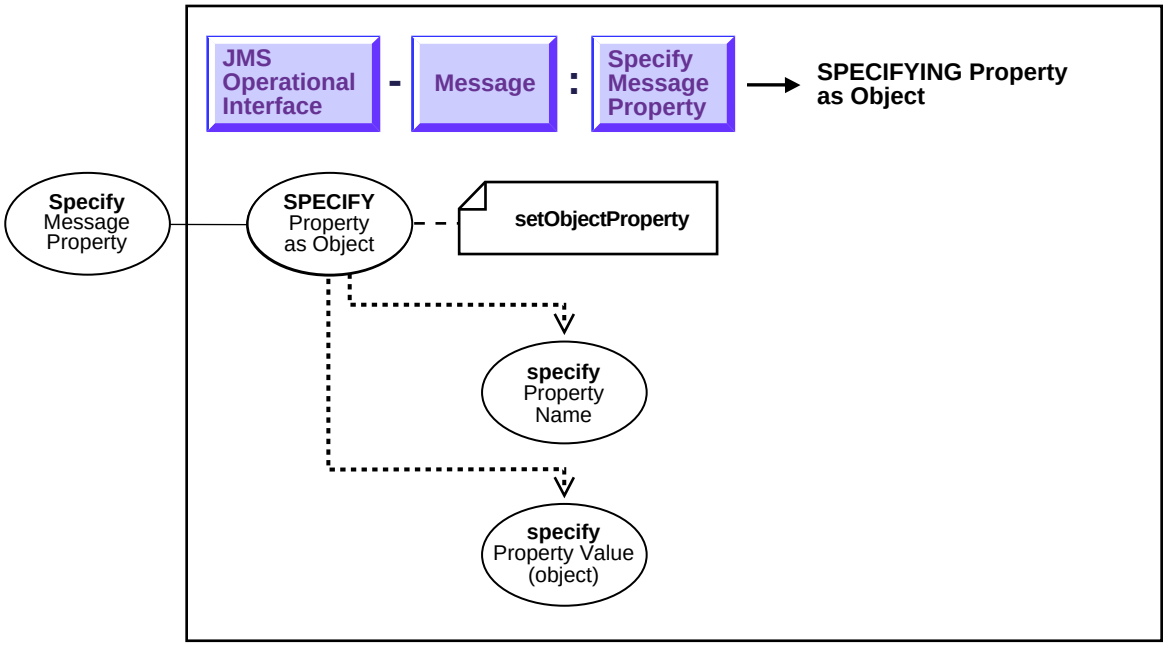
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying Message Property as Object

Figure 16–24 Use Case Diagram: Specify Message Property as Object



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify Message Property as Object

Usage Notes

Only objectified primitive values supported - Boolean, Byte, Short, Integer, Long, Float, Double and String.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsMessage.setObjectProperty

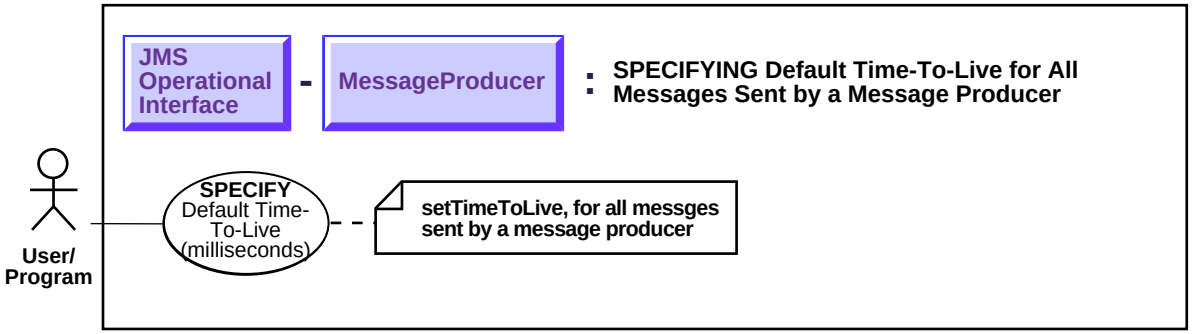
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Setting Default TimeToLive for All Messages Sent by a Message Producer

Figure 16–25 Use Case Diagram: Set Default TimeToLive for All Messages Sent by a MessageProducer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Set Default TimeToLive for All Messages Sent by a Message Producer

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).

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsProducer.setTimeToLive

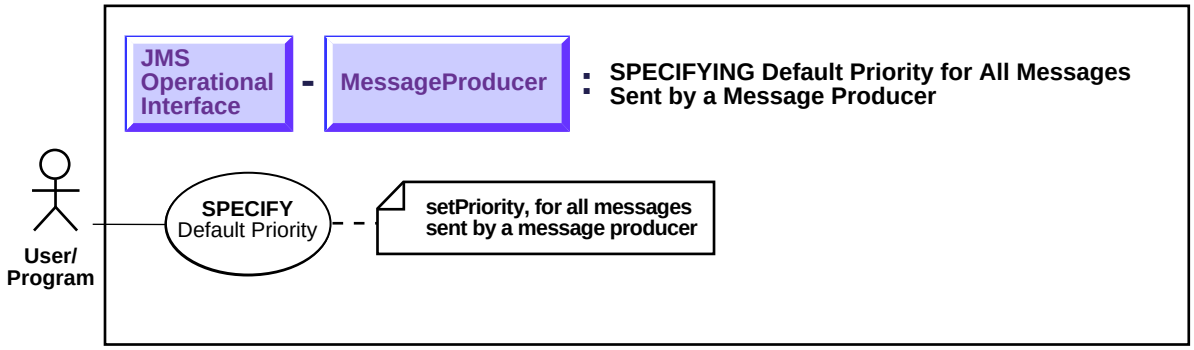
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
/* 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 Message Producer

Figure 16–26 Use Case Diagram: Set Default Priority for All Messages Sent by a Message Producer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Set Default Priority for All Messages Sent by a Message Producer

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, it overrides the producer's default value set by this method.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsProducer.setPriority

Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

Example 1

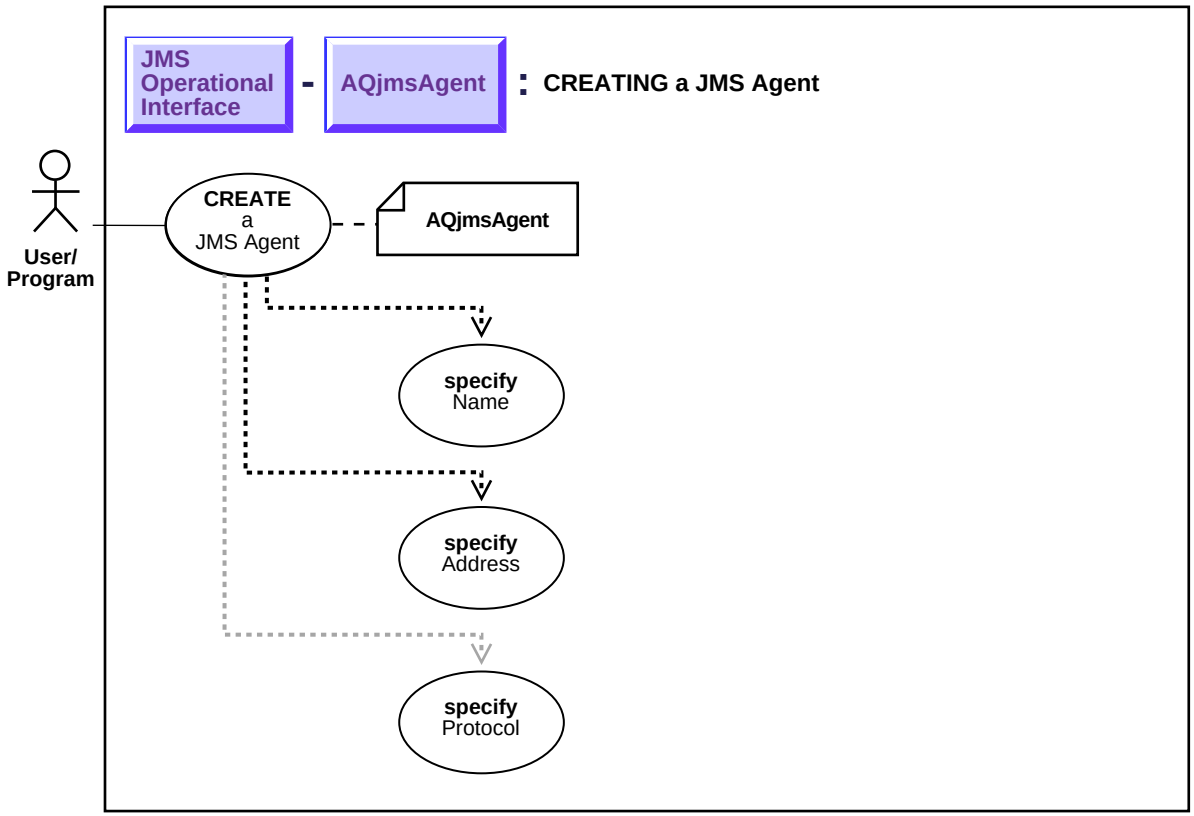
```
/* Set default priority value to 2 for all messages sent by the QueueSender*/
QueueSender sender;
sender.setPriority(2);
```

Example 2

```
/* Set default priority value to 2 for all messages sent by the TopicPublisher*/
TopicPublisher publisher;
publisher.setPriority(1);
```

Creating an AQjms Agent

Figure 16–27 Use Case Diagram: Create an AQjmsAgent



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Create an AQjms Agent

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsAgent

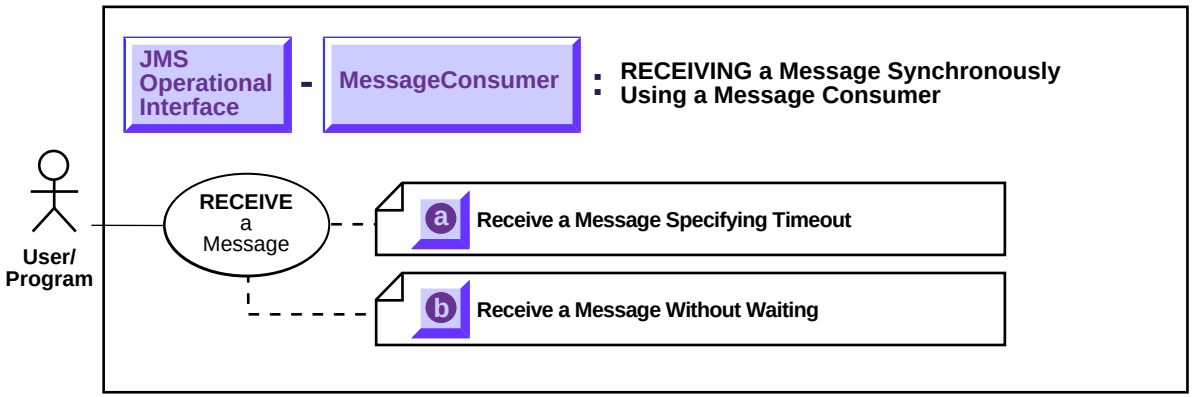
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Two Ways of Receiving a Message Synchronously Using a Message Consumer

Figure 16–28 Use Case Diagram: Two Ways to Receive a Message Synchronously Using a Message Consumer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Two Ways to Receive a Message Synchronously Using a Message Consumer

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsConsumer.receive

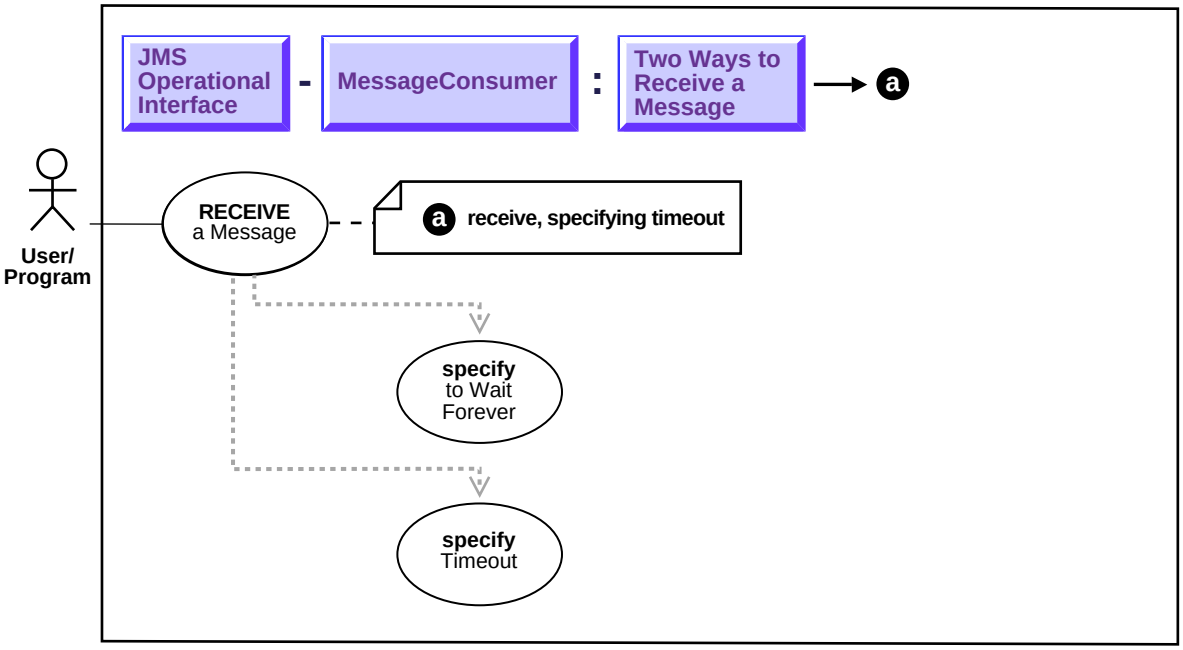
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Receiving a Message Using a Message Consumer by Specifying Timeout

Figure 16–29 Use Case Diagram: Receive a Message Using a Message Consumer by Specifying Timeout



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Receive a Message Using a Message Consumer by Specifying Timeout

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsConsumer.receive

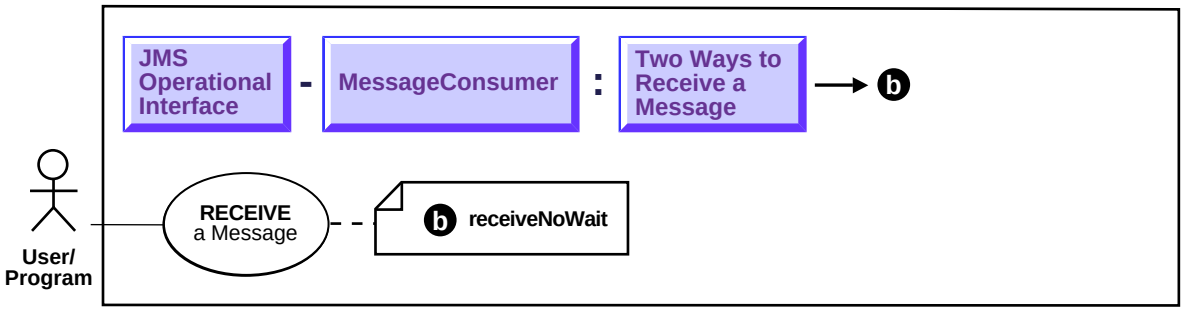
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
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);
```

Receiving a Message Using a Message Consumer Without Waiting

Figure 16–30 Use Case Diagram: Receive a Message Using a Message Consumer Without Waiting



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Receive a Message Using a Message Consumer Without Waiting

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsConsumer.receiveNoWait

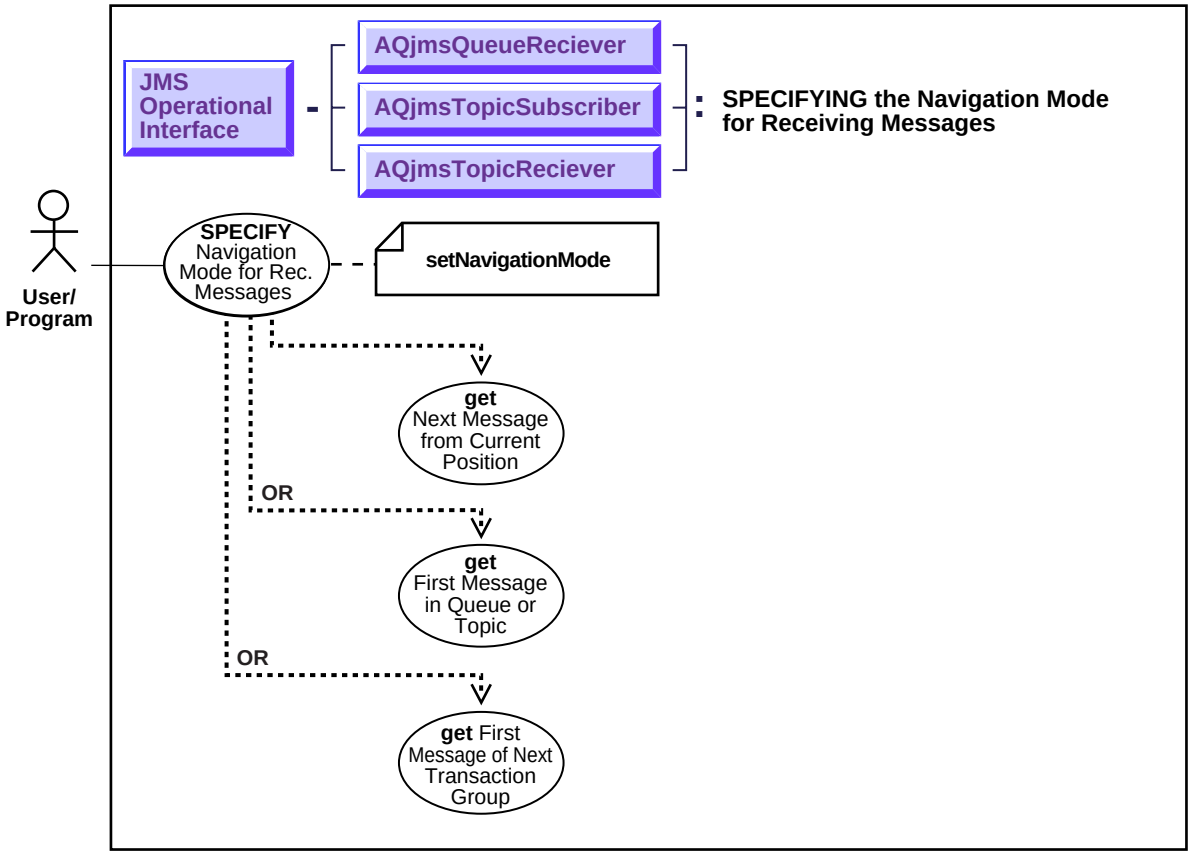
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Specifying the Navigation Mode for Receiving Messages

Figure 16–31 Use Case Diagram: Specify the Navigation Mode for Receiving Messages



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify the navigation mode for receiving messages.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsQueueReceiver.setNavigationMode, AQjmsTopicReceiver.setNavigationMode, AQjmsTopicSubscriber.setNavigationMode

Examples

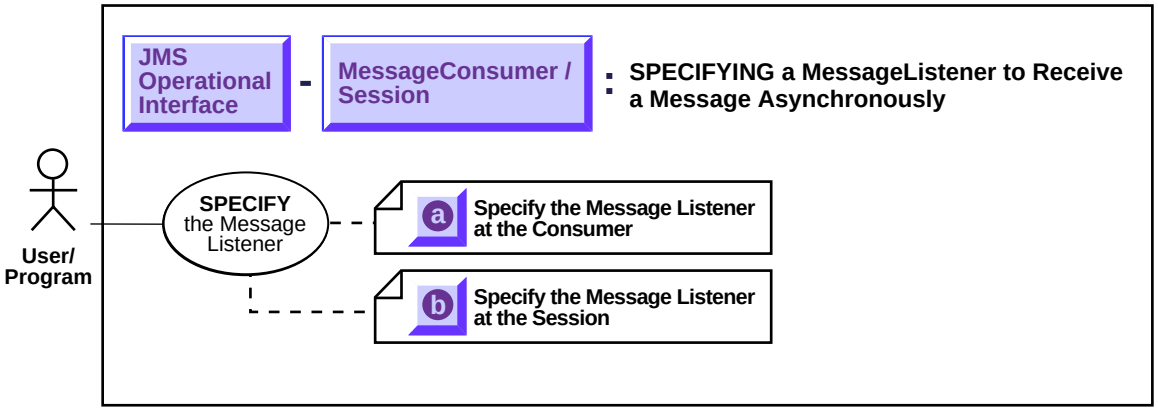
See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
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();
```

Two Ways of Specifying a Message Listener to Receive a Message Asynchronously

Figure 16–32 Use Case Diagram: Two Ways to Specify a Message Listener to Receive a Message Asynchronously



To refer to the table of all basic operations having to do with the Operational Interface see:

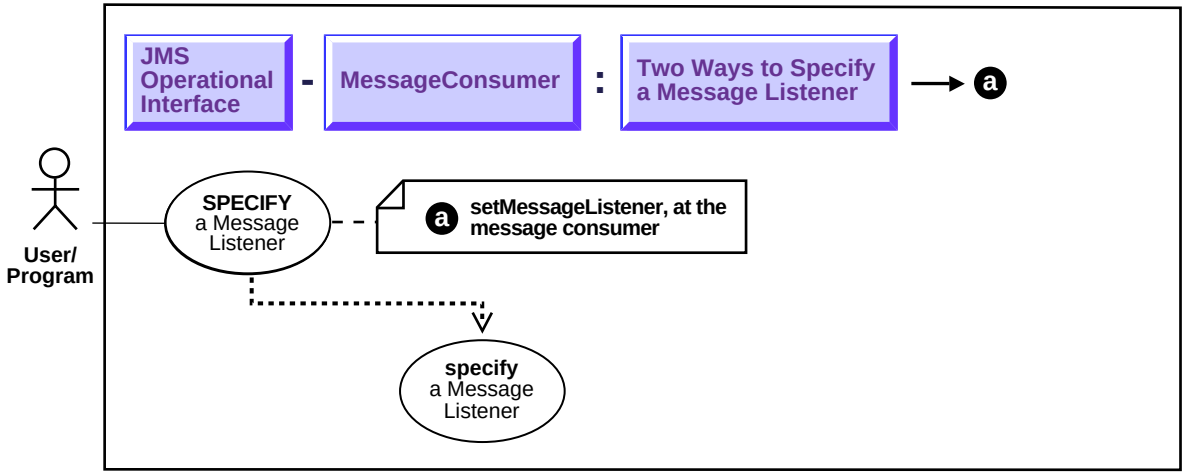
- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

There are two ways to specify a message listener to receive a message asynchronously:

- Specifying a Message Listener at the Message Consumer on page 16-61
- Specifying a Message Listener at the Session on page 16-64

Specifying a Message Listener at the Message Consumer

Figure 16–33 Use Case Diagram: Specify a Message Listener at the Message Consumer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify a Message Listener at the Message Consumer

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsConsumer.setMessageListener

Examples

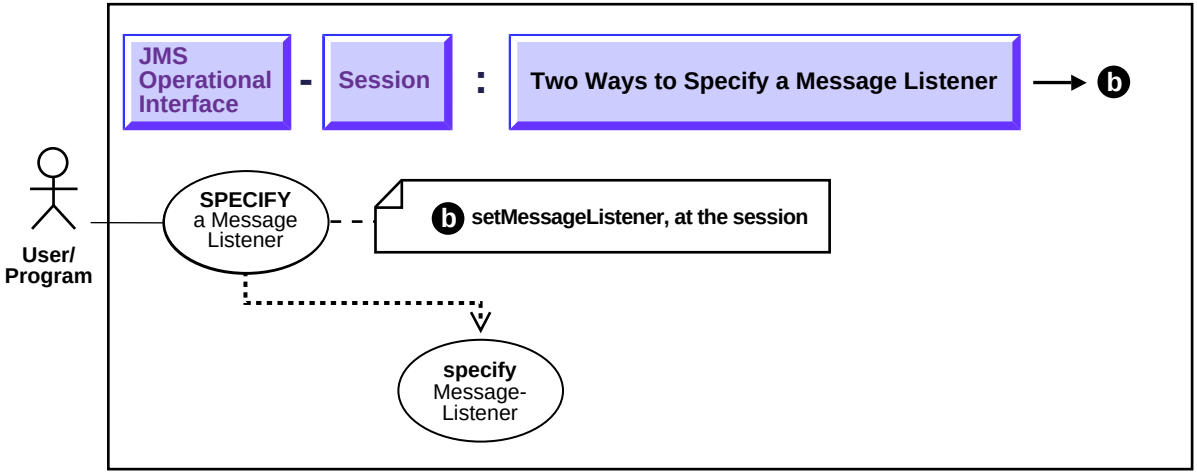
See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
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

Figure 16–34 Use Case Diagram: Specify a Message Listener at the Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify a Message Listener at the Session

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.setMessageListener

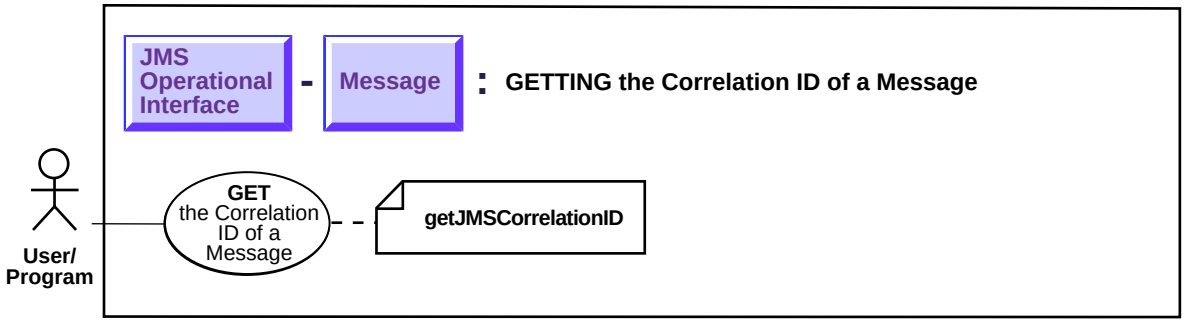
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Correlation ID of a Message

Figure 16–35 Use Case Diagram: Get the Correlation ID of a Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Correlation ID of a Message

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getJMSCorrelationID

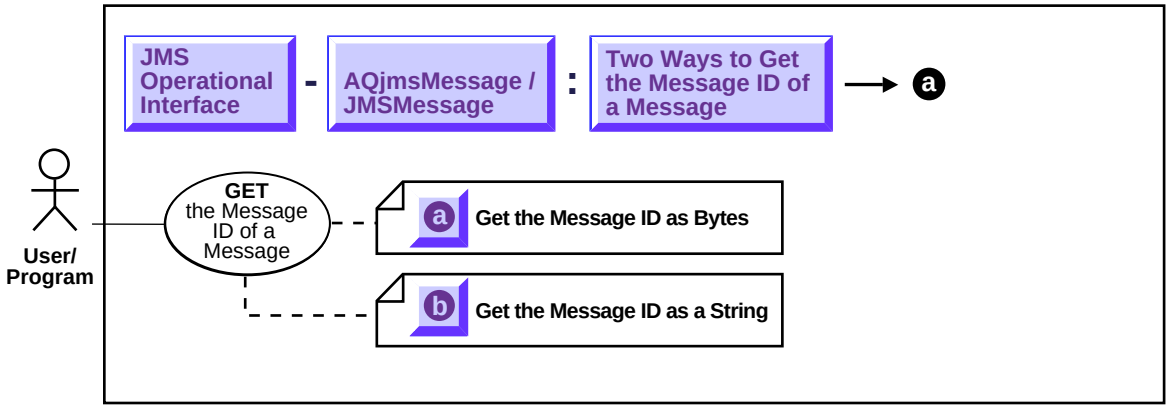
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Two Ways of Getting the Message ID of a Message

Figure 16–36 Use Case Diagram: Two Ways to Get the Message ID of a Message



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

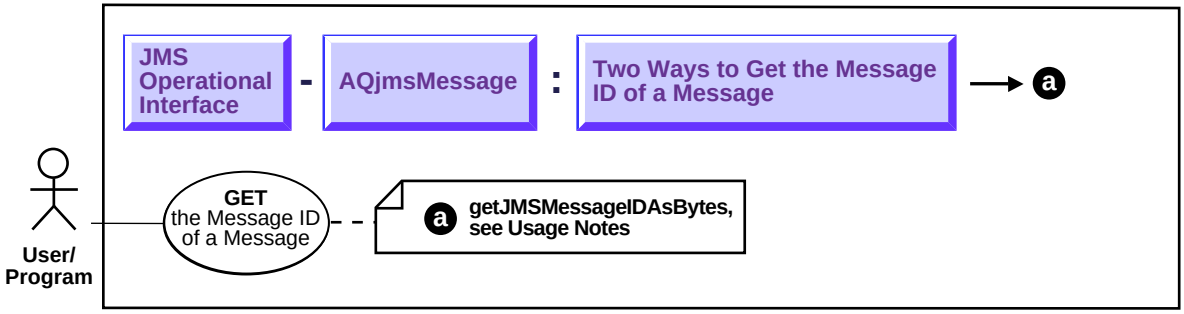
There are two ways to get the message id of a message.

a. By using the bytes — see Getting the Message ID of a Message as Bytes on page 16-68

b. By using the string — see Getting the Message ID of a Message as a String on page 16-69

Getting the Message ID of a Message as Bytes

Figure 16–37 Use Case Diagram: Get the Message ID of a Message as Bytes



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message ID of a Message as Bytes

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getJMSMessageID

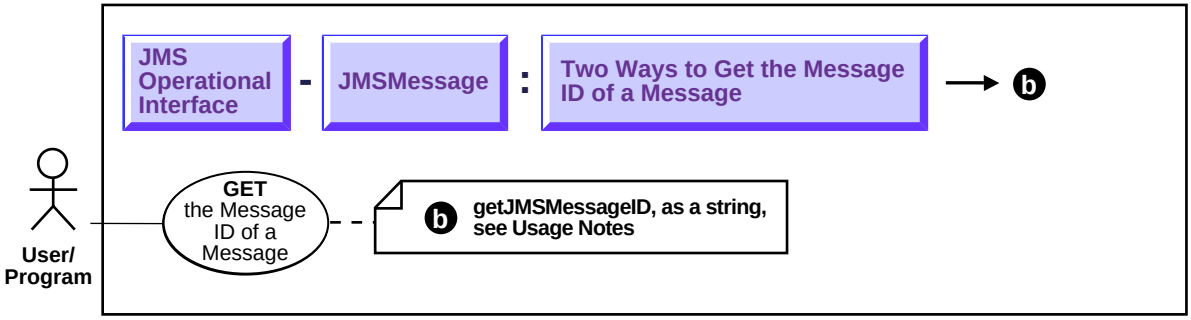
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message ID of a Message as a String

Figure 16–38 Use Case Diagram: Get the Message ID of a Message as a String



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message ID of a Message as String

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getJMSMessageID

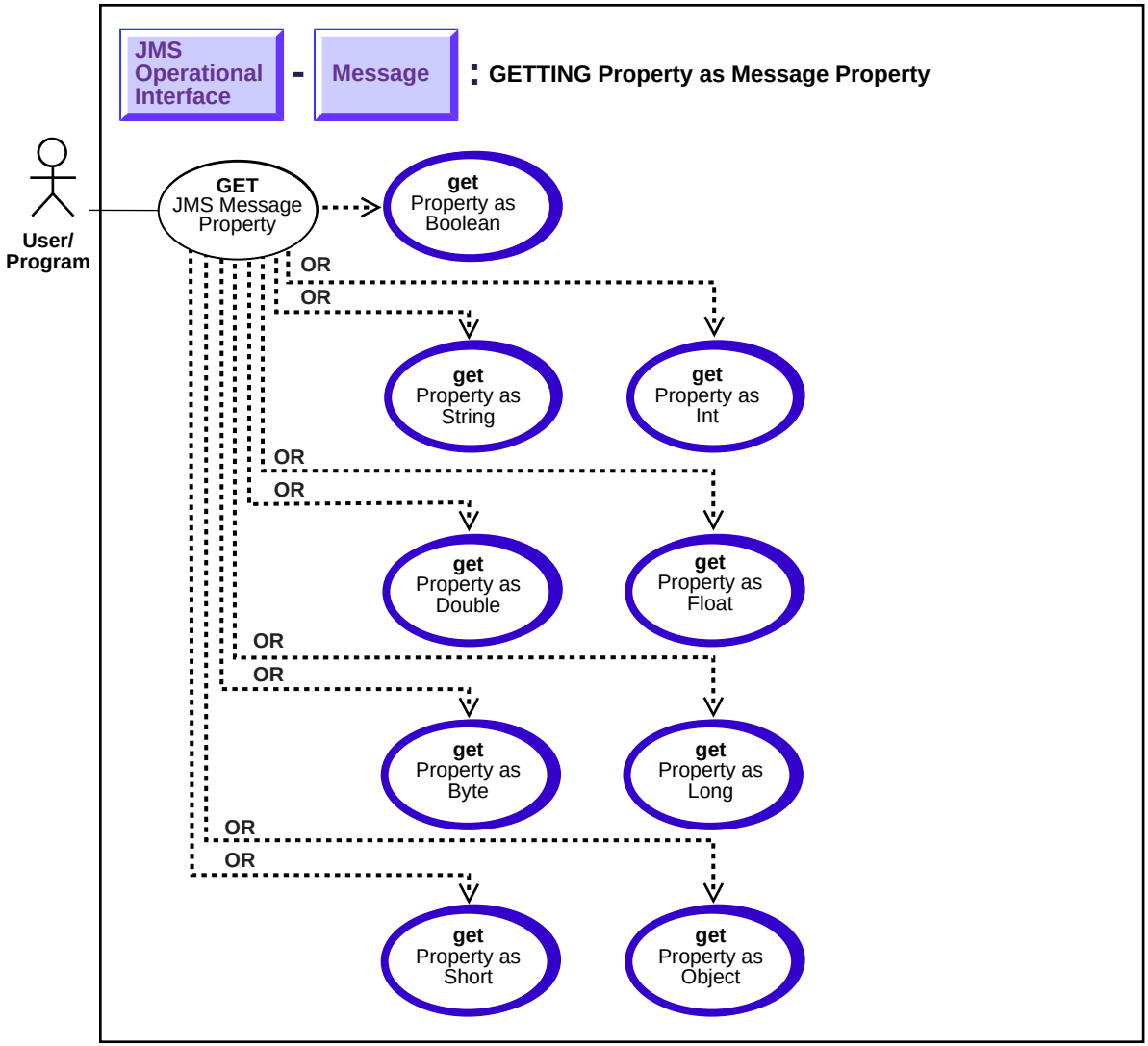
Examples

See Chapter 3, "AQ Programmatic Environments"for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the JMS Message Property

Figure 16–39 Use Case Diagram: Get the JMS Message Property



To refer to the table of all basic operations having to do with the Operational Interface see:

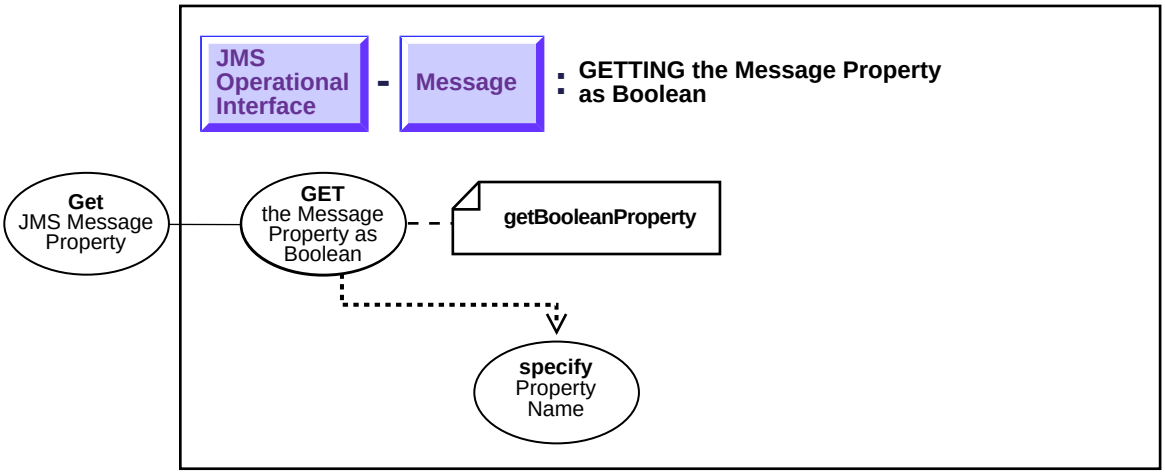
- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

There are several ways to get the JMS message property:

- Getting the Message Property as a Boolean on page 16-73
- Getting the Message Property as a String on page 16-74
- Getting the Message Property as Int on page 16-76
- Getting the Message Property as Double on page 16-78
- Getting the Message Property as Float on page 16-79
- Getting the Message Property as Byte on page 16-81
- Getting the Message Property as Long on page 16-82
- Getting the Message Property as Short on page 16-84
- Getting the Message Property as Object on page 16-85

Getting the Message Property as a Boolean

Figure 16–40 Use Case Diagram: Get the Message Property as a Boolean



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as a Boolean

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getBooleanProperty

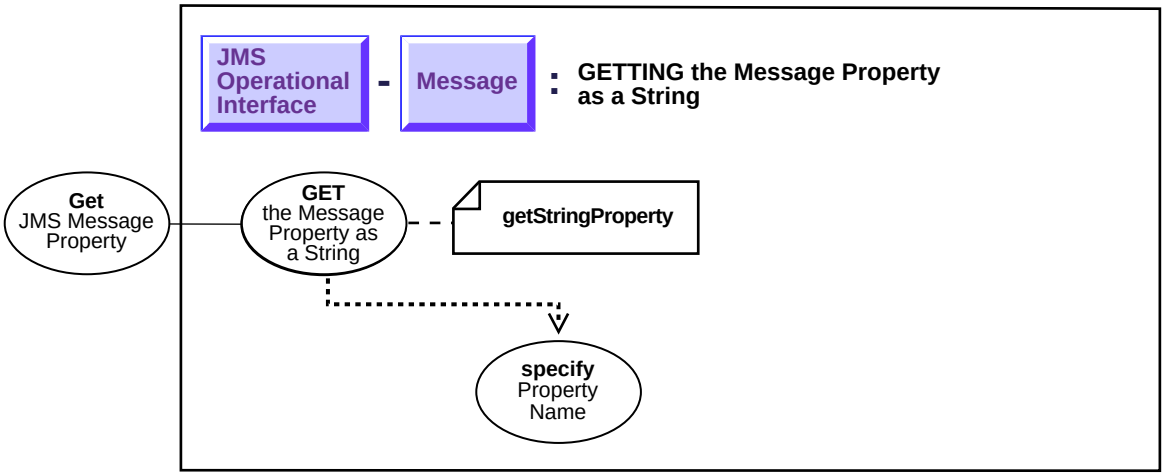
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as a String

Figure 16–41 Use Case Diagram: Get the Message Property as a String



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as a String

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getStringProperty

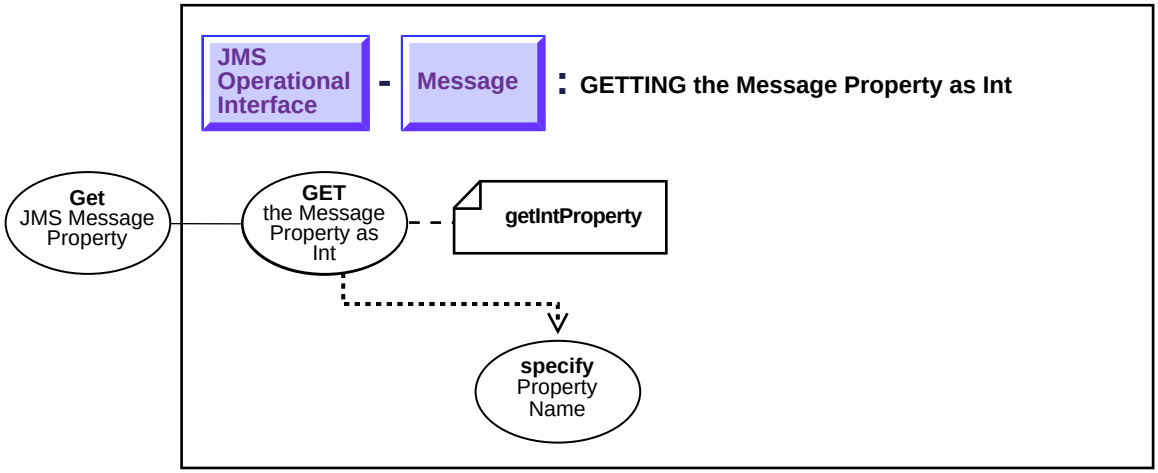
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
TextMessage message;

message.setStringProperty("JMS_OracleExcpQ", "scott.text_ecxcp_queue"); /*set
exception queue for message*/

message.setStringProperty("color", "red"); /*set user-defined property - color
*/
```

Getting the Message Property as Int

Figure 16–42 Use Case Diagram: Get the Message Property as Int



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Int

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getIntProperty

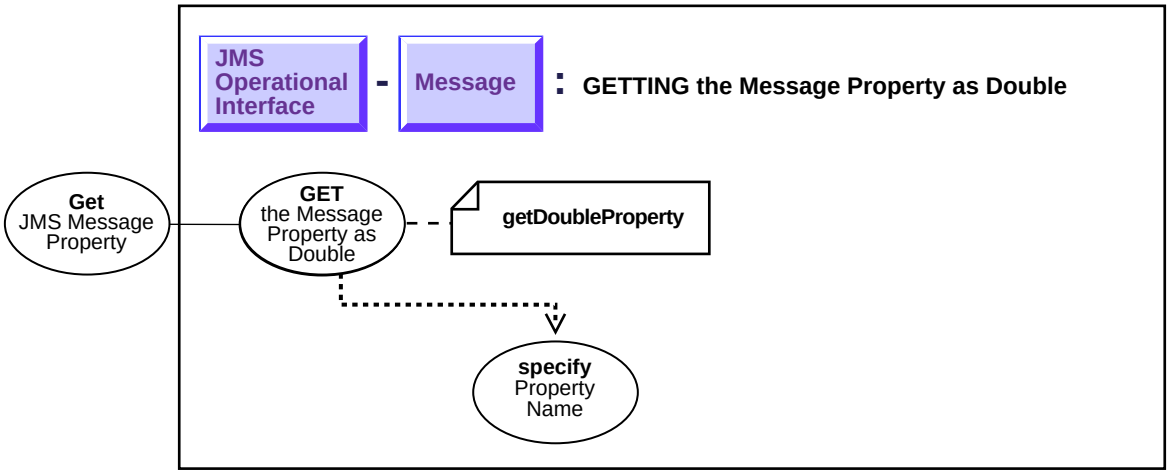
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
StreamMessage message;
message.setIntProperty("MMS_OracleDelay", 10); /*set message delay to 10
seconds*/

message.setIntProperty("empid", 1000); /*set user-defined property - empId*/
```

Getting the Message Property as Double

Figure 16–43 Use Case Diagram: Get the Message Property as Double



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Double

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getDoubleProperty

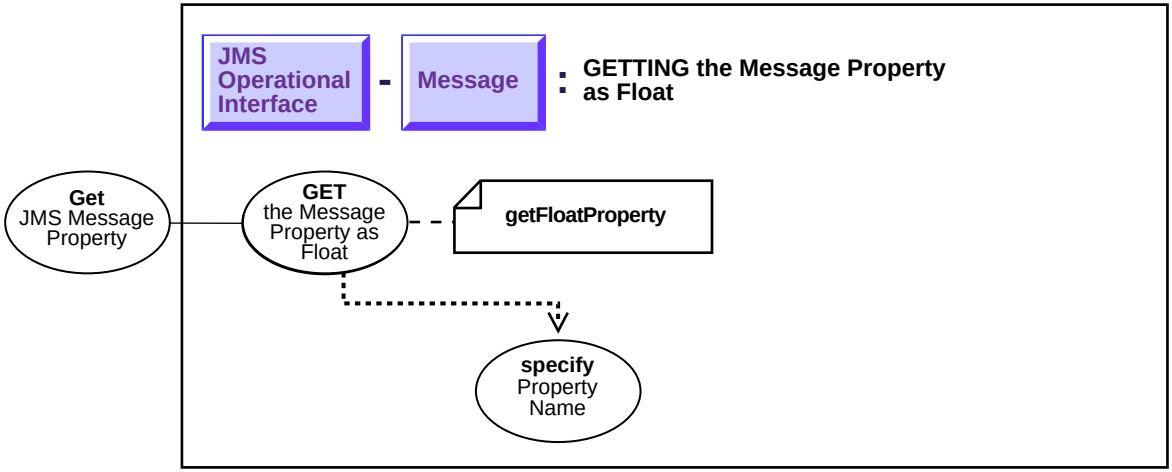
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as Float

Figure 16–44 Use Case Diagram: Get the Message Property as Double



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Float

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getFloatProperty

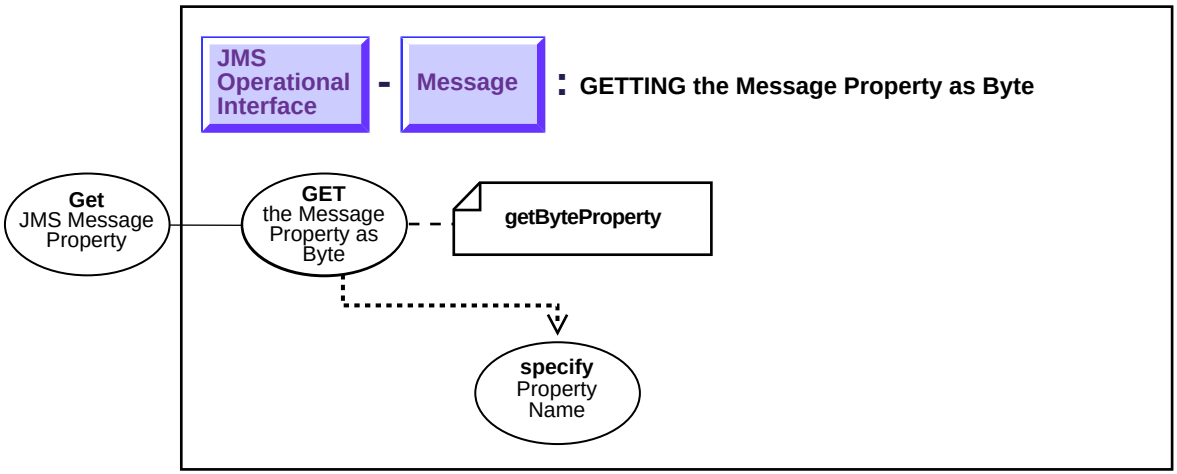
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as Byte

Figure 16–45 Use Case Diagram: Get the Message Property as Double



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Byte

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getByteProperty

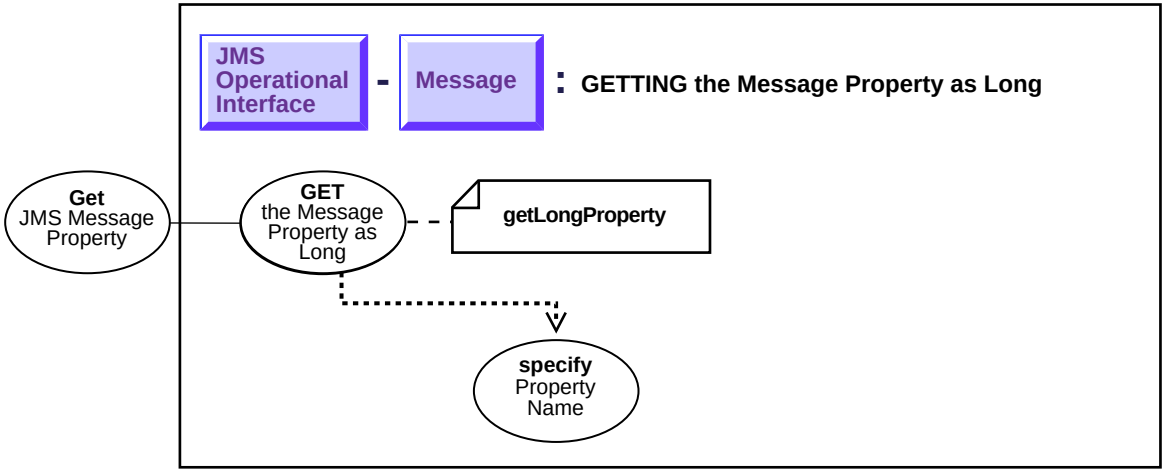
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as Long

Figure 16–46 Use Case Diagram: Get the Message Property as Double



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Long.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getLongProperty

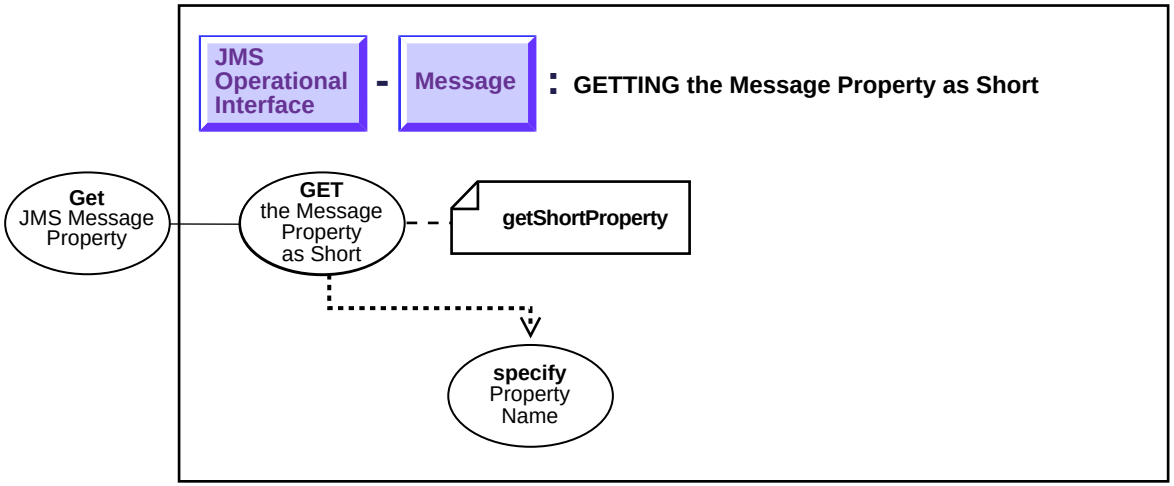
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as Short

Figure 16–47 Use Case Diagram: Get the Message Property as Short



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Short

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getShortProperty

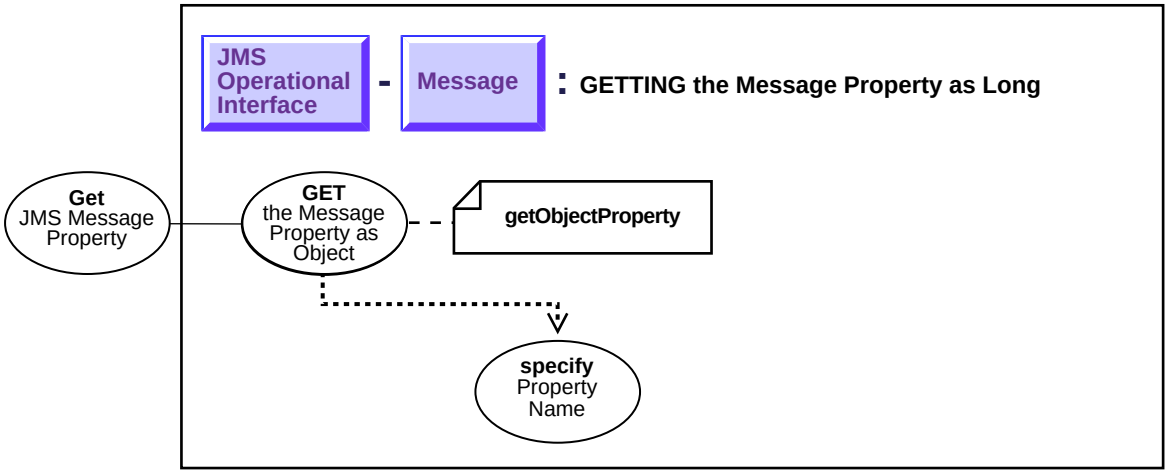
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Message Property as Object

Figure 16–48 Use Case Diagram: Get the Message Property as Object



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Message Property as Object

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsMessage.getObjectProperty

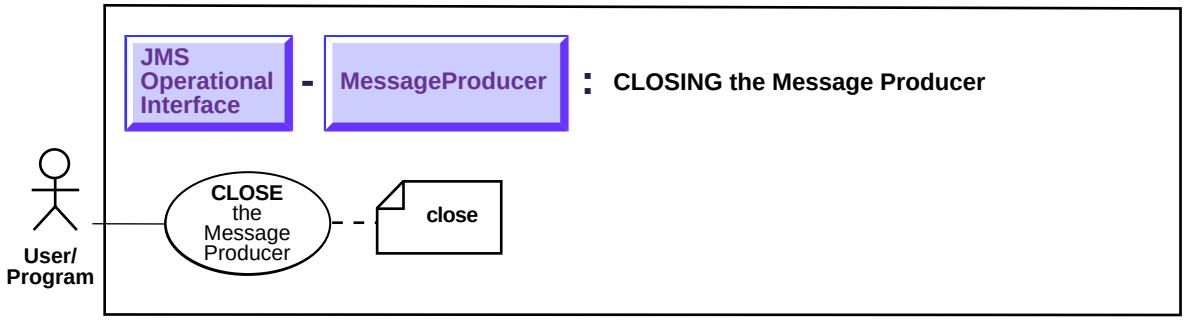
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

```
TextMessage message;
message.setObjectProperty("empid", new Integer(1000);
```

Closing a Message Producer

Figure 16–49 Use Case Diagram: Close a Message Producer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Close a Message Producer

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsProducer.close

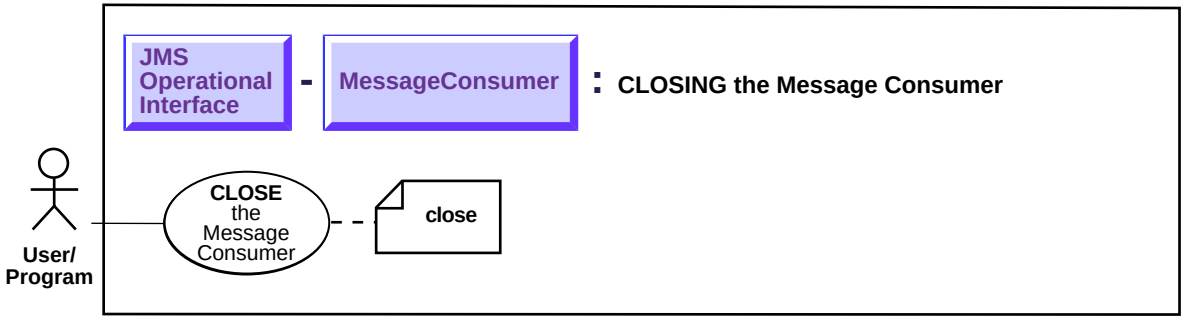
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Closing a Message Consumer

Figure 16–50 Use Case Diagram: Close a Message Consumer



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Close a Message Consumer

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsConsumer.close

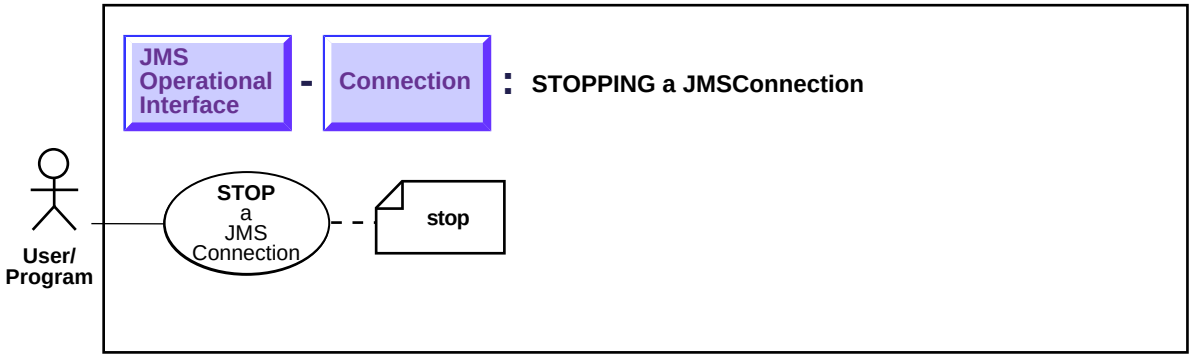
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Stopping a JMS Connection

Figure 16–51 Use Case Diagram: Stop a JMS Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Stop a JMS Connection

Usage Notes

This method is used to temporarily stop a Connection's delivery of incoming messages.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsConnection.stop

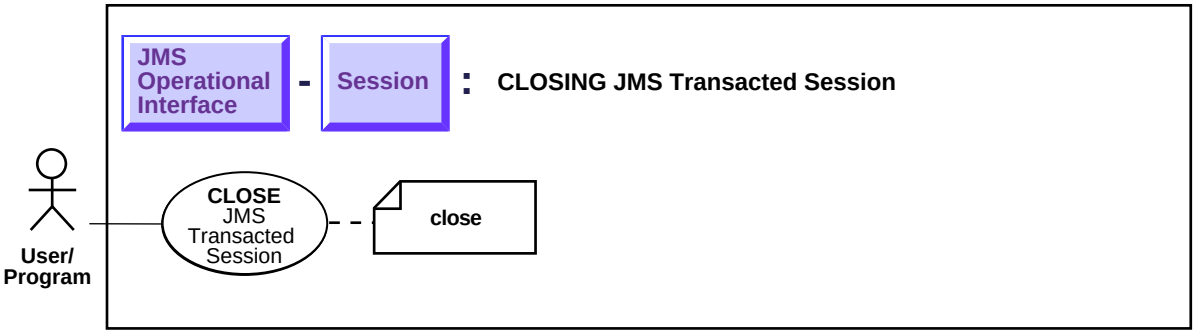
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Closing a JMS Session

Figure 16–52 Use Case Diagram: Close a JMS Transacted Session



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Close a JMS Session

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsSession.close

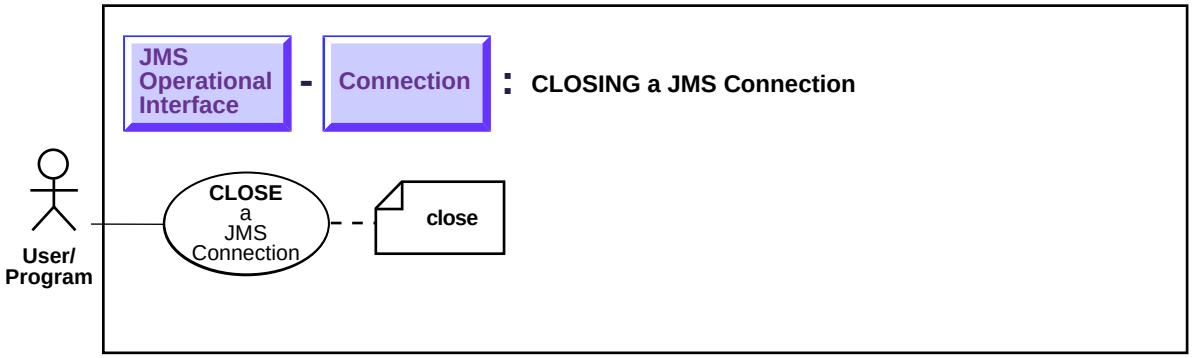
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Closing a JMS Connection

Figure 16–53 Use Case Diagram: Close a JMS Connection



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Close a JMS Connection

Usage Notes

This method closes the connection and releases all resources allocated on behalf of the connection. Since 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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsConnection.close

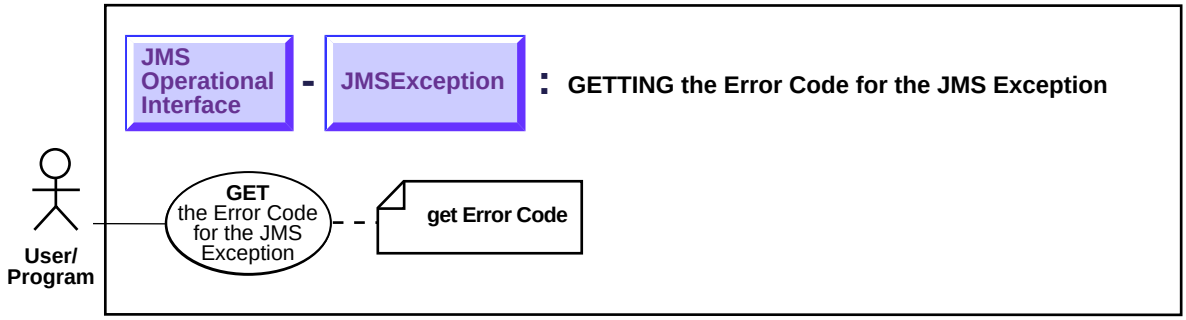
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Error Code for the JMS Exception

Figure 16–54 Use Case Diagram: Get the Error Code for the JMS Exception



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Error Code for the JMS Exception

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsException.getErrorCode

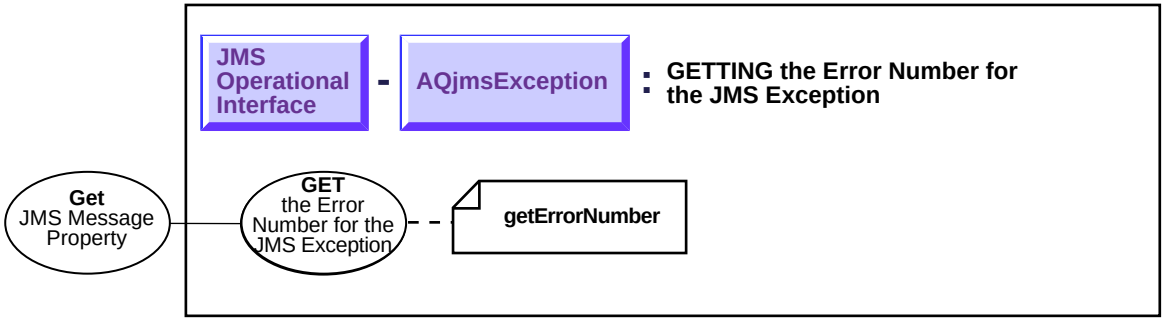
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Error Number for the JMS Exception

Figure 16–55 Use Case Diagram: Get the Error Number for the JMS Exception



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Error Number for the JMS Exception

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsException.getErrorNumber

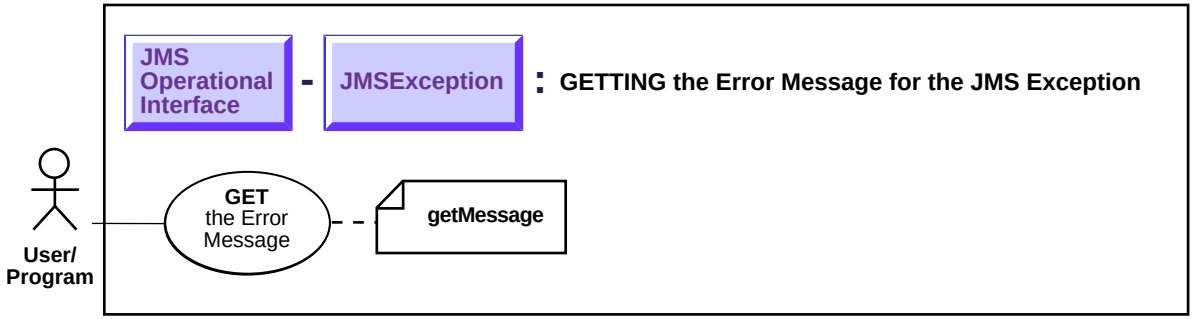
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Error Message for the JMS Exception

Figure 16–56 Use Case Diagram: Get the Error Message for the JMS Exception



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Error Message for the JMS Exception

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference* oracle.jms, AQjmsException.getMessage

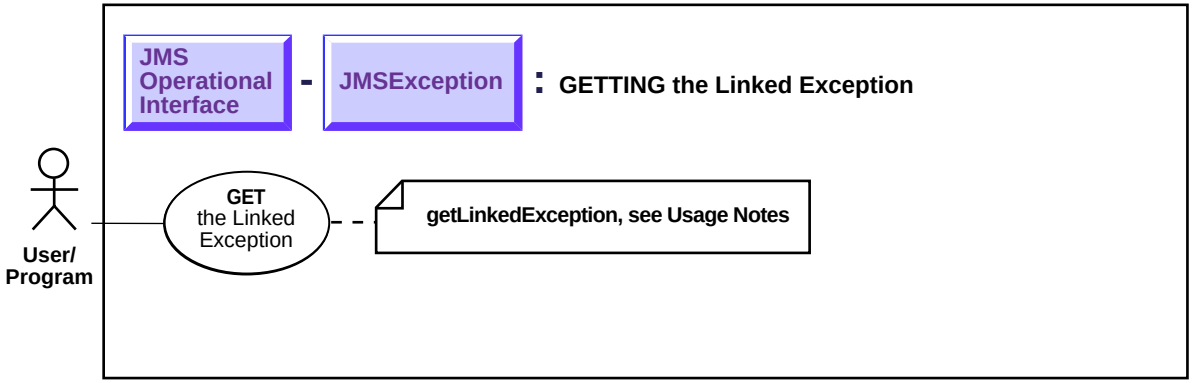
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Getting the Exception Linked to the JMS Exception

Figure 16–57 Use Case Diagram: Get the Exception Linked to the JMS Exception



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Exception Linked to the JMS Exception

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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsException.getLinkedException

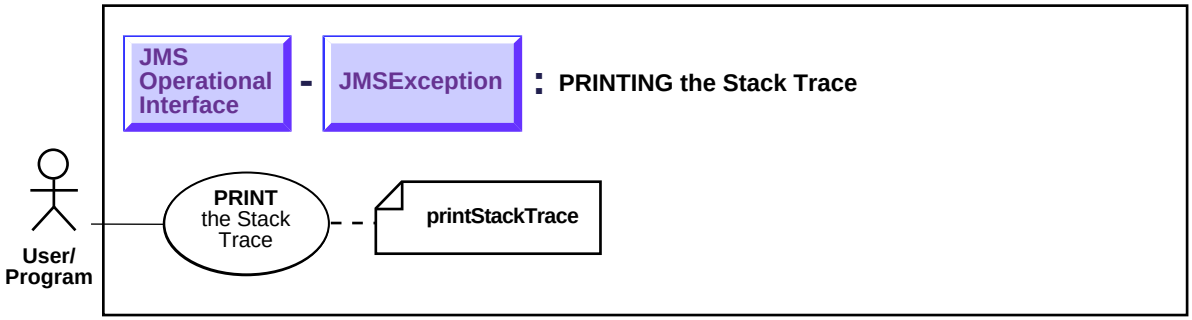
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Printing the Stack Trace for the JMS Exception

Figure 16–58 Use Case Diagram: Print the Stack Trace for the JMS Exception



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Print the Stack Trace for the JMS Exception

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms, AQjmsException.printStackTrace

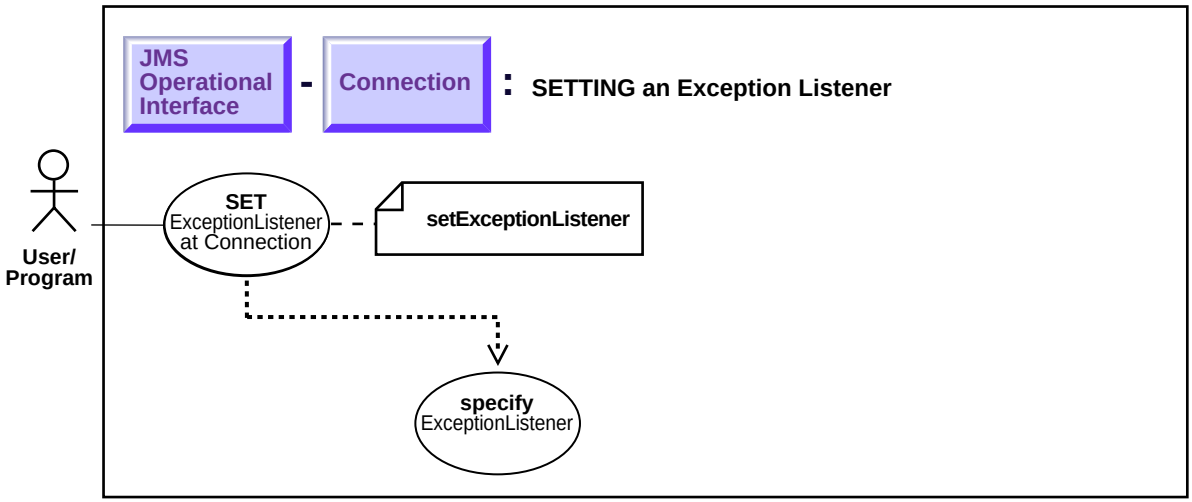
Examples

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment.

- No example is provided with this release.

Setting the Exception Listener

Figure 16–59 Use Case Diagram: Setting the Exception Listener



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify an Exception Listener for the connection.

Usage Notes

If a serious problem is detected for the connection, the connection's ExceptionListener, if one has been registered, will be informed. This is done 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.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

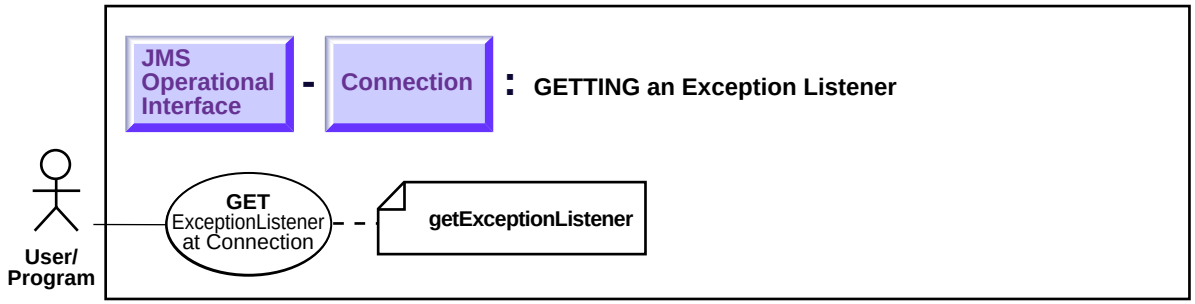
- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms AQjmsConnection.setExceptionListener

Examples

```
//register an exception listener
Connection jms_connection;
jms_connection.setExceptionListener(
    new ExceptionListener() {
        public void onException (JMSException jmsException) {
            System.out.println("JMS-EXCEPTION: " + jmsException.toString());
        }
    };
  );
```

Getting the Exception Listener

Figure 16–60 Use Case Diagram: Getting the Exception Listener



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the Exception Listener for the connection.

Usage Notes

Not applicable.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

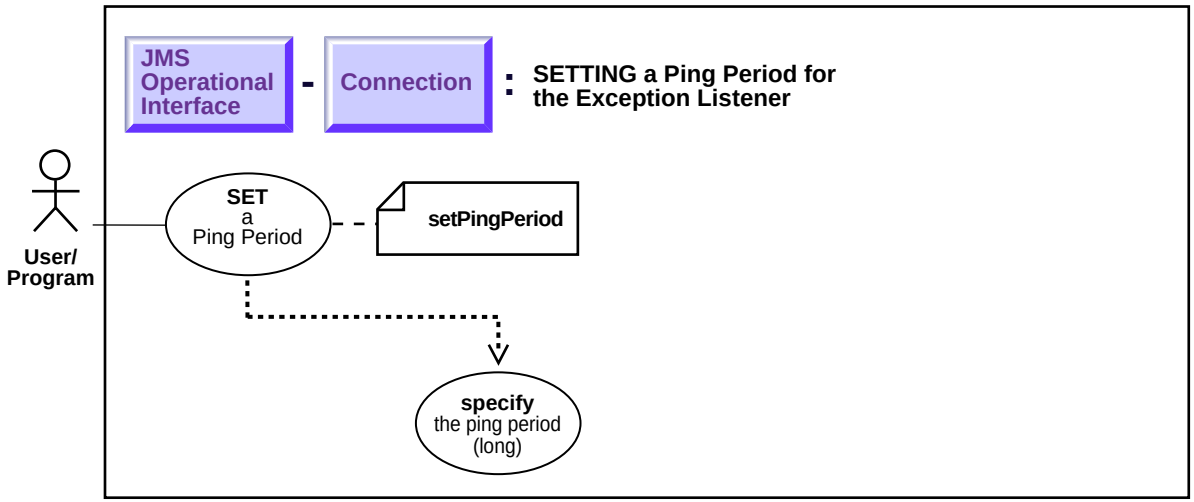
- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms AQjmsConnection.getExceptionListener

Examples

```
//Get the exception listener
Connection jms_connection;
ExceptionListener el = jms_connection.getExceptionListener();
```

Setting the Ping Period for the Exception Listener

Figure 16–61 Use Case Diagram: Setting the Ping Period for the Exception Listener



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Specify the ping period for the Exception Listener.

Usage Notes

If an exception listener is set for the connection, the connection pings the database periodically to ensure that the database is accessible. The period is specified in milliseconds. The default value is 2 minutes. If an exception listener is not set for the connection, the database is not pinged. This method can be called before or after the exception listener is set.

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

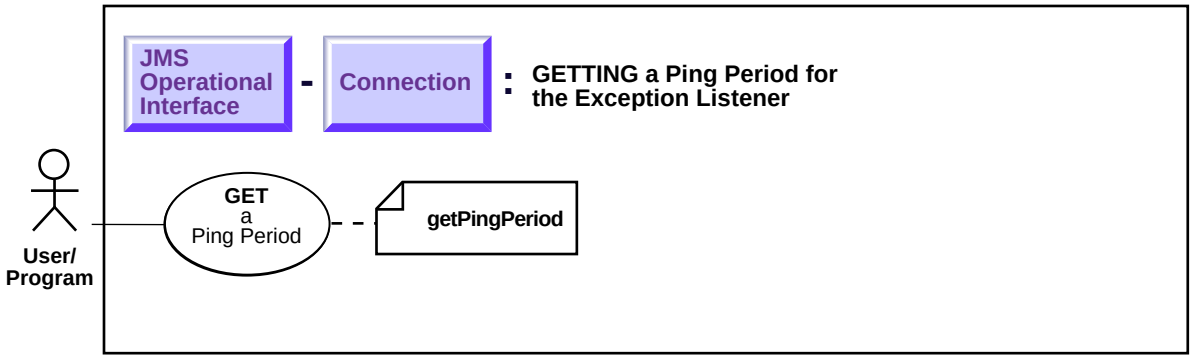
- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms AQjmsConnection.setPingPeriod

Examples

```
//set the ping period to 4 minutes
Connection jms_connection;
jms_connection.setPingPeriod(4*60*1000);
```

Getting the Ping Period for the Exception Listener

Figure 16–62 Use Case Diagram: Getting the Ping Period for the Exception Listener



To refer to the table of all basic operations having to do with the Operational Interface see:

- "Use Case Model: Operational Interface — Basic Operations" on page 14-2

Purpose

Get the ping period for the Exception Listener.

Usage Notes

If an exception listener is set for the connection, the connection pings the database periodically to ensure that the database is accessible. The period is specified in milliseconds. The default value is 2 minutes. If an exception listener is not set for the connection, the database is not pinged. This method will return the value of the period set by the last call to `setPingPeriod`. If `setPingPeriod` was never called, then the default value is returned

Syntax

See Chapter 3, "AQ Programmatic Environments" for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- See Java (JDBC): *Oracle9i Supplied Java Packages Reference*, oracle.jms AQjmsConnection.getPingPeriod

Examples

```
//get the ping period
Connection jms_connection;
long pp = jms_connection.getPingPeriod();
```

17

Internet Access to Advanced Queuing

You can perform AQ operations over the Internet by using the Internet Data Access Presentation (IDAP), which defines message structure using XML. An IDAP-structured message is transmitted over the Internet using HTTP or SMTP.

This chapter discusses the following topics:

- Overview of Advanced Queuing Operations Over the Internet
- The Internet Data Access Presentation (IDAP)
- IDAP and AQ XML Schemas
- Deploying the AQ XML Servlet
- Using HTTP to Access the AQ XML Servlet
- Using HTTP and HTTPS for Advanced Queuing Propagation
- Using SMTP to Access the AQ Servlet
- Customizing the AQ Servlet

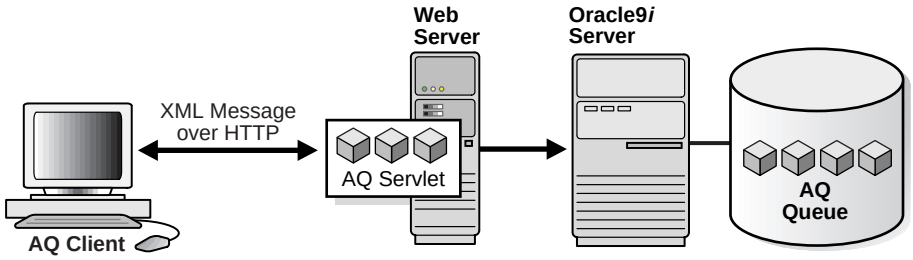
Overview of Advanced Queuing Operations Over the Internet

Figure 17–1 shows the architecture for performing AQ operations over HTTP. The major components are:

- The AQ client program
- The Web server / ServletRunner hosting the AQ servlet
- The Oracle database server

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

Figure 17–1 Architecture for Performing AQ Operations Using HTTP



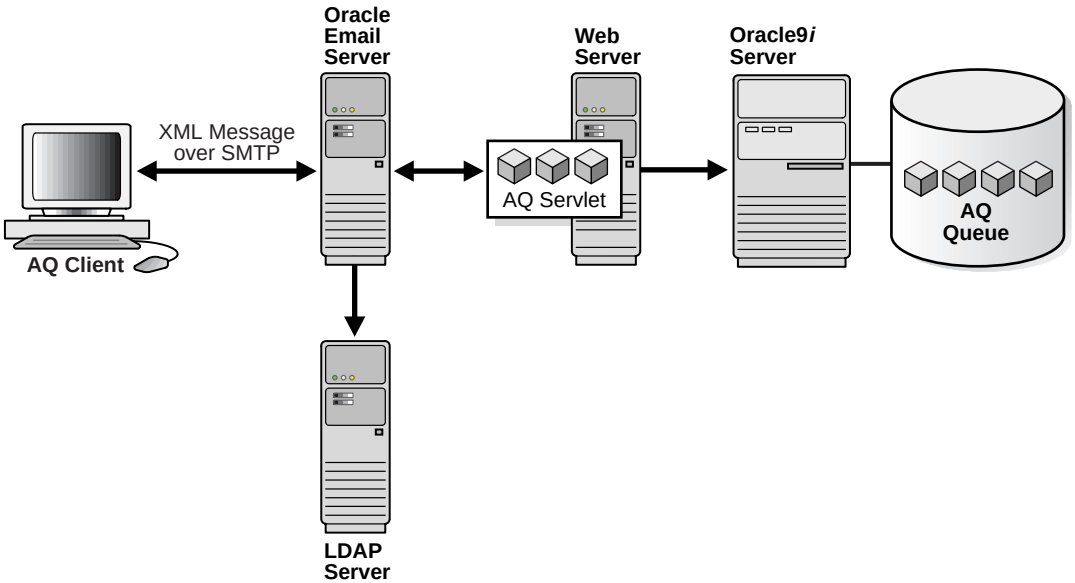
See "Using HTTP to Access the AQ XML Servlet" and "Using HTTP and HTTPS for Advanced Queuing Propagation" on page 17-60 for details.

Figure 17–2 shows additional components in the architecture for sending AQ messages over SMTP:

- Email server
- LDAP server (Oracle Internet Directory)

The email server verifies client signatures using certificates stored in LDAP and then routes the request to the AQ servlet.

Figure 17–2 Architecture for Performing AQ Operations Using SMTP



See "Using SMTP to Access the AQ Servlet" on page 17-63 for more details.

The Internet Data Access Presentation (IDAP)

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

- All protocol tags are scoped to the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes.
- The receiver processes IDAP messages that have correct namespaces; for requests with incorrect namespaces, the receiver returns an invalid request error.
- The receiver processes IDAP messages without namespaces as though they had the correct namespaces, if the context is valid.
- The IDAP namespace has the value `http://ns.oracle.com/AQ/schemas/envelope`
- An XML document forming the request of an IDAP invocation does not require the use of an XML DTD or a schema.

IDAP Message Structure

IDAP structures a message request or response as follows:

- IDAP envelope (the root or top element in an XML tree))
- IDAP header (first element under the root)
- IDAP body (the AQ XML document)

The IDAP Envelope

The tag of this root element is `IDAP:Envelope`. IDAP defines a global attribute `IDAP:encodingStyle` that indicates serialization rules used instead of those described by the IDAP specification. This attribute may appear on any element and is scoped to that element and all child elements not themselves containing such an attribute. Omitting `IDAP:encodingStyle` means that type specification has been followed (unless overridden by a parent element).

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

IDAP Headers

The tag of this first element under the root is `IDAP:Header`. An IDAP header passes necessary information, such as the transaction ID, with the request. The header is encoded as a child of the `IDAP:Envelope` XML element. Headers are identified by the name element and are namespace-qualified. A header entry is encoded as an embedded element.

The IDAP Body

The IDAP body, tagged `IDAP: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 `IDAP: Fault`, indicating information about an error.

For performing AQ operations, the IDAP body must contain an AQ XML document. The AQ XML document has the namespace `http://ns.oracle.com/AQ/schemas/access`

IDAP 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.

In the case of SMTP (email), the method invocation can be done by the filter interface of the email server, which invokes a Java method with the email-message-body as argument. This results in remote invocation of the `POST` method on the AQ servlet. The response is emailed directly to the recipient specified in the reply of the message. The response header can contain SMTP-protocol-related headers also.

HTTP Headers

The `POST` method within the HTTP request header performs the IDAP method invocation. The request should include the header `IDAPMethodName,` 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:

```
IDAPMethodName: 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 `IDAP:Body` part of the payload.

Method Invocation Body

IDAP method invocation consists of a method request and optionally a method response. The IDAP 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 IDAP payload in the rest of this chapter.

The IDAP payload is defined as follows:

- The IDAP root element is the top element in the XML tree.
- The IDAP 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 `IDAP:Body` element. This request can be one of the 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:

a. The HTTP infrastructure on the receiving site is able to receive and process the request.

b. The HTTP infrastructure on the receiving site cannot receive and process the request.

c. The IDAP 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.

d. The IDAP 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 (a), the HTTP infrastructure passes the headers and body to the IDAP infrastructure. In (b), the result is an HTTP response containing an HTTP error in the status field and no XML body. In (c), the result of the method request consists of a response or error. In (d), the result of the method is an error that prevented the dispatching infrastructure on the receiving side from successful completion. In (c) and (d), additional message headers may for extensibility again be present in the results of the request.

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`. An IDAP result indicates success and an error indicates failure. The method response will never contain both a result and an error.

AQ XML Documents

The body of an IDAP message is an AQ XML document, which 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:** AQ Internet Access is supported only for 8.1-style queues. 8.0-style queues cannot be accessed using IDAP.

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

`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. ■ `message_header`	Each message consists of a `message_header` and `message_payload` `message_id`—unique identifier of the message, supplied during dequeue `correlation`—correlation identifier of the message

Table 17–1 Client Requests for Enqueue—Arguments and Attributes for AQXmlSend and AQXmlPublish

Argument	Attribute
	`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 `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_payload`**	`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. 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

Message Payloads

AQ supports messages of the following types:

- RAW
- Oracle object (ADT)
- Java Messaging Service (JMS) types:
  - Text message
  - Map message
  - Bytes message
  - Object message

All these types of queues can be accessed using IDAP. If the queue holds messages in RAW, Oracle object, or JMS format, 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 above formats, they are converted to XML before being sent to the client.

The message payload type depends on the type of the queue on which the operation is being performed. A discussion of the queue types follows:

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

Oracle object (ADT) type queues For ADT 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 the *Oracle9i Application Developer's Guide - XML* for more details on mapping SQL types to XML.

Example 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 different XML elements, depending on the JMS type. IDAP supports queues/topics with the following JMS types: TextMessage, MapMessage, BytesMessage and ObjectMessage. JMS queues with payload type StreamMessage are not supported through IDAP.

The JMS types and XML components are shown in Table 17-2. The distinct XML element for each JMS type is shown in its respective column. Required elements are shown in bold.

Table 17–2 JMS Types and XML Components

Used for queues/topics with payload type:			
`AQ$_JMS_TEXT_MESSAGE`	`AQ$_JMS_MAP_MESSAGE`	`AQ$_JMS_BYTES_MESSAGE`	`AQ$_JMS_OBJECT_MESSAGE`
`jms_text_message`	**`jms_map_message`**	**`jms_bytes_message`**	**`jms_object_message`**

Table 17–2 JMS Types and XML Components

`oracle_jms_properties`			
`user_properties`			
`text_data`—string representing the text payload	**`map_data`**—set of name-value pairs called items, consisting of: ▪ `name` ▪ `int_value` or `string_value` or `long_value` or `double_value` or `boolean_value` or `float_value` or `short_value` or `byte_value`	**`bytes_data`**—hex representation of the payload bytes	**`ser_object_data`**—hex representation of the serialized object

All JMS messages consist of the following common elements:

- `oracle_jms_properties`, which consists of
  - `type`—type of the message
  - `reply_to`—consists of an `agent_name`, `address`, and `protocol`
  - `userid`—supplied by AQ; client cannot specify
  - `appid`—application identifier
  - `groupid`—group identifier
  - `group_sequence`—sequence within the group identified by `group_id`
  - `timestamp`—the time the message was sent, which cannot be specified during enqueue. It is automatically populated in a message that is dequeued.
  - `recv_timestamp`—the time the message was received
- `user_properties`—in addition to the above predefined properties, users can also specify their own message properties as name-value pairs. The `user_properties` consists of a list of property elements. Each property is a name-value pair consisting of the following:

- `name`—property name
- `int_value`—integer property value or

  `string_value`—string property value or

  `long_value`—long property value or

  `double_value`—double property value or

  `boolean_value`—boolean property value or

  `float_value`— float property value or

  `short_value`—short property value or

  `byte_value`—byte property value or

The following examples show enqueue requests using the different message and queue types.

Enqueue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Enqueue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Enqueue Request Example—Sending a Message to a JMS Queue

The 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://ns.oracle.com/AQ/schemas/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>
```

Enqueue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Enqueue Request Example—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://ns.oracle.com/AQ/schemas/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>426C6F622064617461202D20626C6F622064617461202D20626C6F62206461746120426C6F6
```

```
22064617461202D20626C6F622064617461202D20626C6F62206461746120426</RAW>
                  </message_payload>
                </message>
            </message_set>
          </AQXmlSend>
      </Body>
</Envelope>
```

Enqueue Request Example—Sending/Publishing and Committing the Transaction

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/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>
```

Client Requests for Dequeue

Client requests for dequeue use the `AQXmlReceive` method, which takes the arguments and argument attributes shown in Table 17–3. Required arguments are shown in bold.

Table 17–3 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 17–3 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 behavior associated with the dequeue. The dequeue_mode can be specified as one of: ■ `REMOVE` (default): Read the message and update or delete it. This is the default. 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 will be 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 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`.

Dequeue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Dequeue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Dequeue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Dequeue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Dequeue Request Example—Receiving Messages and Committing

In the dequeue request examples, if you include `AQXmlCommit` at the end of the `RECEIVE` request, the transaction is committed upon completion of the operation. In "Dequeue Request Example—Receiving Messages from a Multiconsumer Queue" on page 17-23, the receive request can include the commit flag as follows:

```
 <?xml version="1.0"?>

<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/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>
```

Dequeue Request Example—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://ns.oracle.com/AQ/schemas/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>
```

Client Requests for Registration

Client requests for registration use the AQXmlRegister method, which takes the arguments and argument attributes shown in Table 17–4. Required arguments are shown in bold.

Table 17–4 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>.

Register Request Example—Registering for Notification at an Email Address

To notify an email 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://ns.oracle.com/AQ/schemas/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>
```

Client Requests to Commit a Transaction

A request to commit all actions performed by the user in a session uses the AQXmlCommit method.

Commit Request Example

A commit request has the following format.

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

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.

Rollback Request Example

A rollback request has the following format:

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

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 17–5.

Table 17–5 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

Server Request Example—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://ns.oracle.com/AQ/schemas/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 17–6.

Table 17–6 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

Server Request Example—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://ns.oracle.com/AQ/schemas/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 6</message_id>
            </publish_result>
         </AQXmlPublishResponse>
      </Body>
   </Envelope>
```

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 17–7.

Table 17–7 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

Dequeue Response Example—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://ns.oracle.com/AQ/schemas/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>
```

Dequeue Response Example—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://ns.oracle.com/AQ/schemas/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>
```

Server Response to a Register Request

The response to a register request uses the `AQXmlRegisterResponse` method, which consists of `status_response`. (See Table 17–7 for a description of `status_response`.)

Commit Response

The response to a commit request uses the AQXmlCommitResponse method, which consists of `status_response`. (See Table 17–7 for a description of `status_response`.)

Example

The response to a commit request has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <AQXmlCommitResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
      </AQXmlCommitResponse>
   </Body>
</Envelope>
```

Rollback Response

The response to a rollback request uses the `AQXmlRollbackResponse` method, which consists of `status_response`. (See Table 17–7 for a description of `status_response`.)

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.

Response in Case of Error

In case of an error in any of the above 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 AQ servlet has not been set up correctly
- `detail`, which consists of
  - `status_response`

Example

A FAULT message has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
   <Body>
      <Fault xmlns="http://ns.oracle.com/AQ/schemas/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>
```

IDAP and AQ XML Schemas

IDAP exposes the IDAP schema and AQ XML schema to the client. All documents sent by the parser are validated against these schemas:

- IDAP schema—`http://ns.oracle.com/AQ/schemas/envelope`

- AQ XML schema—`http://ns.oracle.com/AQ/schemas/access`

The IDAP Schema

The IDAP schema describes the structure of a document: envelope, header, and body.

```
<?xml version='1.0'?>
<!-- XML Schema for IDAP v 1.0 Envelope -->

<!-- Copyright (c) Oracle Corporation 2000 All Rights Reserved.  -->

<schema xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:tdsc="http://ns.oracle.com/AQ/schemas/envelope"
        targetNamespace="http://ns.oracle.com/AQ/schemas/envelope">

  <!-- IDAP envelope, header and body -->
  <element name="Envelope" type="tdsc:Envelope"/>
  <complexType name='Envelope'>
    <element ref='tdsc:Header' minOccurs='0'/>
    <element ref='tdsc:Body' minOccurs='1'/>
    <any minOccurs="0" maxOccurs="*" />
    <anyAttribute/>
  </complexType>

  <element name="Header" type="tdsc:Header"/>
  <complexType name='Header'>
    <any minOccurs="0" maxOccurs="*" processContents="skip"/>
    <attribute name="mustUnderstand" type="tdsc:mu_type" use="optional"
value="0"/>
  </complexType>

  <element name="Body" type="tdsc:Body"/>
  <complexType name='Body'>
    <any minOccurs='0' maxOccurs='*'/>
    <anyAttribute/>
  </complexType>

  <simpleType  name="mu_type" base="string">
      <enumeration value="0"/>
      <enumeration value="1"/>
  </simpleType>

  <!-- IDAP fault reporting structure -->
  <complexType name='Fault' final='extension'>
```

```
    <element name='faultcode' type='string'/>
    <element name='faultstring' type='string'/>
    <element name='faultactor' type='uriReference' minOccurs='0'/>
    <element name='detail' type='tdsc:detail' minOccurs='0'/>
  </complexType>

  <complexType name='detail'>
    <any minOccurs='0' maxOccurs='*'/>
    <anyAttribute/>
  </complexType>

</schema>
```

AQ XML Schema

The AQ XML schema describes the contents of the IDAP body for Internet access to AQ features.

```
<?xml version="1.0"?>
<!-- ****************** AQ xml schema  ****************** -->

<!-- Copyright (c) Oracle Corporation 2000 All Rights Reserved.  -->

<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://ns.oracle.com/AQ/schemas/access"
        xmlns:aq = "http://ns.oracle.com/AQ/schemas/access"
        xmlns:xsd = "http://www.w3.org/1999/XMLSchema">

<import namespace = "http://ns.oracle.com/AQ/schemas/envelope"
        schemaLocation = "envelope.xsd" />

<!-- ****************** AQ xml client operations  ****************** -->

   <element name="AQXmlSend">
      <complexType content="mixed">
        <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 content="mixed">
```

```
      <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 content="mixed">
      <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 content="mixed">
      <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 content="empty">
   </complexType>
</element>

<element name="AQXmlRollback">
   <complexType content="empty">
   </complexType>
</element>

<!-- ***************** AQ xml server notifications ***************** -->

<element name="AQXmlNotification">
   <complexType content="mixed">
      <sequence>
         <element ref="aq:notification_options" maxOccurs="1"/>
         <element ref="aq:message_set" minOccurs="0" maxOccurs="1"/>
      </sequence>
   </complexType>
```

```
 </element>

<!-- ***************** AQ xml server responses  ***************** -->

<element name="AQXmlSendResponse">
   <complexType content="mixed">
     <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 content="mixed">
     <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 content="mixed">
     <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 content="mixed">
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
   </complexType>
 </element>

<element name="AQXmlCommitResponse">
   <complexType content="mixed">
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
   </complexType>
 </element>

<element name="AQXmlRollbackResponse">
   <complexType content="mixed">
```

```
        <element ref="aq:status_response" minOccurs="1" maxOccurs="1"/>
    </complexType>
  </element>

<element name="destination">
  <complexType base='string' derivedBy="extension">
    <attribute name="lookup_type" type="aq:dest_lookup_type"
        use="default" value="DATABASE"/>
  </complexType>
</element>

<!-- **** destination lookup type ******* -->
<!-- lookup_type can be specified to either lookup LDAP or use  -->
<simpleType name="dest_lookup_type" base="string">
   <enumeration value="DATABASE"/>
   <enumeration value="LDAP"/>
 </simpleType>

<!-- ****************** Producer Options ****************** -->
<element name="producer_options">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:visibility" minOccurs="0" maxOccurs="1"/>
     <element ref="aq:transformation" minOccurs="0" maxOccurs="1"/>
   </complexType>
 </element>

<!-- ****************** Consumer Options ****************** -->
<element name="consumer_options">
   <complexType content="mixed">
     <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"/>
    </complexType>
 </element>

<!-- ****************** Register Options ****************** -->
<element name="register_options">
   <complexType content="mixed">
```

```
        <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"/>
      </complexType>
    </element>

   <element name="recipient_list">
     <complexType content="mixed">
<element ref="aq:recipient" minOccurs="1" maxOccurs="*"/>
     </complexType>
   </element>

   <!-- ****************** Message Set ************************ -->
   <element name="message_set">
      <complexType content="mixed">
        <element ref="aq:message_count" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:message" minOccurs="0" maxOccurs="*"/>
      </complexType>
   </element>

   <!-- ****************** Message ************************ -->
   <element name="message">
      <complexType content="mixed">
        <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"/>
      </complexType>
   </element>

   <!-- ****************** Message header ****************** -->
   <element name="message_header">
      <complexType content="mixed">
        <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:correlation" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:expiration" minOccurs="0" maxOccurs="1"/>
        <element ref="aq:delay" 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"/>
      </complexType>
    </element>
```

```
<!-- ****************** Oracle JMS properties ****************** -->
<element name="oracle_jms_properties">
   <complexType content="mixed">
     <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"/>
   </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 content="mixed">
     <element ref="aq:property" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ****************** Property  ****************** -->
<element name="property">
   <complexType content="mixed">
     <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 content="mixed">
     <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"/>
   </complexType>
</element>

<!-- ****************** Send result ****************** -->
<element name="send_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ****************** Publish result ****************** -->
<element name="publish_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_id" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ****************** Receive result ****************** -->
<element name="receive_result">
   <complexType content="mixed">
     <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
     <element ref="aq:message_set" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ****************** Notification ************************** -->
<element name="notification_options">
```

```
    <complexType content="mixed">
      <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
      <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
    </complexType>
</element>

<element name="transformation" type="string"/>
<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="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="sender_id">
  <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="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 base="string">
   <enumeration value="ON_COMMIT"/>
   <enumeration value="IMMEDIATE"/>
 </simpleType>
```

```
 </element>

<element name="dequeue_mode">
<simpleType base="string">
   <enumeration value="BROWSE"/>
   <enumeration value="LOCKED"/>
   <enumeration value="REMOVE"/>
   <enumeration value="REMOVE_NODATA"/>
 </simpleType>
</element>

<element name="navigation_mode">
<simpleType base="string">
   <enumeration value="FIRST_MESSAGE"/>
   <enumeration value="NEXT_MESSAGE"/>
   <enumeration value="NEXT_TRANSACTION"/>
 </simpleType>
</element>

<element name="acknowledge">
  <complexType content="empty">
  </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 content="mixed">
     <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"/>
   </complexType>
</element>

<element name="text_data" type="string"/>

<!-- ****************** JMS map message ****************** -->
<element name="jms_map_message">
   <complexType content="mixed">
     <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"/>
   </complexType>
</element>

<!-- ****************** Map data ****************** -->
<element name="map_data">
   <complexType content="mixed">
     <element ref="aq:item" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>

<!-- ****************** Map Item  ****************** -->
<element name="item">
   <complexType content="mixed">
     <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 content="mixed">
        <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"/>
      </complexType>
   </element>

   <element name="bytes_data" type="string"/>

   <!-- ****************** JMS object message ****************** -->
   <element name="jms_object_message">
      <complexType content="mixed">
        <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"/>
      </complexType>
   </element>

   <element name="ser_object_data" type="string"/>

   <!-- ****************** AQ xml Email Authentication  ****************** -->
   <element name="AQXmlEmailAuthentication">
      <complexType content="mixed">
        <sequence>
           <element ref="aq:user_name" minOccurs="1" maxOccurs="1" />
           <element ref="aq:encrypted_key" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:hashed_session_key" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:encrypt_algorithm" minOccurs="1" maxOccurs="1"/>
           <element ref="aq:hashing_algorithm" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>

   <element name="user_name"          type="string"/>
   <element name="encrypted_key"      type="string"/>
   <element name="hashed_session_key" type="string"/>
   <element name="encrypt_algorithm"  type="string"/>
   <element name="hashing_algorithm"  type="string"/>

</schema>
```

Deploying the AQ XML Servlet

The 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.

Creating the AQ XML Servlet Class

The AQ servlet creates a JDBC OCI connection pool to connect to the Oracle9i 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 *Oracle9i Supplied Java Packages Reference* for information on the `AQxmlDataSource` class.

The user specified in the AQxmlDataSource is the AQ servlet `super-user.` This user must have `CREATE SESSION` privilege and `EXECUTE` privilege on the `DBMS_AQIN` package.

Example:

Create a user `AQADM` as the 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_aqin to aqadm;
```

A sample servlet can be created using this superuser as follows:

```
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.AQ.xml.*;

/**
 * This is a sample 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 IDAP requests and performs the requested AQ operations.

> **Note:** The example assumes that the 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 AQ XML Servlet

The 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). Note the following considerations:

- Because the servlet uses JDBC OCI drivers to connect to the Oracle9*i* server, the Oracle9*i* client libraries must be installed on the machine hosting the servlet, as follows:

  The `LD_LIBRARY_PATH` must contain `$ORACLE_HOME/lib`

- The servlet can be compiled using JDK 1.1.x or JDK 1.2.x libraries.
  - For JDK 1.1.x, the `CLASSPATH` must contain:

    ```
    $ORACLE_HOME/jdbc/lib/classes111.zip
    $ORACLE_HOME/jdbc/lib/jta.zip
    $ORACLE_HOME/jdbc/lib/nls_charset11.zip
    ```

```
$ORACLE_HOME/jdbc/lib/jndi.zip
$ORACLE_HOME/lib/lclasses11.zip
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/lib/xschema.jar
$ORACLE HOME/rdbms/jlib/aqapi11.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqxml.jar
$ORACLE_HOME/rdbms/jlib/xsu111.jar
$ORACLE_HOME/jis/lib/servlet.jar
```

- For JDK 1.2.x, the `CLASSPATH` must contain:

```
$ORACLE_HOME/jdbc/lib/classes12.zip
$ORACLE_HOME/jdbc/lib/jta.zip
$ORACLE_HOME/jdbc/lib/nls_charset12.zip
$ORACLE_HOME/jdbc/lib/jndi.zip
$ORACLE_HOME/lib/lclasses12.zip
$ORACLE_HOME/lib/xmlparserv2.jar
$ORACLE_HOME/lib/xschema.jar
$ORACLE_HOME/rdbms/jlib/aqapi.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/aqxml.jar
$ORACLE_HOME/rdbms/jlib/xsu12.jar
$ORACLE_HOME/jis/lib/servlet.jar
```

- After setting the `CLASSPATH`, compile the servlet using `javac` or any other Java compiler.

User Authentication

After the servlet is installed, the Web server must be configured to authenticate all users that send `POST` requests to the AQ servlet. The AQ servlet allows only authenticated users to access the servlet. If the user is not authenticated, 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 AQ servlet, the user name that is used to connect to the Web server is known as the AQ HTTP agent or AQ Internet user.

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 "AQ restricted stuff"
    AuthType Basic
    AuthUserFile /apache/htdocs/userdb/users
    require valid-user
  </Limit>
</Location>
```

User Authorization

After authenticating the users who connect to the AQ servlet, you establish which operations the users are authorized to perform by doing the following:

1. Register the AQ agent for Internet access.
2. Map the AQ agent to one or more database users.

Registering the AQ Agent

To register the AQ agent for Internet access, use `DBMS_AQADM.CREATE_AQ_AGENT`. The `CREATE_AQ_AGENT` procedure takes an `agent_name`. You specify which protocols the user can use to access the servlet—HTTP, SMTP, or both. For agents accessing the AQ servlet using SMTP, an LDAP `certificate_location` must also be specified. See "Setup for Receiving AQ XML Requests Using SMTP (Email)" on page 17-53 for more information.

Example

Create an AQ agent `JOHN` to access the 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 AQ agents function similarly to `CREATE_AQ_AGENT`. These procedures are documented in the *Oracle9i Supplied PL/SQL Packages and Types Reference.*

Mapping the AQ Agent to Database Users

To map an AQ agent to one or more database users, use `DBMS_AQADM.ENABLE_DB_ACCESS`. With the `ENABLE_DB_ACCESS` procedure, you give an 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

Map the 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 AQ servlet creates a database session as one of the database users (db_user) that maps to the AQ agent. The db_user selected has privileges to access the queue specified in the request.

Example

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. Since OE.OE_NEW_ORDERS_QUE is in the OE schema, it does a CREATE SESSION as OE to perform the requested operation.

The AQ servlet creates a connection pool to the Oracle server using the AQ servlet super-user. This super-user creates sessions on behalf of db_users that the 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 "Creating the AQ XML Servlet Class" on page 17-47 for how to create the AQ servlet super-user.

The AQ servlet super-user can be granted CREATE PROXY 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 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 AQ agents, the protocols they are enabled for, and the mapping between AQ agents and database users. Example entries in this view are shown in Table 17–8.

Table 17–8 The SYSTEM.AQ$INTERNET_USERS View

agent_name	db_username	http_enabled	smtp_enabled
scott	cbadmin	YES	NO
scott	buyer	YES	NO
aqadmin	OE	YES	YES
aqadmin	seller	YES	YES
bookstore		NO	YES

Using an LDAP Server with an AQ XML Servlet

An LDAP server is required if:

- The AQ agent is accessing the AQ servlet using SMTP. (See "Setup for Receiving AQ XML Requests Using SMTP (Email)" on page 17-53 for details.)
- 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, it is resolved to `agent_name, address, protocol` using the LDAP server.

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) ;
        }
    }
```

Setup for Receiving AQ XML Requests Using SMTP (Email)

You must set up the database, Web server, LDAP server, and email server to receive AQ XML requests using SMTP.

Database and LDAP Server Setup

To store AQ agents in the LDAP server, the database must be registered to the LDAP server using the Database Configuration Assistant (DBCA), and the value of `GLOBAL_TOPIC_ENABLED` must be set to `TRUE` (default is `FALSE`; reset using `alter system set global_topic_enabled=TRUE`).

To create AQ agents that can access the servlet using SMTP, use the `DBMS_AQADM.CREATE_AQ_AGENT` procedure.

Example

Create an AQ agent for the `appl` application to access the AQ servlet using SMTP and the digital certificate of the application owner, Kurt:

```
DBMS_AQADM.CREATE_AQ_AGENT(
   agent_name => 'appl',
   enable_http => true,
   enable_smtp => true,
   certificate_location => 'cn=kurt,cn=acme,cn=com');
```

The `certificate_location` parameter is required to authenticate the `appl` application when a message is received.

Web Server Setup

1. Establish a user called `ORACLE_SMTP_AGENT` on the Web server that is allowed to access the AQ servlet.

   The Oracle email server will connect to the servlet using user `ORACLE_SMTP_AGENT`.

2. Specify the email server host name or the IP address in the servlet's init( ) method.

   Use `setEmailServerHost(hostname)` or `setEmailServerAddr(ip_address)`in the AQxmlServlet to do this.

   Example: Specify the email server host as follows:

   ```
   setEmailServerAddr("144.25.186.236");  or
   setEmailServerHost("email-srv.us.oracle.com");
   ```

3. Set up an LDAP context in the servlet, as described in "Using an LDAP Server with an AQ XML Servlet" on page 17-52.

   The LDAP server is used to retrieve certificates for the AQ agent and verify the signature in the incoming message.

Email Server Setup

Internet access to AQ using SMTP requires Oracle Email Server 5.5. Do the following:

1. Check that `DBMS_AQST` is installed on the email server.
2. Create an email account for the destination database, that is, the database against which AQ operations are to be performed using the AQ servlet.

See Also: *Oracle eMail Server 5.5 Administration Guide* for how to create an email account on the email server.

3. Set up an email rule for the destination database email account so that it can handle AQ XML client requests by sending them to the AQ servlet.

   The following information is required:

   - The email account of the destination database, for example, `'aqdb1'`;
   - The password of the email account, for example, `'welcome'`
   - The domain in which this email account resides, for example, `'acme.com'`
   - The complete email address of the destination email address, for example, `'aqdb1@acme.com'`
   - The name of the destination database, for example, `'aqdb1'`
   - The URL of the destination database servlet, for example,

     `http://aq-sun.us.oracle.com:8000/aqserv/servlet/AQTestServlet`

   - The user name and password to access the destination database servlet (user name is `ORACLE_SMTP_AGENT`; password is established in "Web Server Setup" on page 17-54).
   - The host and port for LDAP lookup. For example, `host=ldaphost`, `port=389.`
   - The base distinguished name (DN) for LDAP lookup, that is, the DN of the destination database in the LDAP server, for example, `'cn=aqdb1, cn=oraclecontext,cn=acme,cn=com'`.
   - The login DN and password for LDAP lookup, for example NULL for anonymous binds.

4. Register the rule using `dbms_aqst`:

```
declare
   status binary_integer;
begin
   status := dbms_aqst.register_db(
      'aqdb1', -- email user account for aqdb1
      'welcome', -- email user password
      'acme.com', -- email user domain
      'aqdb1@acme.com', -- complete email address
      'aqdb1', -- name of destination database
```

```
      'http://aq-sun:8000/aqserv/servlet/AQTestServlet', -- URL to access
the destination database servlet
      'welcome', -- password of ORACLE_SMTP_AGENT
      'ldaphost', -- LDAP host for lookup client certificates
      '389', -- LDAP port for LDAP lookup
      'cn=aqdb1,cn=oraclecontext,cn=acme,cn=com', -- base DN of LDAP lookup
      NULL, NULL -- anonymous bind
      );
   dbms_output.put_line('register DB status: ' || status);
end;
```

5. Make sure the operation returns status 0.

After the setup is complete, an AQ agent can send email messages to the database email address to perform AQ operations. The AQ operations should be constructed according to IDAP, signed using the Oracle email S/MIME toolkit, and sent as a binary attachment with the name including `IDAP_MESSAGE`.

Using HTTP to Access the AQ XML Servlet

The procedures for an AQ client to make a request to the AQ servlet using HTTP and for the AQ servlet to process the request are as follows:

AQ Client Request to the 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://ns.oracle.com/AQ/schemas/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.

AQ Servlet Processes a Request Using HTTP

1. The server accepts the client HTTP(S) connection.
2. The server authenticates the user (AQ agent) specified by the client.
3. The server receives the `POST` request.
4. The AQ servlet is invoked.

   If this is the first request being serviced by this servlet, the servlet is initialized—its `init() method` is invoked. The `init () method` creates a connection pool to the Oracle server using the AQxmlDataSource parameters (SID, host, port, AQ servlet super-user name, password) provided by the client.
5. The servlet processes the message as follows:
   - If this is the first request from this client, 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 "AQ Client Request to the 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 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, the 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 will create a session for the agent JOHN with the privileges of database user OE. (See "Mapping the AQ Agent to Database Users" for information on ENABLE_DB_ACCESS.)

- A new database transaction is started if no transaction is active in the HTTP session. Subsequent requests in the session will be 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 above request may be as follows:

```
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/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, the transaction will remain 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, it is automatically aborted.

User Sessions and Transactions

After a client is authenticated and connects to the 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 aborted. 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, it must include this HTTP header containing the session ID cookie in subsequent requests. This is automatically done by most Web browsers. However, if you are using a Java or C client to post requests, this has to be done 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 issued to end the transaction. The commit or rollback requests can also be included as part of other 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, the transaction is automatically aborted. This timeout can be modified in the init() method of the servlet by using `setSessionMaxInactiveTime()`. See "Customizing the AQ Servlet" on page 17-63 for more information.

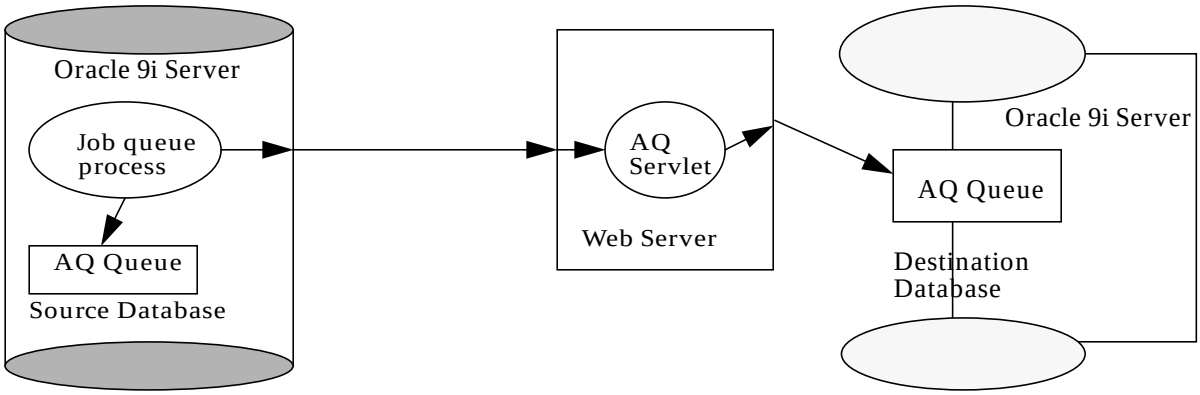
Using HTTP and HTTPS for Advanced Queuing Propagation

Using Advanced Queuing propagation in Oracle9*i*, you can propagate over HTTP and HTTPS (HTTP over SSL) instead of Oracle Net Services (formerly Net8). HTTP, unlike Oracle Net Services, is easy to configure for firewalls.

High-Level Architecture

HTTP AQ propagation uses the infrastructure for Internet access to AQ as its basis. The background process doing propagation pushes messages to an AQ Servlet that enqueues them into the destination database, as shown in Figure 17–3.

Figure 17–3 HTTP Advanced Queuing Propagation



Since 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 below.

Setting Up for HTTP Propagation (and the Differences from Net Services Propagation)

1. The dblink 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 AQ servlet. The username and password of the dblink will be used for authentication with the Web server/servlet runner.
2. An 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 AQ HTTP propagation without any change to the existing code, by following steps 1-3. Similarly an application using AQ http propagation can easily switch back to Net Services propagation just by recreating the dblink with a Net Services connection string, without any other changes.

Setting Up for AQ propagation over HTTP

1. The source database must be created for running Java and XML.
2. Create the dblink with protocol as HTTP and the host and port of the Web server running the AQ servlet, with the username and password for authentication with the webserver/servlet runner.

   For example, if the webserver is running on the machine `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 AQ HTTP agent.
3. 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, communication between the source database and the 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.
4. Deploy the AQ Servlet.

   Create a class AQPropServlet that extends AQxmlServlet as described in [create the 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 Oracle9*i*, the propagation servlet name and deployment path are fixed; that is, they must be `AQPropServlet` and the servlet, respectively.
5. Make sure that the AQ HTTP agent (John) is authorized to perform AQ operations. This is done at the destination database:

   a. Register the AQ agent as follows:

      ```
      dbms_aqadm.create_aq_agent(agent_name => 'John', enable_http => true);
      ```

b. Map the AQ agent to a database user as follows:

```
dbms_aqadm.enable_db_access(agent_name =>'John', db_username =>'CBADM')'
```

6. Start propagation at the source site by calling:

```
dbms_aqdm.schedule_propagation.
dbms_aqadm.schedule_propagation('src_queue', 'dba');
```

Using SMTP to Access the AQ Servlet

The general procedure for an AQ client to make a request to the AQ servlet using SMTP is as follows:

1. The client creates a message with the AQ XML client request. The client signs the message with its private key using the Oracle S/MIME toolkit.
2. The client names the message with a substring, `IDAP_MESSAGE`, and sends it as a binary attachment to the database email address.
3. The email server receives the message.
4. The email server triggers the rule registered for the database email address, which does the following:
   a. Connects to the LDAP server and retrieves the certificate of the sending AQ agent
   b. Verifies the signature of the message
   c. Connects to the Web server as user `ORACLE_SMTP_AGENT` if authentication succeeds, and sends an `HTTP POST` message containing the client request

The procedure for the AQ servlet to process a request is described in "AQ Servlet Processes a Request Using HTTP" on page 17-58. When the servlet sends a response, the email server sends an email message containing the XML response to the address specified in the reply-to field of the original email message.

Customizing the 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 AQ operations.

Setting the Connection Pool Size

The AQ data source is used the specify the backend database to which the servlet connects to perform AQ operations. It contains the database SID, host name, listener port and the username/password of the 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 *Oracle9i Supplied Java Packages Reference* for more information.

The AQ data source creates a pool of 5 connections to the database server by default. If you expect more concurrency, increase the minimum pool size by using the `AQxmlDataSource.setCacheSize(size)` method. The default cache scheme is `OracleConnectionCacheImpl.DYNAMIC_SCHEME`. This implies that the number of connections in the cache grows and shrinks dynamically based on the incoming requests. If you wish to set a maximum limit on the number of connections, you must specify a cache size using the `AQxmlDataSource.setCacheSize(size)` method and set the cache scheme to `OracleConnectionCacheImpl.FIXED_WAIT_SCHEME` using the `AQxmlDataSource.setCacheScheme()` method.

Setting the Session Timeout

After a client is authenticated and connects to the 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 aborted.

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, the transaction is automatically aborted. This timeout can be specified in the init() method of the servlet by using `setSessionMaxInactiveTime()` method.

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);
      }
}
```

Setting the Style Sheet for All Responses from the Servlet

The AQ servlet sends back responses in XML. The servlet administrator can specify a style sheet that is to be set for all responses sent back from this servlet. This can be done by invoking the `setStyleSheet(type,href)` or the `setStyleSheetProcessingInstr(proc_instr) in init()` method of the servlet.

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 AQ Operations

The AQ servlet allows you to register callbacks that will be invoked before and after AQ operations are performed. This allows users to perform AQ and non-AQ operations in the same transaction.

To receive callbacks, users register an object that implements the `oracle.AQ.xml.AQxmlCallback` interface. The `AQxmlCallback` interface has the following methods:

```
public interface AQxmlCallback
{

  /** Callback invoked before any AQ operations are performed by the servlet */
  public void beforeAQOperation(HttpServletRequest request, HttpServletResponse
response,
                                AQxmlCallbackContext ctx);

  /** Callback invoked after any 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 AQ operations. Users can perform other SQL operations in the callback functions using this connection object.

- Note that 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

Before any 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 AQ Servlet callback
 */
public class TestCallback implements oracle.AQ.xml.AQxmlCallback
{

   /** Callback invoked before any 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 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 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);
      }
}
```

A

Oracle Advanced Queuing by Example

In this appendix we provide examples of operations using different programmatic environments:

- Creating Queue Tables and Queues
  - 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 Multiple-Consumer Queue Table and Queue
  - Creating a Queue to Demonstrate Propagation
  - Setting Up Java AQ Examples
  - Creating an Java AQ Session
  - Creating a Queue Table and Queue Using Java
  - Creating a Queue and Start Enqueue/Dequeue Using Java
  - Creating a Multi-Consumer Queue and Add Subscribers Using Java
- Enqueuing and Dequeuing Messages
  - Enqueuing and Dequeuing of Object Type Messages Using PL/SQL
  - Enqueuing and Dequeuing of Object Type Messages Using Pro*C/C++
  - Enqueuing and Dequeuing of Object Type Messages Using OCI
  - Enqueuing and Dequeuing of Object Type Messages (CustomDatum interface) Using Java

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

■ Propagation

- Enqueue of Messages for remote subscribers/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
- Manage Propagation From One Queue To Other Queues In Another Database Using PL/SQL
- Unscheduling Propagation Using PL/SQL

- Dropping AQ Objects
- Revoking Roles and Privileges
- Deploying AQ with XA
- AQ and Memory Usage
  - 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

Creating Queue Tables and Queues

Note: You may need to 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;
```

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');
```

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');
```

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');
```

Creating a Multiple-Consumer 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');
```

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');
```

Setting Up Java 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();
      }
   }
}
```

Creating an Java AQ Session

```
/* Creating an Java AQ Session for the 'aqjava' user as shown in the
   AQDriverManager section above: */
 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;
   }
```

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");
}
```

Creating a Queue and Start Enqueue/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");
}
```

Creating a Multi-Consumer Queue and Add 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 Of Messages

Enqueuing and Dequeuing of 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;

/* 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;
```

Enqueuing and Dequeuing of Object Type Messages Using Pro*C/C++

Note: You may need to 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
```

```
#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 has to 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);
}
```

Enqueuing and Dequeuing of 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);
}
```

Enqueuing and Dequeuing of Object Type Messages (CustomDatum interface) Using Java

To enqueue and dequeue of object type messages follow the lettered steps below.

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();
  }
```

Enqueuing and Dequeuing of Object Type Messages (using SQLData interface) Using Java

To enqueue and dequeue of object type messages follow the lettered steps below.

a. Create the SQL type for the Queue Payload

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

b. Creating 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();
  }
```

Enqueuing and Dequeuing of 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;
```

Enqueuing and Dequeuing of RAW Type Messages Using Pro*C/C++

Note: You may need to 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
```

```
#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 needs to 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;
```

```
}
```

Enqueuing and Dequeuing of 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);
}
```

Enqueue of 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();
    }
```

Dequeue of 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();
    }
```

Dequeue of 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();
}
```

Enqueuing and Dequeuing of Messages by Priority Using PL/SQL

When two messages are enqued with the same priority, the message which was enqued earlier will be dequeued first. However, if two messages are of different priorities, the message with the lower value (higher priority) will be 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 will be retrieved before
the message with priority set to 30 since priority takes precedence over enqueue
time. */
```

Enqueue of 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();
    }
```

Dequeue of 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;
```

Enqueuing and Dequeuing of Messages with Time Delay and Expiration Using PL/SQL

Note: 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, this requires setting the expiration time for 2 weeks. This scenario is described in the following code segment.

```
/* 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;
```

Enqueuing and Dequeuing of Messages by Correlation and Message ID Using Pro*C/C++

Note: You may need to 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
```

```
#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 needs to 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 has to 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;
}
```

Enqueuing and Dequeuing of 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);
}
```

Enqueuing and Dequeuing of 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
   i.e. for RED and GREEN: */
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 i.e. for RED and BLUE.*/
   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;
```

Note that 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;
```

Enqueuing and Dequeuing of Messages to/from a Multiconsumer Queue using OCI

Note: You may need to 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
```

```
#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 i.e. for RED and GREEN */
OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

/* Enqueue MESSAGE 2 for specified recipients i.e. for RED and BLUE */
/* 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);
}
```

Enqueuing and Dequeuing of 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;
```

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 of 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;
```

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. Enqueuing and Dequeuing Messages

```
public static void runTest(AQSession aq_sess)
{
```

```
Connection                db_conn   = null;
AQEnqueueOption           eq_option = null;
AQDequeueOption           dq_option = null;
AQQueue                   queue1    = null;
AQMessage                 adt_msg   = null;
AQMessage                 adt_msg2  = null;
AQObjectPayload           sPayload  = null;
AQObjectPayload           sPayload2 = null;
LobMessage                sPayl     = null;
LobMessage                sPayl2    = null;
AQObjectPayload           rPayload  = null;
LobMessage                rPayl     = null;
byte[]                    smsgid;
AQMessage                 rMessage  = null;
int                       i         = 0;
int                       j         = 0;
int                       id        = 0;
boolean                   more      = false;
byte[]                    b_array;
char[]                    c_array;
String                    mStr      = null;
BLOB                      b1        = null;
CLOB                      c1        = null;
BLOB                      b2        = null;
CLOB                      c2        = null;
BLOB                      b3        = null;
CLOB                      c3        = null;
int                       b_len     = 0;
int                       c_len     = 0;
OracleCallableStatement   blob_stmt0= null;
OracleCallableStatement   clob_stmt0= null;
OracleResultSet           rset0     = null;
OracleResultSet           rset1     = null;
OracleCallableStatement   blob_stmt = null;
OracleResultSet           rset2     = null;
OracleCallableStatement   clob_stmt = null;
OracleResultSet           rset3     = null;

try
{

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

 queue1  = aq_sess.getQueue("aquser", "q1_adt");
```

```
b_array = new byte[5000];
c_array = new char[5000];
for (i = 0; i < 5000; i++)
{
   b_array[i] = 67;
   c_array[i] = 'c';
}
sPayl = new LobMessage();


System.out.println("Enqueue Long messages");

eq_option = new AQEnqueueOption();

/* Enqueue messages with LOB attributes */
for ( i = 0; i < 10; i++)
{
  adt_msg = queue1.createMessage();

  sPayload = adt_msg.getObjectPayload();

  /*  Get Empty BLOB handle */
  blob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
               "select empty_blob() from dual");
  rset0 = (OracleResultSet) blob_stmt0.executeQuery ();
  try
  {
    if (rset0.next())
    {
      b1 = (oracle.sql.BLOB)rset0.getBlob(1);
    }
    if (b1 == null)
    {
      System.out.println("select empty_blob() from dual failed");
    }
  }
  catch (Exception ex)
  {
    System.out.println("Exception during select from dual " + ex);
    ex.printStackTrace();
  }

/*  Get Empty CLOB handle */
  clob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
```

```
              "select empty_clob() from dual");
rset1 = (OracleResultSet) clob_stmt0.executeQuery ();
try
{
  if (rset1.next())
  {
    c1 = (oracle.sql.CLOB)rset1.getClob(1);
  }
  if (c1 == null)
  {
    System.out.println("select empty_clob() from dual failed");
  }
}
catch (Exception ex)
{
  System.out.println("Exception2 during select from dual " + ex);
  ex.printStackTrace();
}
id = i+1;
mStr = "Message #" + id;
sPayl.setId(new BigDecimal(id));
sPayl.setTrailer(new BigDecimal(id));
sPayl.setSubject(mStr);
sPayl.setData(b1);
sPayl.setCdata(c1);

/* Set Object Payload data */
sPayload.setPayloadData(sPayl);

/* Enqueue the message */
queue1.enqueue(eq_option, adt_msg);
System.out.println("Enqueued Message: " + id );
smsgid = adt_msg.getMessageId();

/*
 * Note: The message is initially enqueued with an EMPTY BLOB and CLOB
 * After enqueuing the message, we need to get the lob locators and
 * then populate the LOBs
 */
blob_stmt = (OracleCallableStatement)db_conn.prepareCall(
      "SELECT user_data FROM qt_adt where msgid = ?");
blob_stmt.setBytes(1,smsgid);
rset2 = (OracleResultSet) blob_stmt.executeQuery ();
try
{
```

```
    if (rset2.next())
    {
      /* Get message contents */
      sPayl2 = (LobMessage)rset2.getCustomDatum(1,
                      ((CustomDatumFactory)LobMessage.getFactory()));

      /* Get BLOB locator */
      b2 = sPayl2.getData();

      /* Popuate the BLOB */
      if (b2 == null)
      {
         System.out.println("Blob select null");
      }
      if ((i % 3) == 0)
      {
        b_len = b2.putBytes(1000,b_array);
      }
      else
      {
        b_len = b2.putBytes(1,b_array);
      }

      /* Get CLOB locator */
      c2 = sPayl2.getCdata();

      /* Populate the CLOB */
      if (c2 == null)
      {
         System.out.println("Clob select null");
      }
      if ((i % 4) == 0)
      {
        c_len = c2.putChars(2500,c_array);
      }
      else
      {
        c_len = c2.putChars(1,c_array);
      }
    }
  }
  catch (Exception ex)
  {
    System.out.println("Blob or Clob exception: " + ex);
  }
```

```
  }

  Thread.sleep(30000);


 // dequeue messages
 dq_option = new AQDequeueOption();
 dq_option.setWaitTime(AQDequeueOption.WAIT_NONE);


 for (i = 0 ; i < 10 ; i++)
 {
   /* Dequeue the message */
    adt_msg2 = ((AQOracleQueue)queue1).dequeue(dq_option,
                                                LobMessage.getFactory());

    /* Get payload containing LOB data */
    rPayload = adt_msg2.getObjectPayload();
    rPayl = (LobMessage) rPayload.getPayloadData();

    System.out.println("\n  Message: #" + (i+1));
    System.out.println("    Id: " + rPayl.getId());
    System.out.println("    Subject: " + rPayl.getSubject());

    /* Get BLOB data */
    b3 = rPayl.getData();
    System.out.println("    " + b3.length() + " bytes of data");

    /* Get CLOB data */
    c3 = rPayl.getCdata();
    System.out.println("    " + c3.length() + " chars of data");
    System.out.println("    Trailer: " + rPayl.getTrailer());
    db_conn.commit();
 }


}
catch (java.sql.SQLException sql_ex)
{
  System.out.println("SQL Exception: " + sql_ex);
  sql_ex.printStackTrace();
}
catch (Exception ex)
{
```

```
        System.out.println("Exception-2: " + ex);
        ex.printStackTrace();
      }

    }
```

Propagation

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

Enqueue of Messages for remote subscribers/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
i.e. for RED at address another_msg_queue and GREEN at address msg_
queue@another_db.world: */
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 i.e. for RED at address
   another_msg_queue and BLUE.*/
   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.

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');
```

Manage 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');
```

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');
```

For additional examples of Alter Propagation, Enable Propagation and Disable Propagation, see:

- "Example: Alter a Propagation Schedule Using PL/SQL (DBMS_AQADM)" on page 9-76
- "Example: Enable a Propagation Using PL/SQL (DBMS_AQADM)" on page 9-79
- "Example: Disable a Propagation Using PL/SQL (DBMS_AQADM)" on page 82

Dropping AQ Objects

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

```
/* 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

```
CONNECT sys/change_on_install
DROP USER aq;
```

Deploying AQ with XA

Note: You may need to 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 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
 *        (above 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 per 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");
}
```

AQ and Memory Usage

Create_types.sql: Create Payload Types and Queues in Scott's Schema

Note: You may need to set up data structures for certain examples to work, such as:

```
/* 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');
```

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 scott's schema via *create_types.sql*, above. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c` (see below). If there are no messages, 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 */
```

Enqueuing Messages (Reuse Memory) Using OCI

This program, `enqreuse.c`, enqueues each line of text into a queue 'msgqueue' that has been created in scott's 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 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 */
```

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 scott's schema by executing *c*`reate_types.sql`. Messages are enqueued using `enqnoreuse` or `enqreuse`. If there are no messages, 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 */
```

Dequeuing Messages (Reuse Memory) Using OCI

This program, `deqreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in scott's schema by executing `create_types.sql`. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c`. If there are no messages, 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 will automatically resize 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 will be
   * 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 timesout & exits.
   * If it does reach here, it will be 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 */
```

B

Oracle JMS Interfaces, Classes and Exceptions

Table B–1 Interfaces, Classes, and Exceptions

Interface / Class / Exception
Oracle JMSClasses (part 1) on page B-6
Oracle JMS Classes (part 2) on page B-7
Oracle JMS Classes (part 3) on page B-8
Oracle JMS Classes (part 4) on page B-9
Oracle JMS Classes (part 5) on page B-10
Oracle JMS Classes (part 6) on page B-11
Oracle JMS Classes (part 7) on page B-13
Oracle JMS Classes (part 8) on page B-14
Oracle JMS Classes (part 9) on page B-15
Oracle JMS Classes (part 10) on page B-16
Interface - javax.jms.BytesMessage on page B-18
Interface - javax.jms.Connection on page B-19
Interface - javax.jms.ConnectionFactory on page B-20
Interface - javax.jms.ConnectionMetaData on page B-21
Interface - javax.jms.DeliveryMode on page B-22
Interface - javax.jms.Destination on page B-23
Interface - javax.jms.MapMessage on page B-24
Interface - javax.jms.Message on page B-25
Interface - javax.jms.MessageConsumer on page B-27
Interface - javax.jms.MessageListener on page B-28
Interface - javax.jms.MessageProducer on page B-29
Interface - javax.jms.ObjectMessage on page B-30
Interface - javax.jms.Queue on page B-31
Interface - javax.jms.QueueBrowser on page B-32
Interface - javax.jms.QueueConnection on page B-33
Interface - javax.jms.QueueConnectionFactory on page B-34
Interface - javax.jms.QueueReceiver on page B-35

Table B–1 Interfaces, Classes, and Exceptions

Interface / Class / Exception (Cont.)
Interface - javax.jms.QueueSender on page B-36
Interface - javax.jms.QueueSession on page B-37
Interface - javax.jms.Session on page B-38
Interface - javax.jms.StreamMessage on page B-39
Interface - javax.jms.TextMessage on page B-40
Interface - javax.jms.Topic on page B-41
Interface - javax.jms.TopicConnection on page B-42
Interface - javax.jms.TopicConnectionFactory on page B-43
Interface - javax.jms.TopicPublisher on page B-44
Interface - javax.jms.TopicSession on page B-45
Interface - javax.jms.TopicSubscriber on page B-46
Exception javax.jms.InvalidDestinationException on page B-47
Exception javax.jms.InvalidSelectorException on page B-48
Exception javax.jms.JMSException on page B-49
Exception javax.jms.MessageEOFException on page B-50
Exception javax.jms.MessageFormatException on page B-51
Exception javax.jms.MessageNotReadableException on page B-52
Exception javax.jms.MesageNotWriteableException on page B-53
Interface - oracle.jms.AdtMessage on page B-54
Interface - oracle.jms.AQjmsQueueReceiver on page B-55
Interface - oracle.jms.AQjmsQueueSender on page B-56
Interface - oracle.jms.AQjmsTopicPublisher on page B-57
Interface - oracle.jms.TopicReceiver on page B-58
Interface - oracle.jms.AQjmsTopicSubscriber on page B-59
Interface - oracle.jms.AQjmsTopicReceiver on page B-60
Class - oracle.jms.AQjmsAdtMessage on page B-61
Class - oracle.jms.AQjmsAgent on page B-62

Table B–1 Interfaces, Classes, and Exceptions

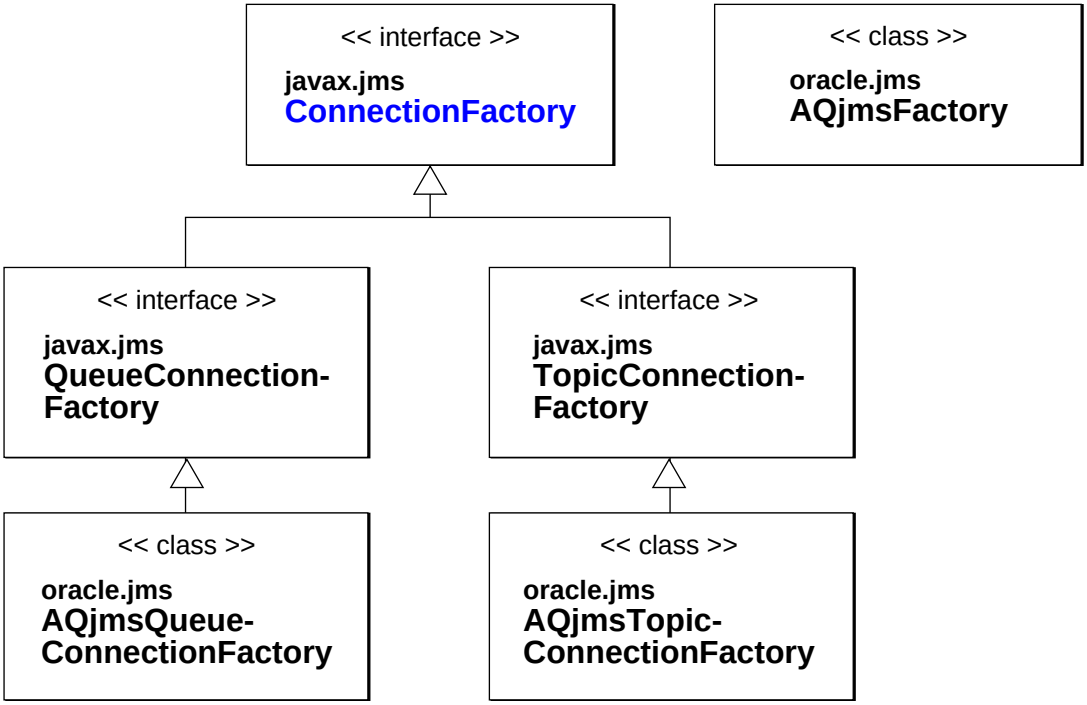
Interface / Class / Exception (Cont.)
Class - oracle.jms.AQjmsBytesMessage on page B-63
Class - oracle.jms.AQjmsConnection on page B-64
Interface - oracle.jms.AQjmsConnectionMetadata on page B-65
Class - oracle.jms.AQjmsConstants on page B-66
Interface - oracle.jms.AQjmsConsumer on page B-67
Class - oracle.jms.AQjmsDestination on page B-68
Class - oracle.jms.AQjmsDestinationProperty on page B-70
Class - oracle.jms.AQjmsFactory on page B-71
Class - oracle.jms.AQjmsMapMessage on page B-72
Class - oracle.jms.AQjmsMessage on page B-73
Class - oracle.jms.AQjmsObjectMessage on page B-74
Class - oracle.jms.AQjmsOracleDebug on page B-75
Class - oracle.jms.AQjmsProducer on page B-76
Class - oracle.jms.AQjmsQueueBrowser on page B-77
Class - AQjmsQueueConnectionFactory on page B-78
Class - oracle.jms.AQjmsSession on page B-79
Class - oracle.jms.AQjmsStreamMessage on page B-82
Class - oracle.jms.AQjmsTextMessage on page B-83
Class - oracle.jms.AQjmsTopicConnectionFactory on page B-84
Exception oracle.jms.AQjmsInvalidDestinationException on page B-86
Exception oracle.jms.AQjmsInvalidSelectorException on page B-87
Exception oracle.jms.AQjmsMessageEOFException on page B-88
Exception oracle.jms.AQjmsMessageFormatException on page B-89
Exception oracle.jms.AQjmsMessageNotReadableException on page B-90
Exception oracle.jms.AQjmsMesssageNotWriteableException on page B-91
Interface - oracle.AQ.AQQueueTable on page B-92
Class - oracle.AQ.AQQueueTableProperty on page B-93

Table B–1 Interfaces, Classes, and Exceptions

Interface / Class / Exception (Cont.)
Interface - oracle.jms.TopicBrowser on page B-95
Class - oracle.jms.AQjmsTopicBrowser on page B-96

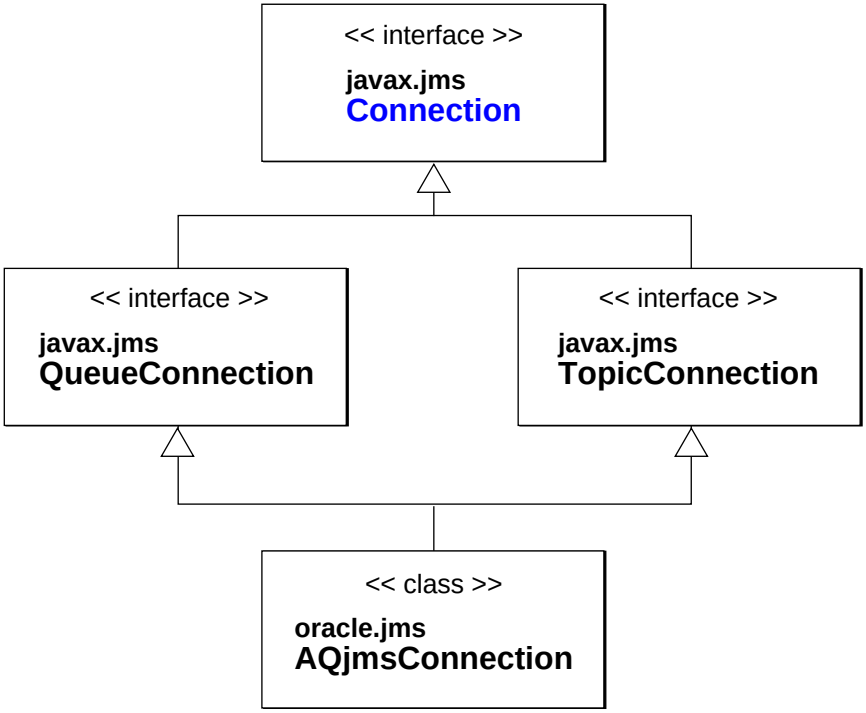
Oracle JMSClasses (part 1)

Figure B–1 Class Diagram: Oracle Class Classes (part 1)



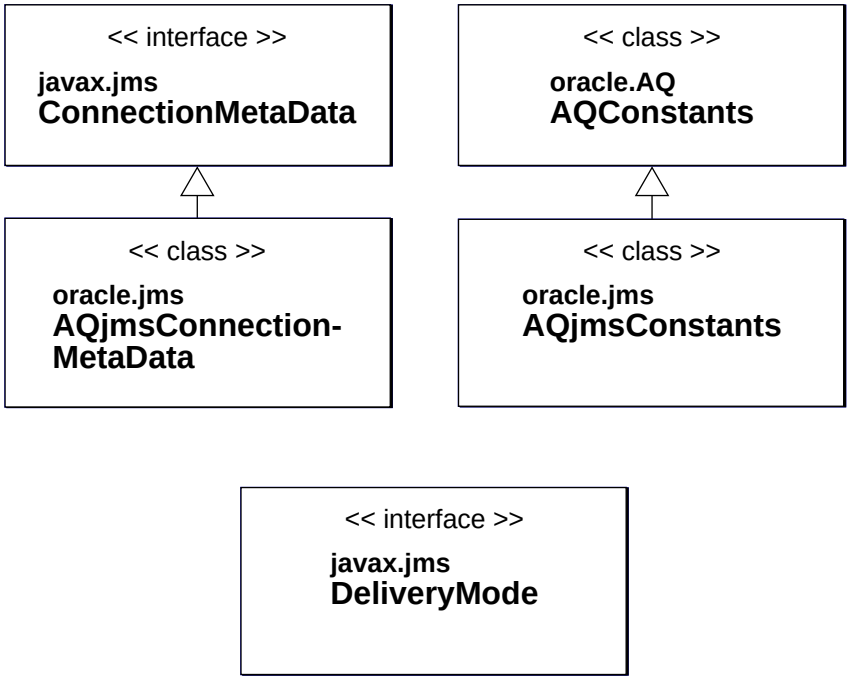
Oracle JMS Classes (part 2)

Figure B–2 Class Diagram: Oracle Class Classes (part 2)



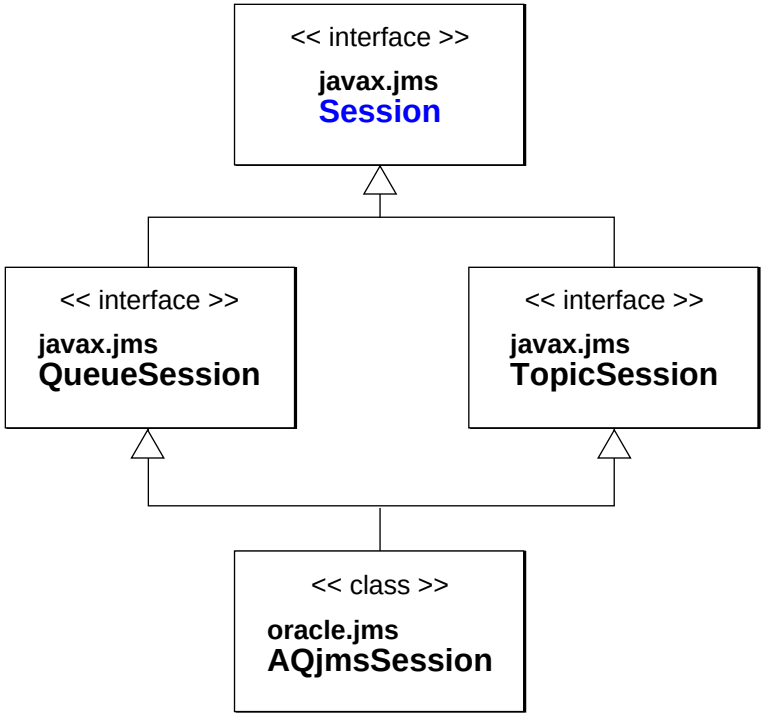
Oracle JMS Classes (part 3)

Figure B–3 Class Diagram: Oracle Class Classes (part 3)



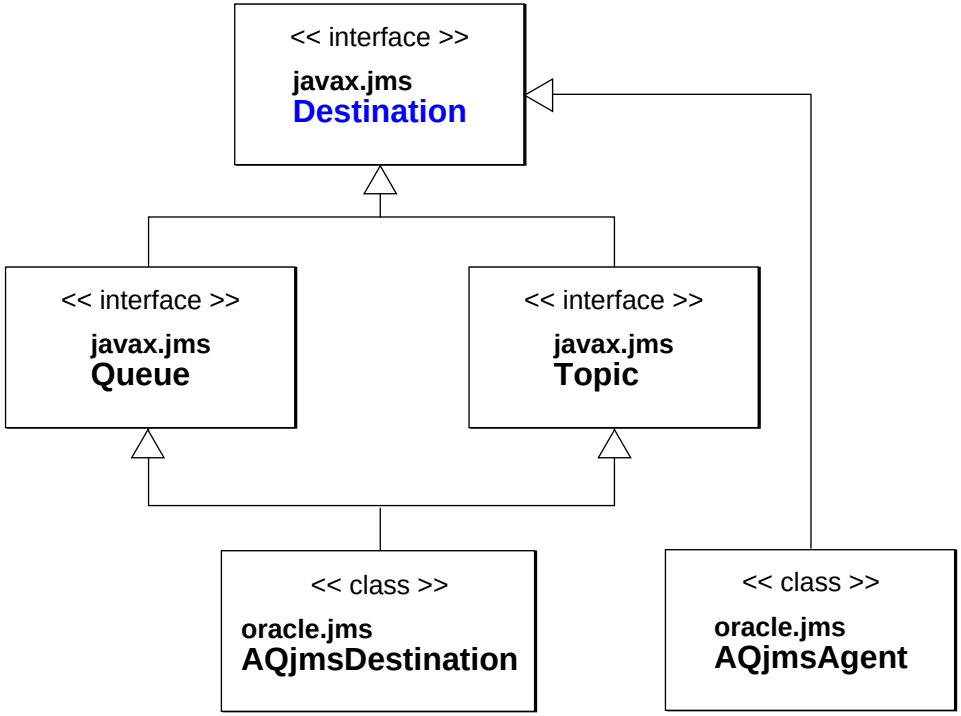
Oracle JMS Classes (part 4)

Figure B–4 Class Diagram: Oracle Class Classes (part 4)



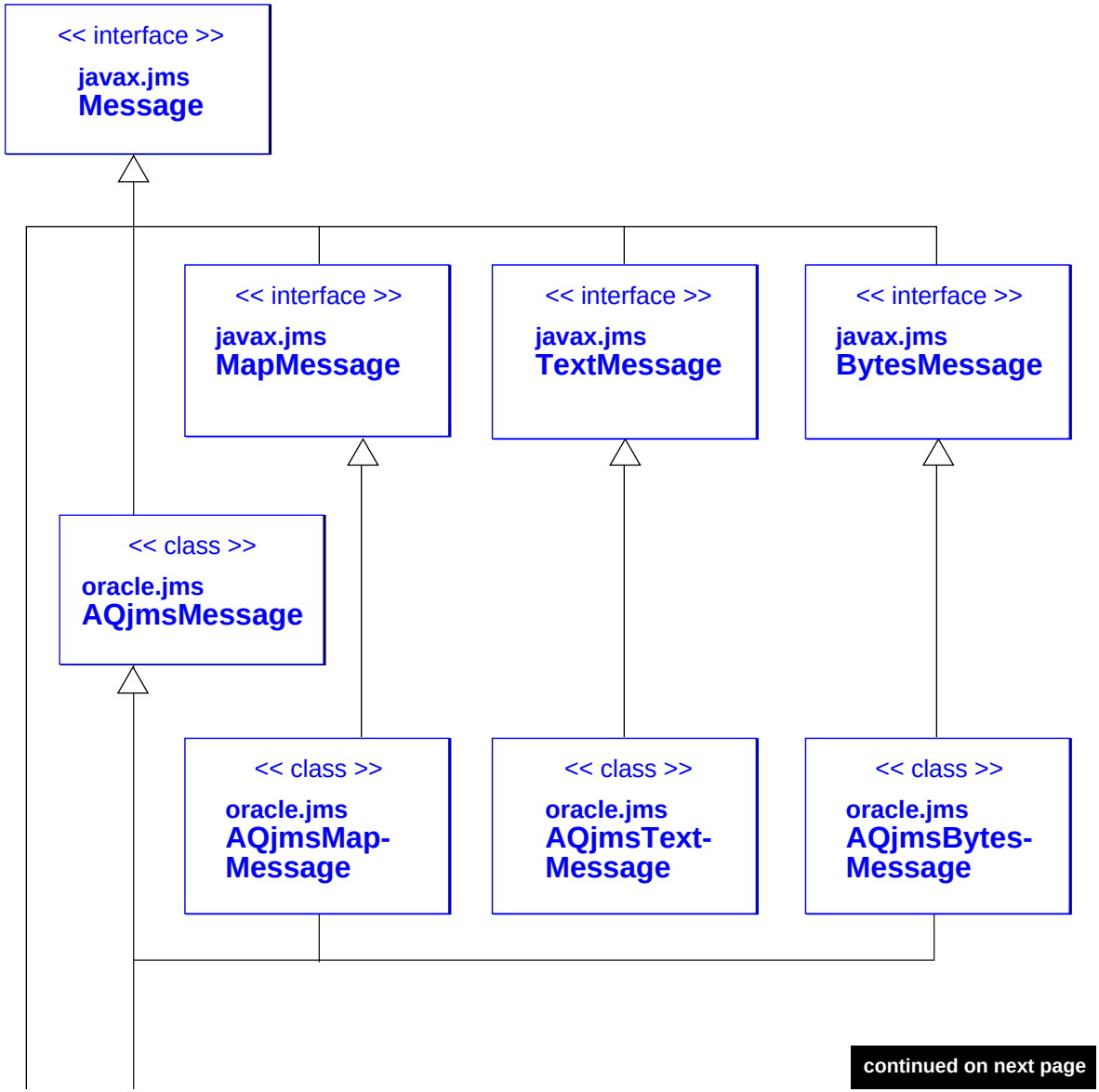
Oracle JMS Classes (part 5)

Figure B–5 Class Diagram: Oracle Class Classes (part 5)



Oracle JMS Classes (part 6)

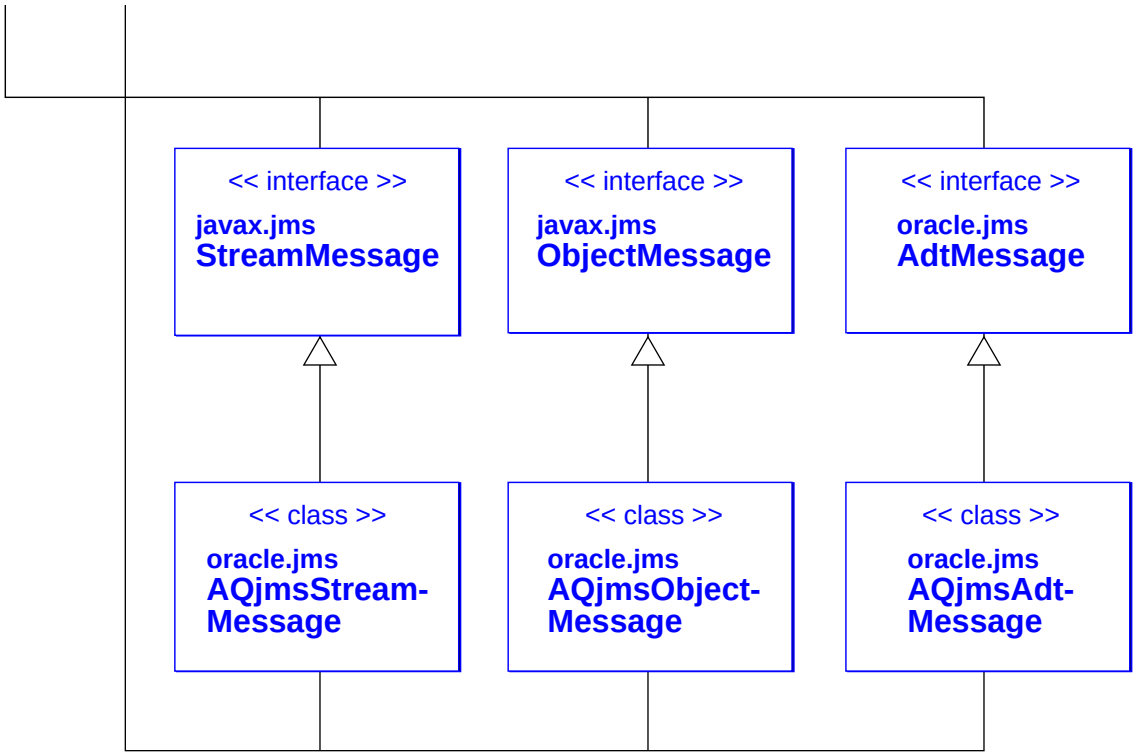
Figure B–6 Class Diagram: Oracle Class Classes (part 6)



continued on next page

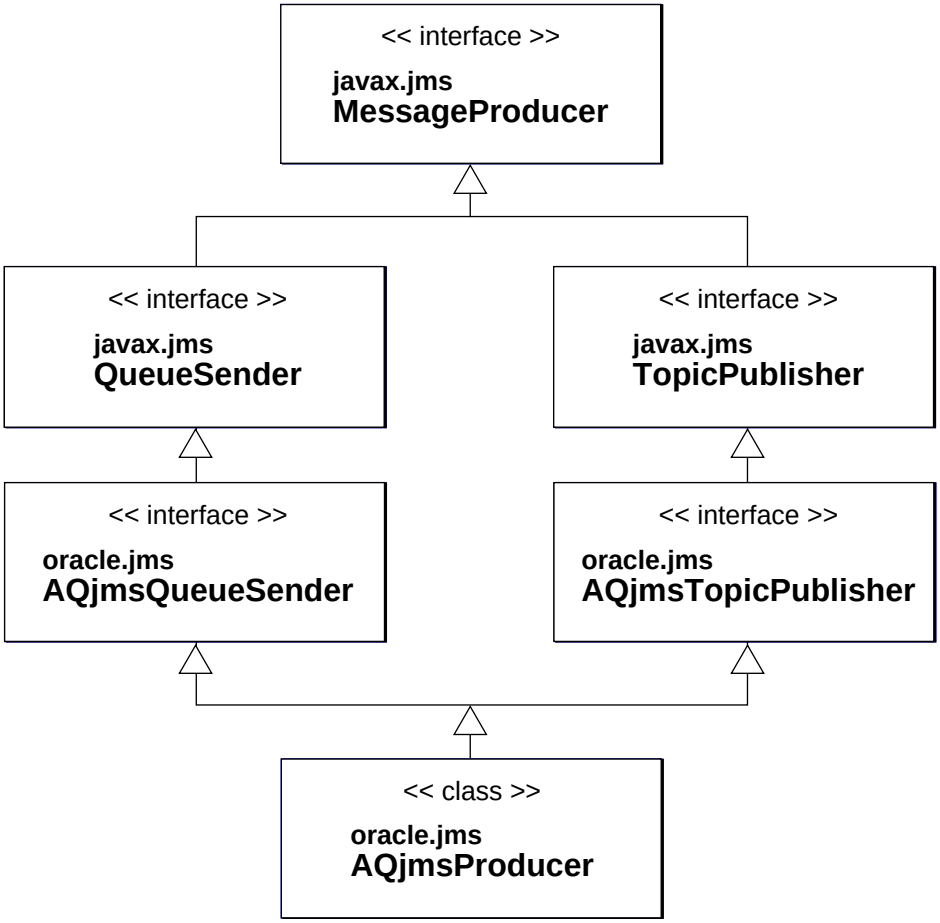
Oracle JMS Classes (part 6 continued)

Figure B–7 Class Diagram: Oracle Class Classes (part 8)



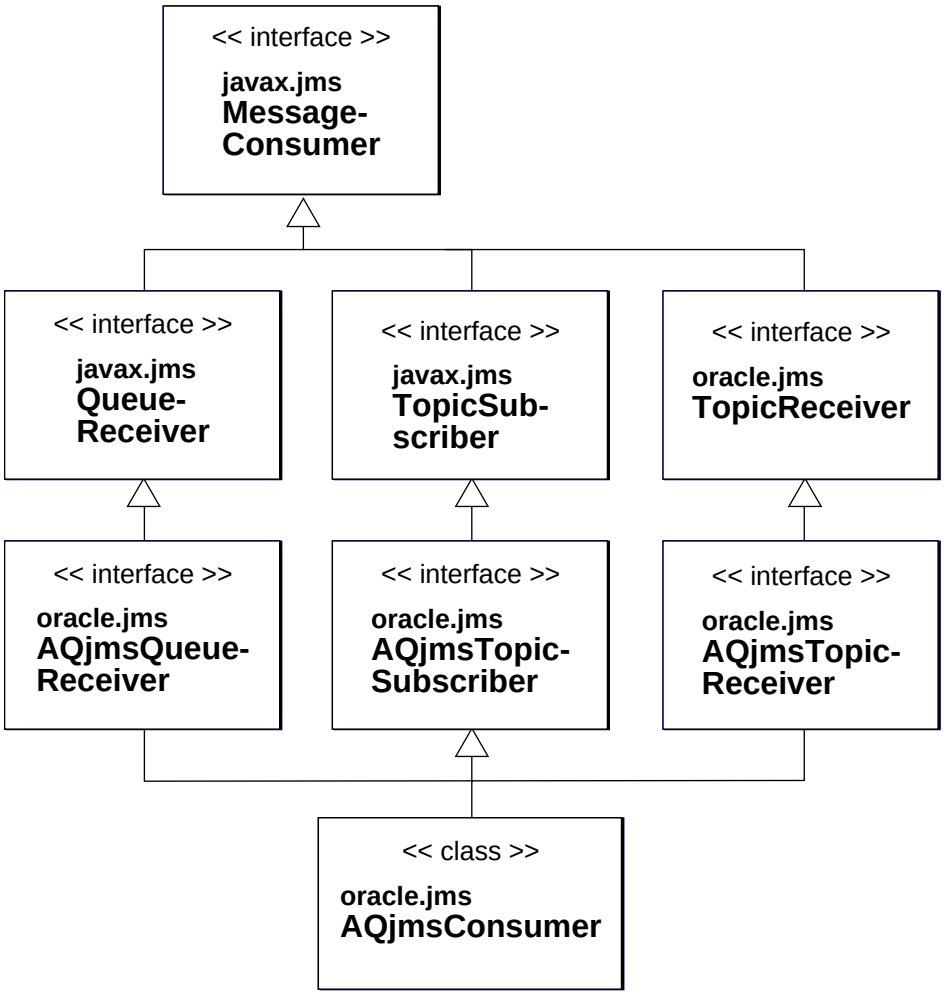
Oracle JMS Classes (part 7)

Figure B–8 Class Diagram: Oracle Class Classes (part 7)



Oracle JMS Classes (part 8)

Figure B–9 Class Diagram: Oracle Class Classes (part 8)

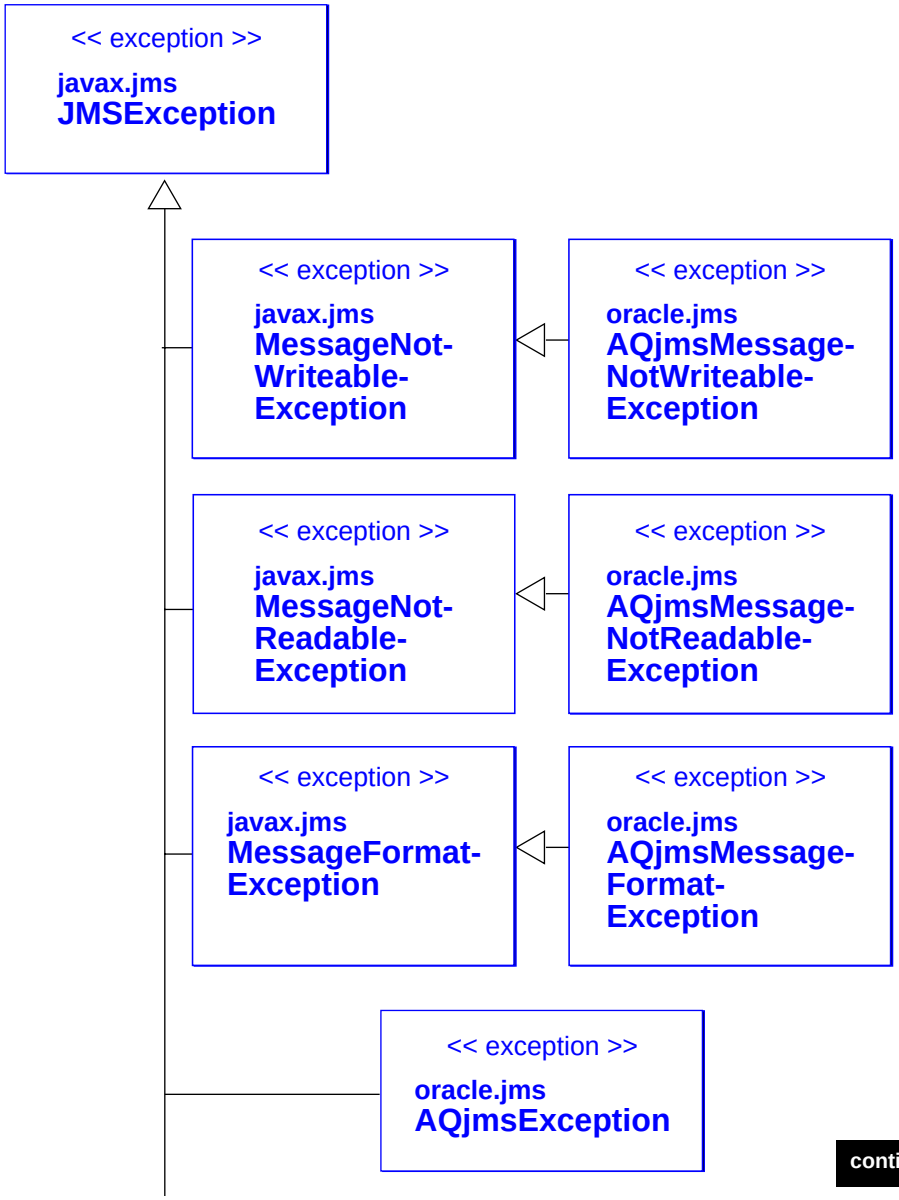


Oracle JMS Classes (part 9)

Figure B–10 Class Diagram: Oracle Class Classes (part 9)

Oracle JMS Classes (part 10)

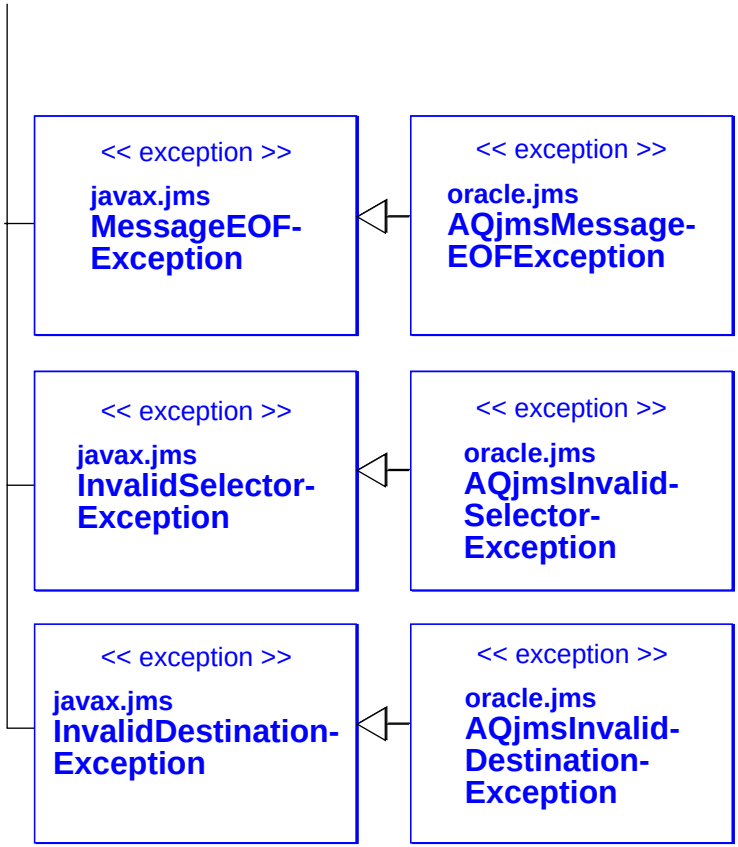
Figure B–11 Class Diagram: Oracle Class Classes (part 10)



continued on next page

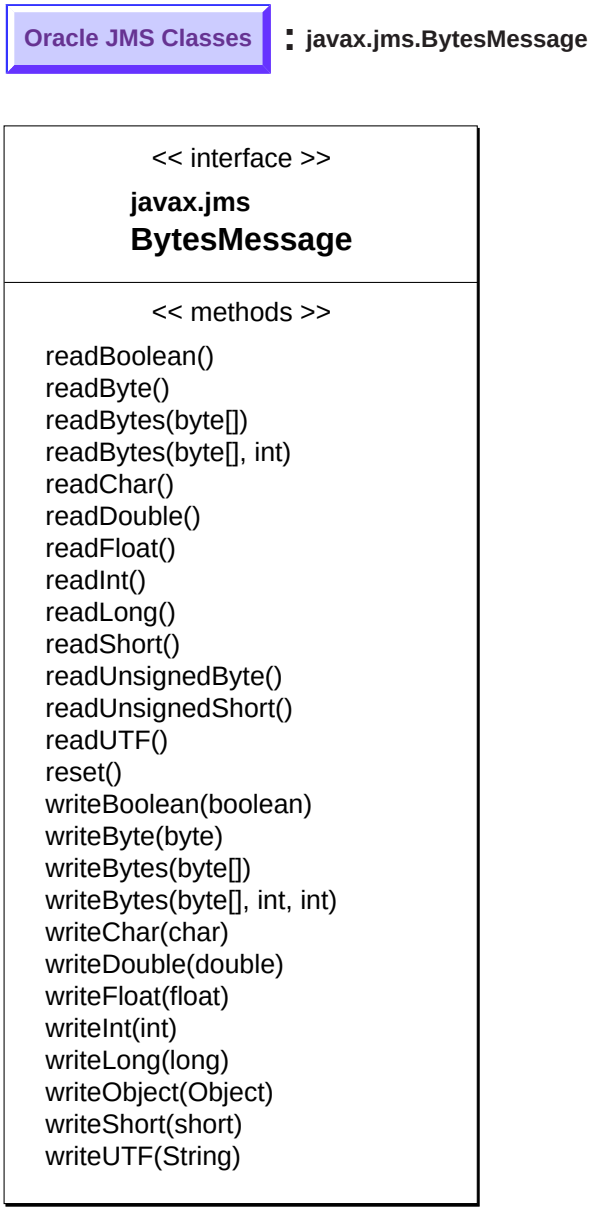
Oracle JMS Classes (part 10 continued)

Figure B–12 Class Diagram: Oracle Class Classes (part 8)



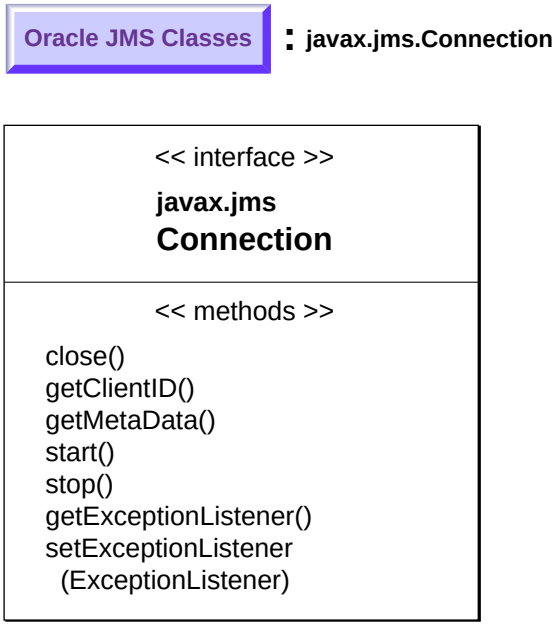
Interface - javax.jms.BytesMessage

Figure B–13 Class Diagram: Interface - javax.jms.BytesMessage



Interface - javax.jms.Connection

Figure B–14 Class Diagram: Interface - javax.jms.Connection



Use Cases

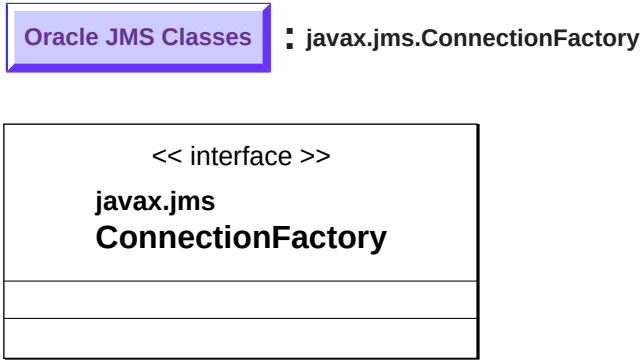
Starting a JMS Connection

Stopping a JMS Connection

Closing a JMS Connection

Interface - javax.jms.ConnectionFactory

Figure B–15 Class Diagram: Interface - javax.jms.ConnectionFactory



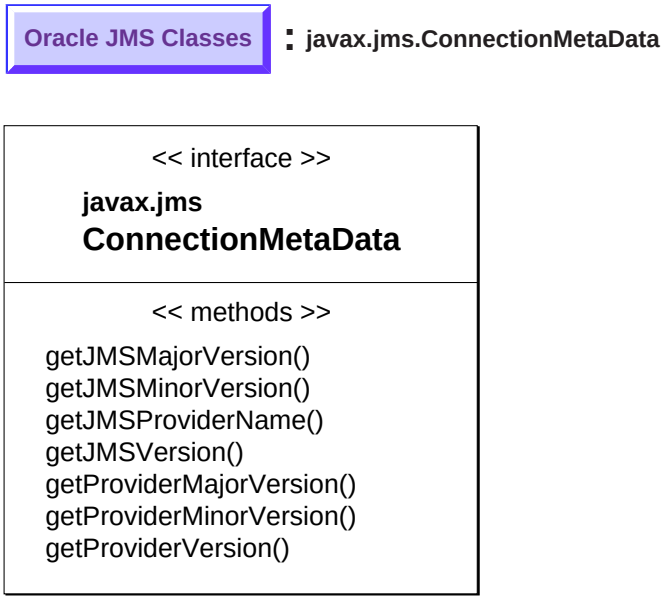
Use Cases

Creating a Queue Connection with Username/Password

Creating a Queue Connection with Open JDBC Connection

Interface - javax.jms.ConnectionMetaData

Figure B–16 Class Diagram: Interface - javax.jms.ConnectionMetaData

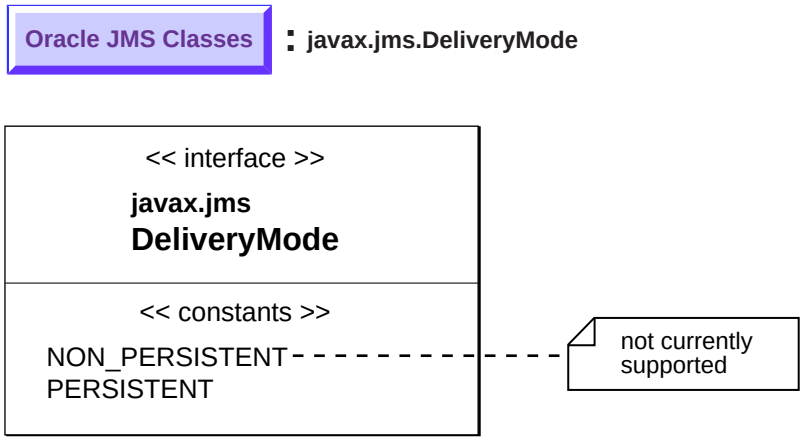


Use Cases

No use case available.

Interface - javax.jms.DeliveryMode

Figure B–17 Class Diagram: Interface - javax.jms.DeliveryMode

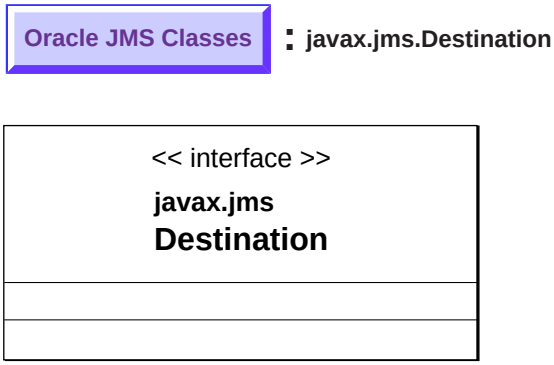


Use Cases

No use case available.

Interface - javax.jms.Destination

Figure B–18 Class Diagram: Interface - javax.jms.Destination

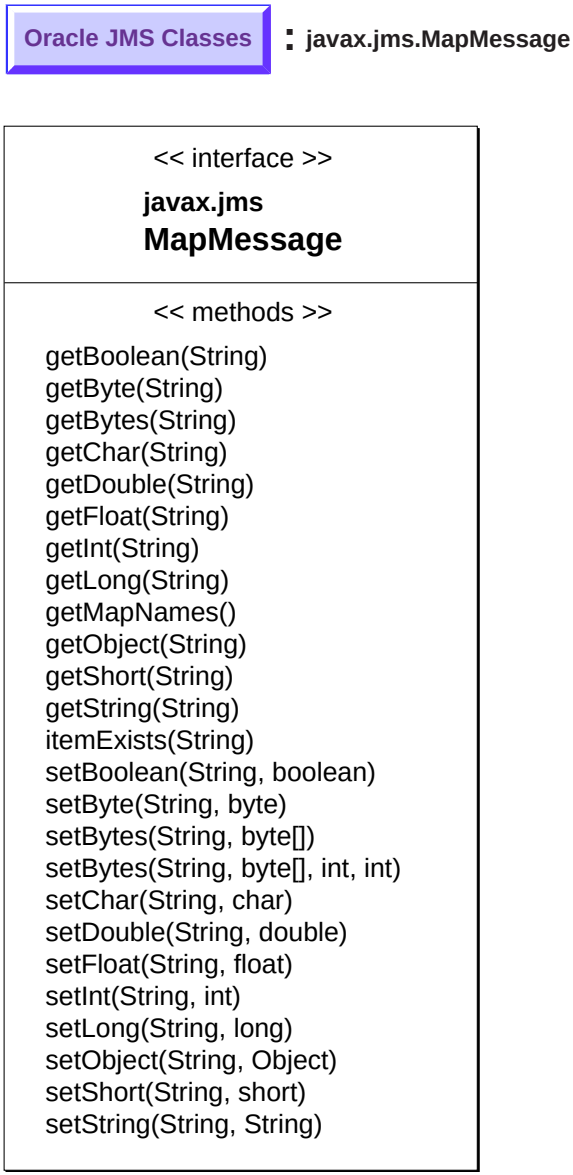


Use Cases

No use case available.

Interface - javax.jms.MapMessage

Figure B–19 Class Diagram: Interface - javax.jms.MapMessage



Interface - javax.jms.Message

Figure B–20 Class Diagram: Interface - javax.jms.Message

Use Cases

Specifying Message Correlation ID

Specifying JMS Message Property

Specifying Message Property as Boolean

Specifying Message Property as String

Specifying Message Property as Int

Specifying Message Property as Double

Specifying Message Property as Float

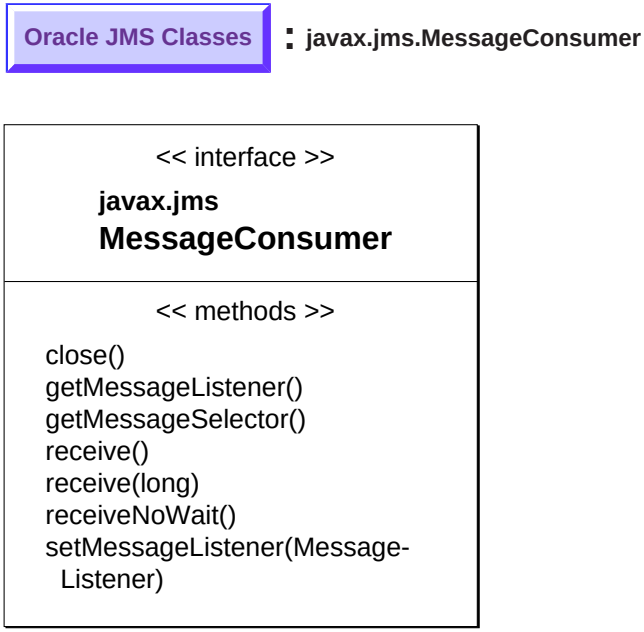
Specifying Message Property as Byte

Specifying Message Property as Long

Specifying Message Property as Object

Interface - javax.jms.MessageConsumer

Figure B–21 Class Diagram: Interface - javax.jms.MessageConsumer



Use Cases

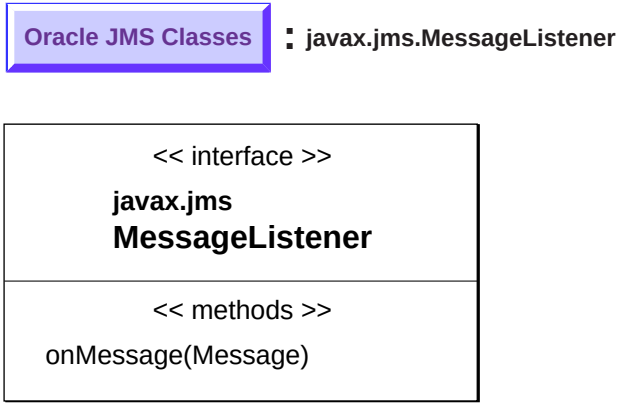
Two Ways of Receiving a Message Synchronously Using a Message Consumer

Receiving a Message Using a Message Consumer by Specifying Timeout

Receiving a Message Using a Message Consumer Without Waiting

Interface - javax.jms.MessageListener

Figure B–22 Class Diagram: Interface - javax.jms.MessageListener

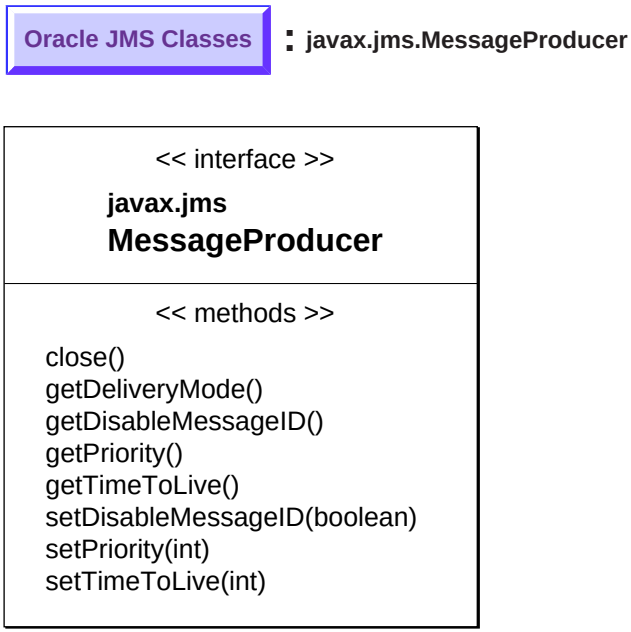


Use Cases

No use case available.

Interface - javax.jms.MessageProducer

Figure B–23 Class Diagram: Interface - javax.jms.MessageProducer



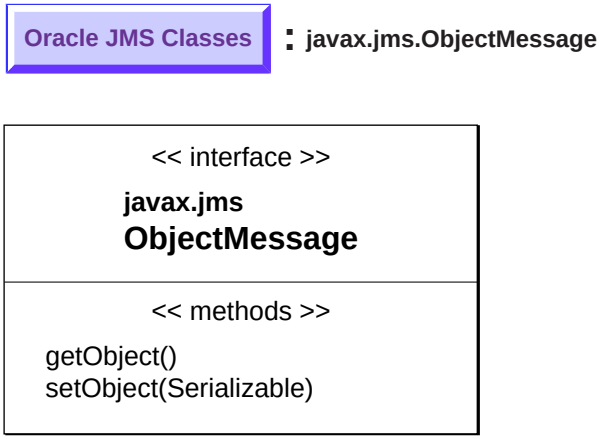
Use Cases

Setting Default TimeToLive for All Messages Sent by a Message Producer

Setting Default Priority for All Messages Sent by a Message Producer

Interface - javax.jms.ObjectMessage

Figure B–24 Class Diagram: Interface - javax.jms.ObjectMessage

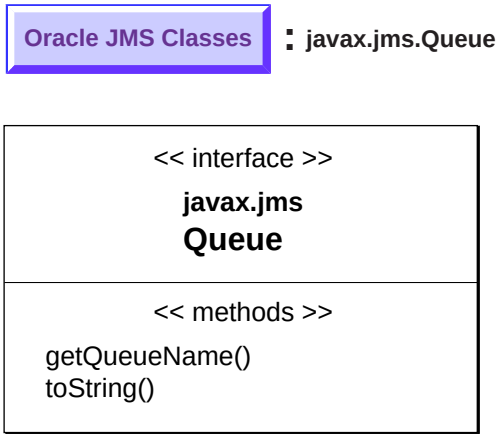


Use Cases

No use case available.

Interface - javax.jms.Queue

Figure B–25 Class Diagram: Interface - javax.jms.Queue

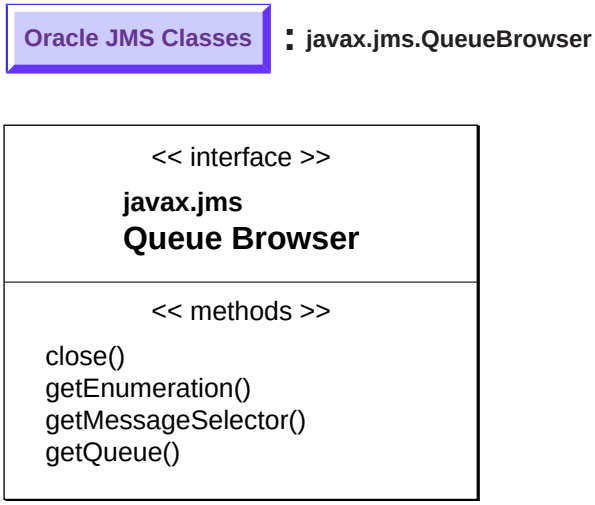


Use Cases

No use case available.

Interface - javax.jms.QueueBrowser

Figure B–26 Class Diagram: Interface - javax.jms.QueueBrowser

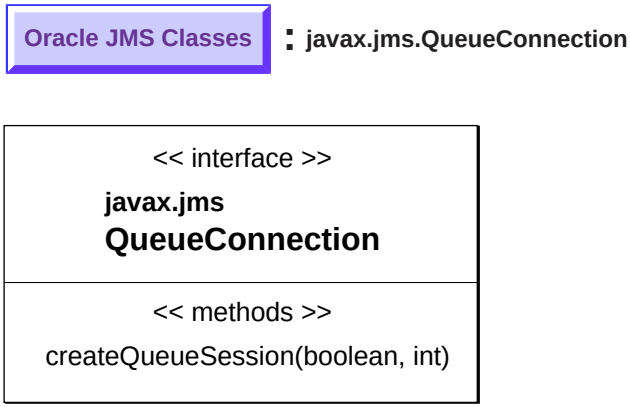


Use Cases

No use case available.

Interface - javax.jms.QueueConnection

Figure B–27 Class Diagram: Interface - javax.jms.QueueConnection

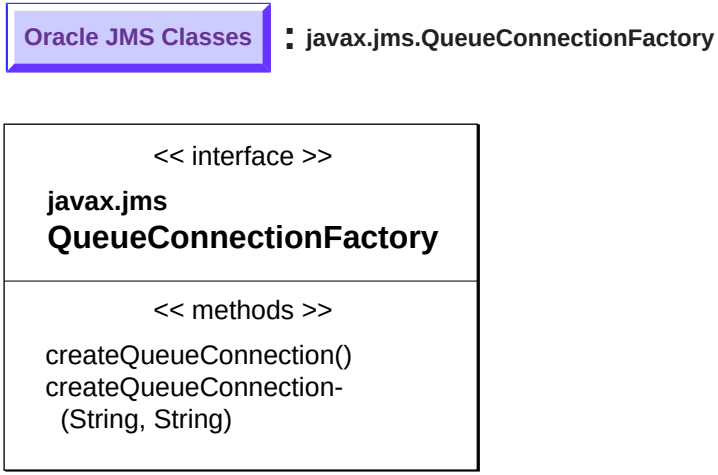


Use Cases

Creating a Queue Session

Interface - javax.jms.QueueConnectionFactory

Figure B–28 Class Diagram: Interface - javax.jms.QueueConnectionFactory

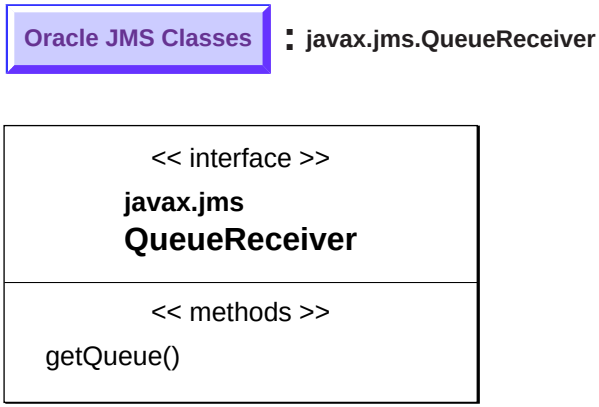


Use Cases

Four Ways of Creating a Queue Connection

Interface - javax.jms.QueueReceiver

Figure B–29 Class Diagram: Interface - QueueReceiver

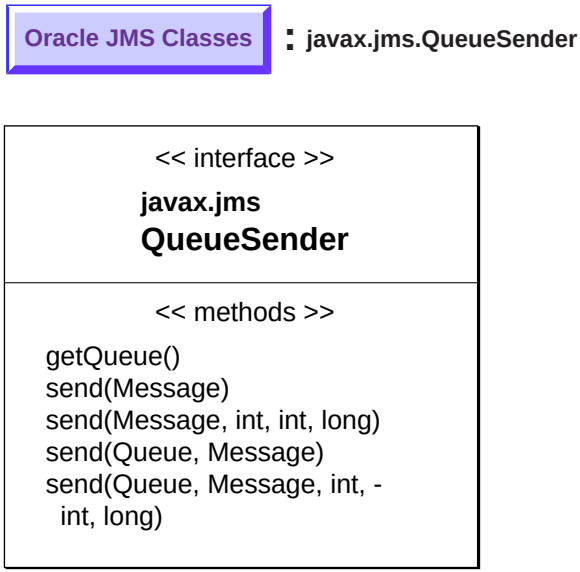


Use Cases

No use case available.

Interface - javax.jms.QueueSender

Figure B–30 Class Diagram: Interface - javax.jms.QueueSender



Use Cases

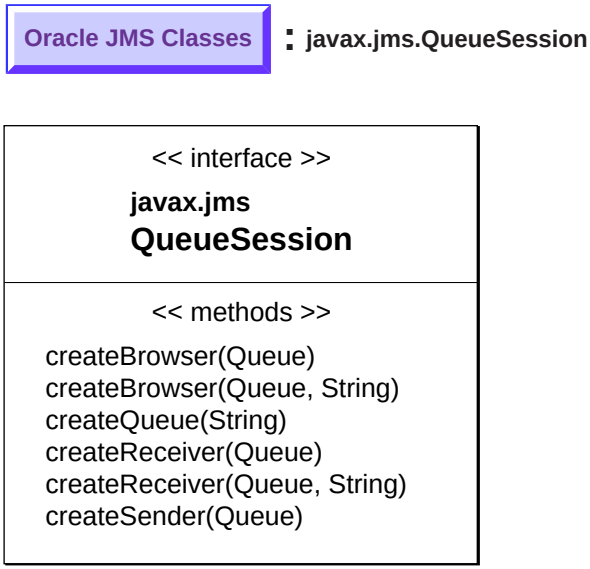
Creating a Queue Sender

Two Ways of Sending Messages Using a Queue Sender

Sending a Message Using a Queue Sender with Default Send Options

Interface - javax.jms.QueueSession

Figure B–31 Class Diagram: Interface - javax.jms.QueueSession



Use Cases

No use case available.

Interface - javax.jms.Session

Figure B–32 Class Diagram: Interface - javax.jms.Session (not available)

Use Cases

Creating a Queue Sender

Two Ways of Creating a Queue Browser for JMS Message Queues

Creating a Queue Browser for Queues with Text, Stream, Objects, Bytes or Map Messages

Creating a Queue Receiver for Queues of Standard JMS Type Messages

Creating a Queue Connection with Open JDBC Connection

Creating a Map Message

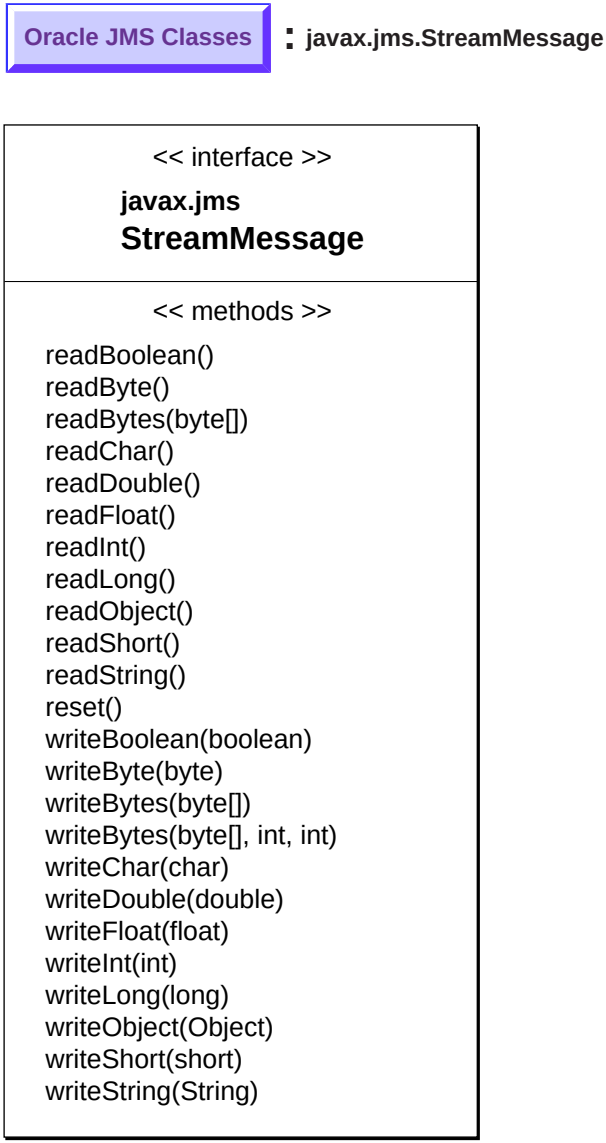
Creating a Stream Message

Creating an Object Message

Creating a Text Message

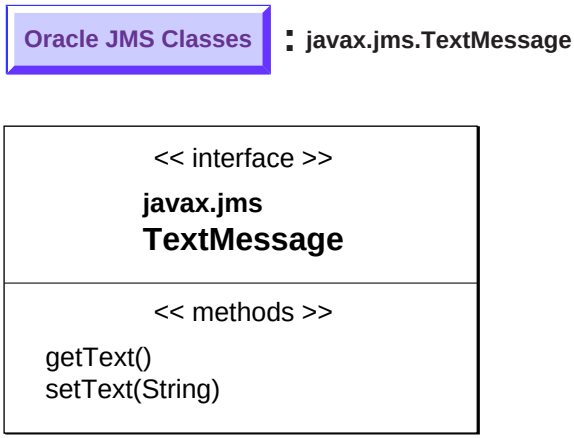
Interface - javax.jms.StreamMessage

Figure B–33 Class Diagram: Interface - javax.jms.StreamMessage



Interface - javax.jms.TextMessage

Figure B–34 Class Diagram: Interface - javax.jms.TextMessage

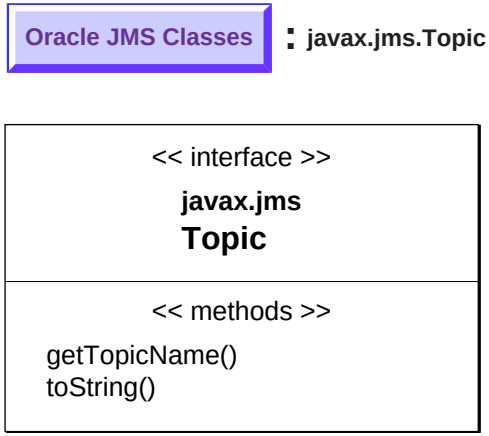


Use Cases

No use case available.

Interface - javax.jms.Topic

Figure B–35 Class Diagram: Interface - javax.jms.Topic

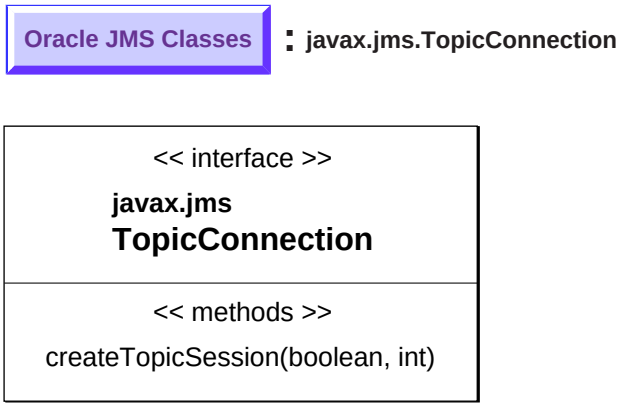


Use Cases

No use case available.

Interface - javax.jms.TopicConnection

Figure B–36 Class Diagram: Interface - javax.jms.TopicConnection

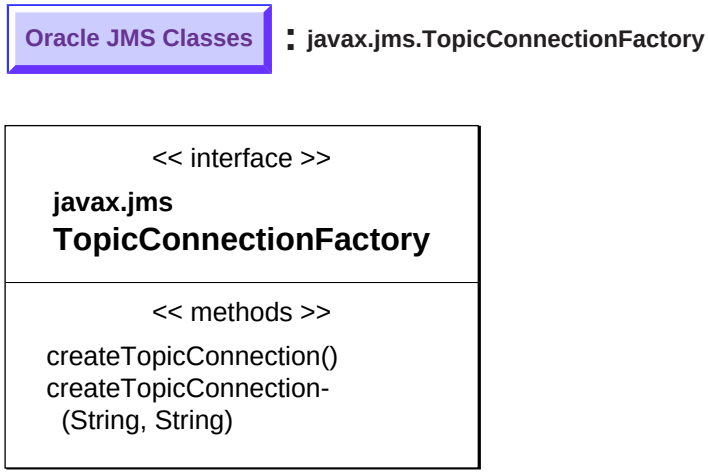


Use Cases

Creating a Topic Connection with Username / Password

Interface - javax.jms.TopicConnectionFactory

Figure B–37 Class Diagram: Interface - javax.jms.TopicConnectionFactory

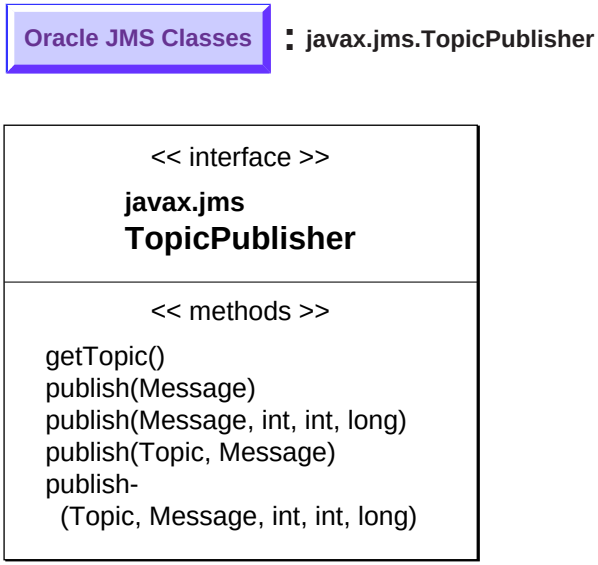


Use Cases

No use case available.

Interface - javax.jms.TopicPublisher

Figure B–38 Class Diagram: Interface - javax.jms.TopicPublisher



Use Cases

Four Ways of Publishing Messages Using a Topic Publisher

Publishing a Message with Minimal Specification

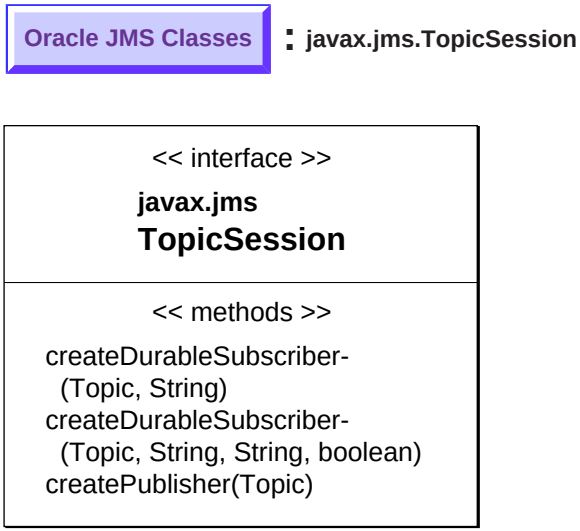
Publishing a Message Specifying Correlation and Delay

Publishing a Message Specifying Priority and Time-To-Live

Publishing Messages Specifying a Recipient List Overriding Topic Subscribers

Interface - javax.jms.TopicSession

Figure B–39 Class Diagram: Interface - javax.jms.TopicSession



Use Cases

Four Ways of Publishing Messages Using a Topic Publisher

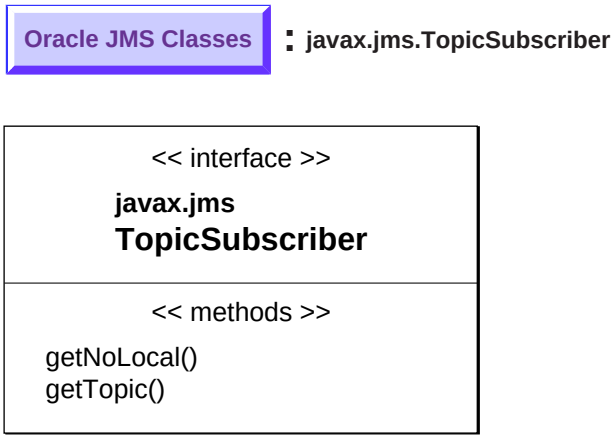
Two Ways of Creating a Durable Subscriber for a Topic of Standard JMS Type Messages

Creating a Durable Subscriber for a JMS Topic without Selector

Creating a Durable Subscriber for a JMS Topic with Selector

Interface - javax.jms.TopicSubscriber

Figure B–40 Class Diagram: Interface - javax.jms.TopicSubscriber

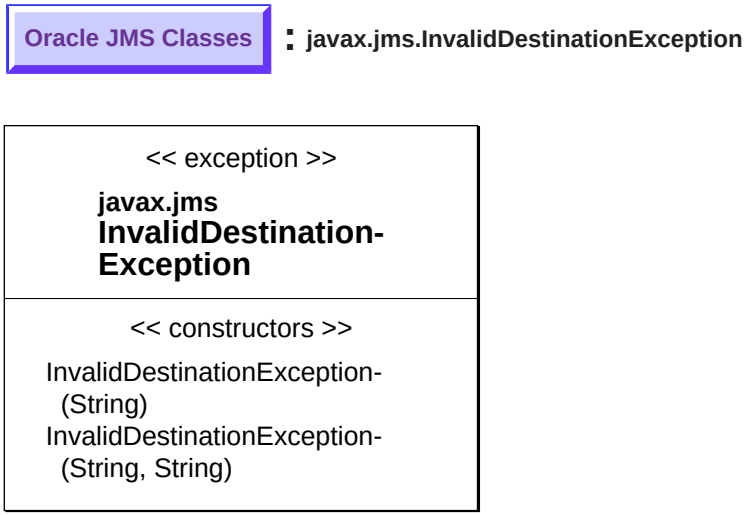


Use Cases

No use case available.

Exception javax.jms.InvalidDestinationException

Figure B–41 Class Diagram: Exception javax.jms.InvalidDestinationException

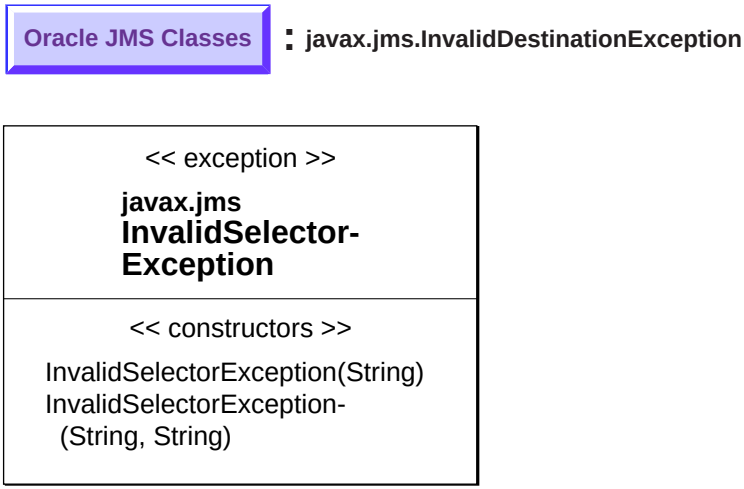


Use Cases

No use case available.

Exception javax.jms.InvalidSelectorException

Figure B–42 Class Diagram: Exception javax.jms.InvalidSelectorException

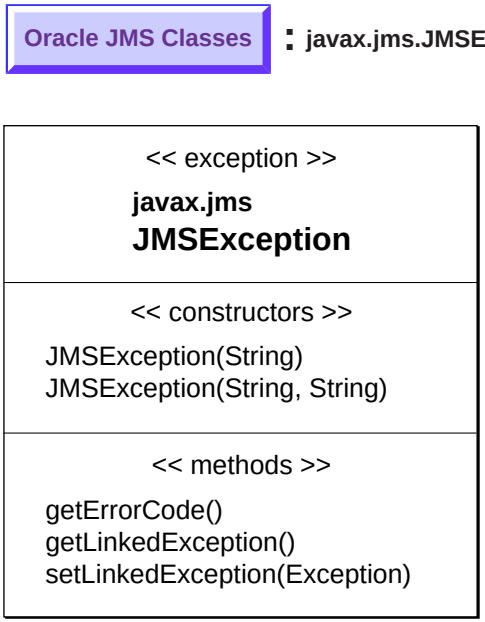


Use Cases

No use case available.

Exception javax.jms.JMSException

Figure B–43 Class Diagram: Exception javax.jms.JMSException



Use Cases

Getting the Error Code for the JMS Exception

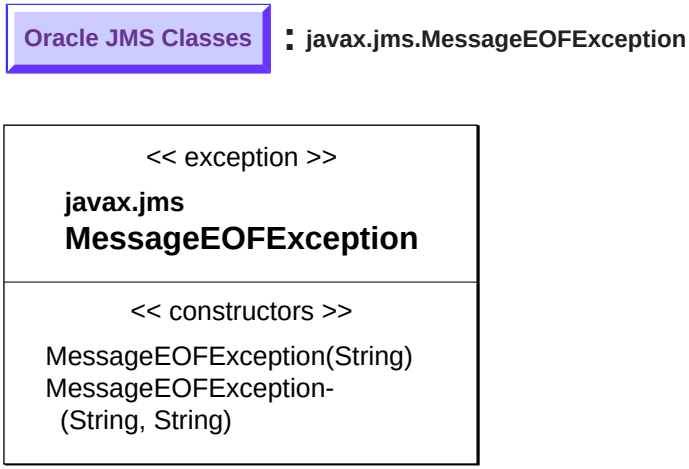
Getting the Error Message for the JMS Exception

Getting the Exception Linked to the JMS Exception

Printing the Stack Trace for the JMS Exception

Exception javax.jms.MessageEOFException

Figure B–44 Exception javax.jms.MessageEOFException

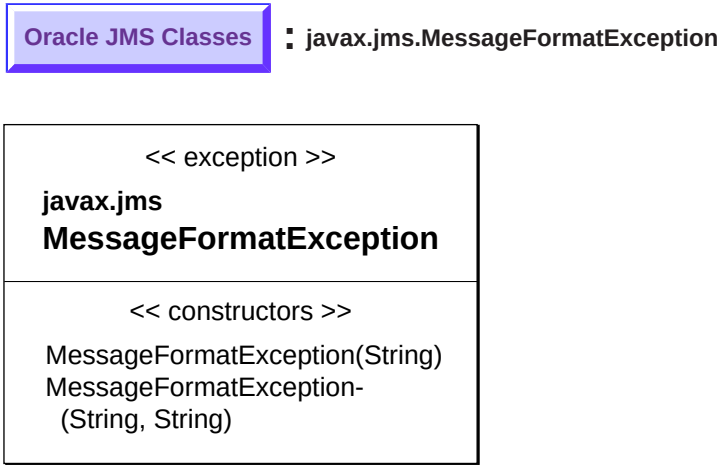


Use Cases

No use case available.

Exception javax.jms.MessageFormatException

Figure B–45 Exception javax.jms.MessageFormatException

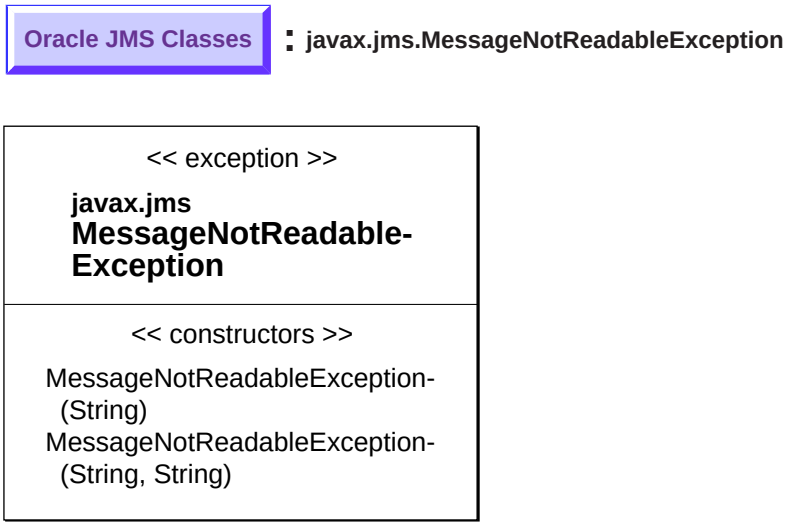


Use Cases

No use case available.

Exception javax.jms.MessageNotReadableException

Figure B–46 Exception javax.jms.MesageNotReadableException

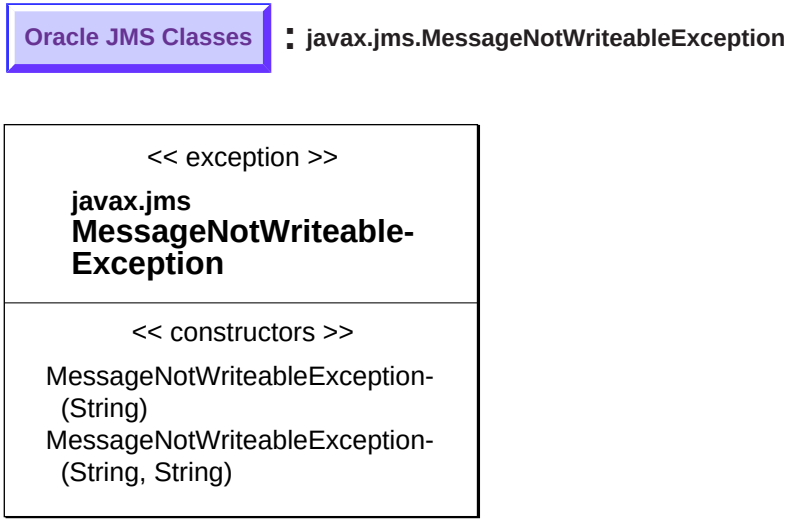


Use Cases

No use case available.

Exception javax.jms.MesageNotWriteableException

Figure B–47 Exception javax.jms.MessageNotWriteableException

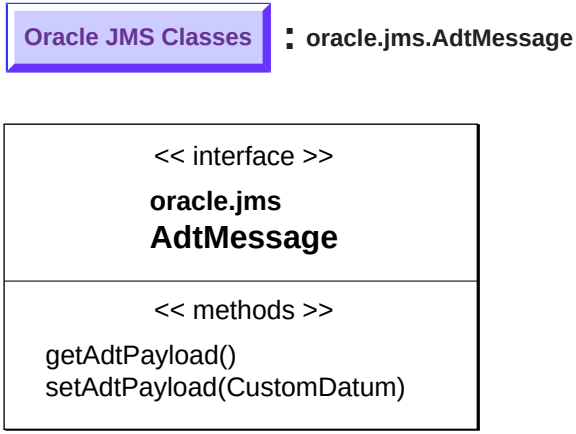


Use Cases

No use case available.

Interface - oracle.jms.AdtMessage

Figure B–48 Interface - oracle.jms.AdtMessage

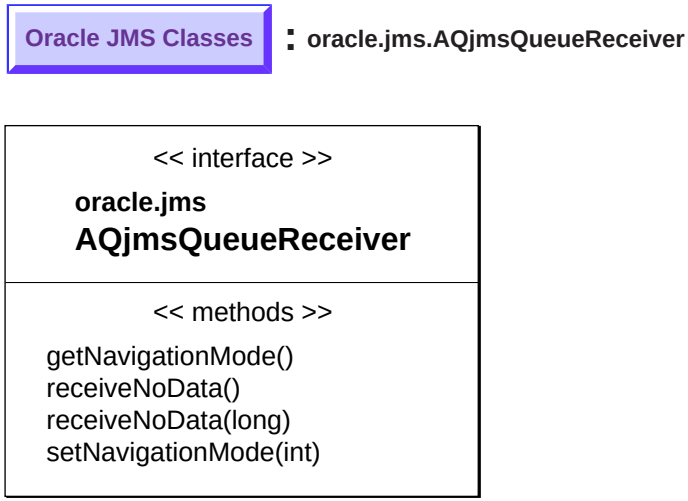


Use Cases

No use case available.

Interface - oracle.jms.AQjmsQueueReceiver

Figure B–49 Interface - oracle.jms.AQjmsQueueReceiver

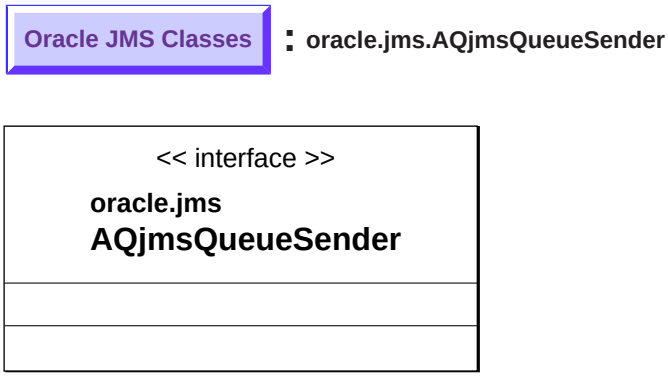


Use Cases

Specifying the Navigation Mode for Receiving Messages

Interface - oracle.jms.AQjmsQueueSender

Figure B–50 Interface - oracle.jms.AQjmsQueueSender

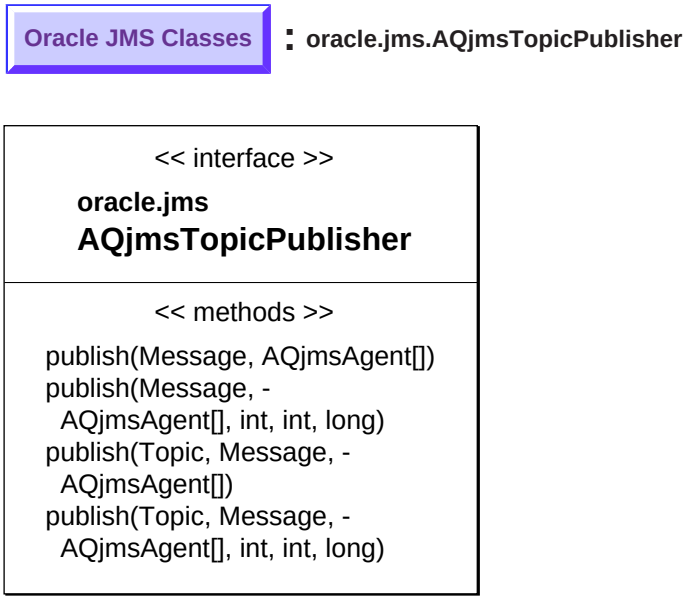


Use Cases

No use case available.

Interface - oracle.jms.AQjmsTopicPublisher

Figure B–51 Interface - oracle.jms.AQjmsTopicPublisher

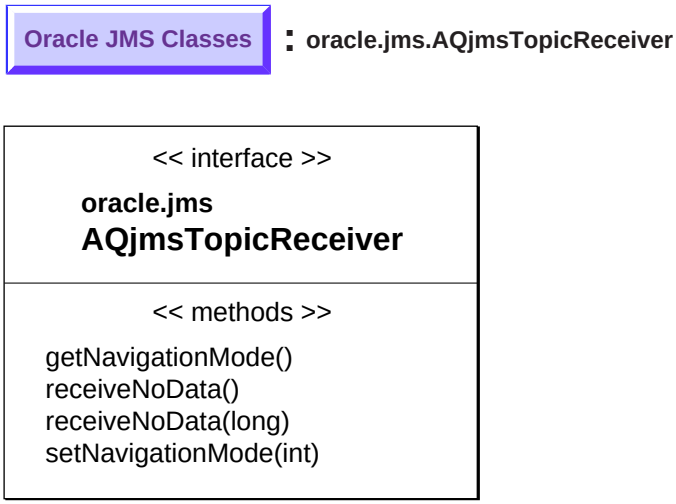


Use Cases

No use case available.

Interface - oracle.jms.TopicReceiver

Figure B–52 Interface - oracle.jms.TopicReceiver

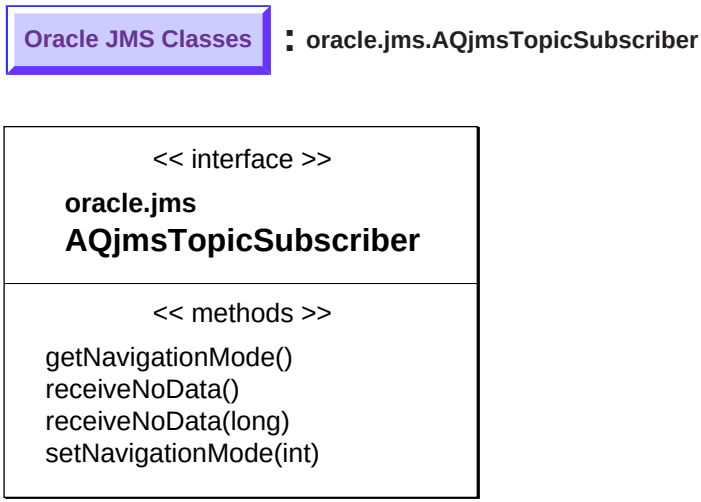


Use Cases

No use case available.

Interface - oracle.jms.AQjmsTopicSubscriber

Figure B–53 Class Diagram: Interface - oracle.jms.AQjmsTopicSubscriber

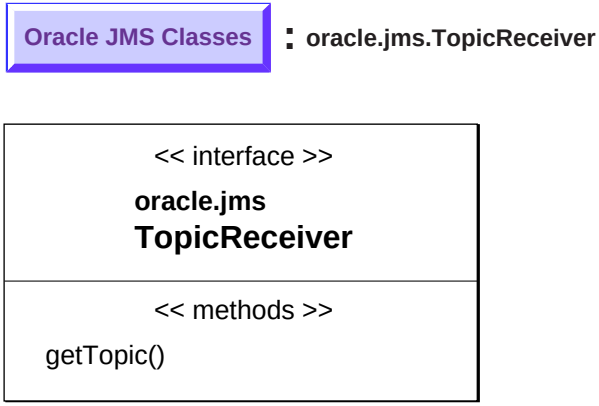


Use Cases

No use case available.

Interface - oracle.jms.AQjmsTopicReceiver

Figure B–54 Interface - oracle.jms.

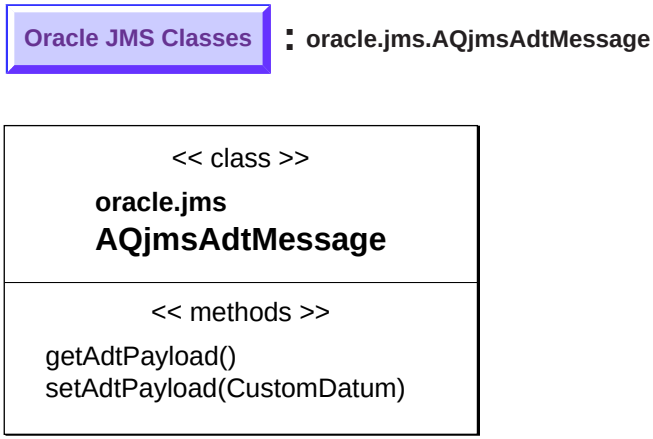


Use Cases

No use case available.

Class - oracle.jms.AQjmsAdtMessage

Figure B–55 Class oracle.jms.AQjmsAdtMessage



Use Cases

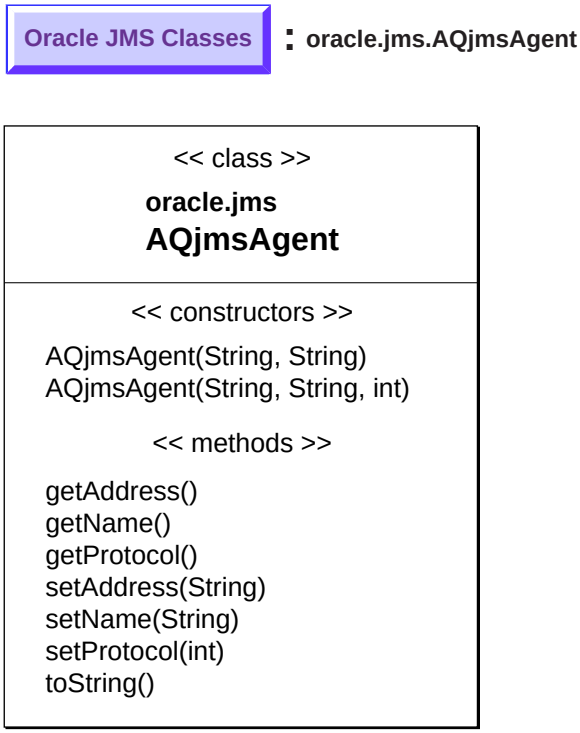
No use case available.

Class - oracle.jms.AQjmsAgent

Figure B–56 Class Diagram: Class - oracle.jms.AQjmsAgent

Use Cases

No use case available.

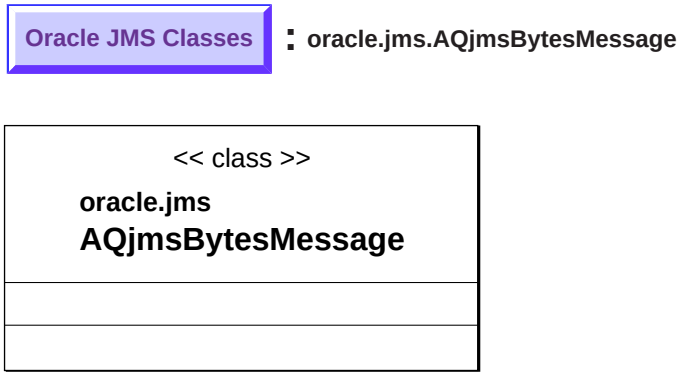


Use Cases

Creating an AQjms Agent

Class - oracle.jms.AQjmsBytesMessage

Figure B–57 Class Diagram: Class - oracle.jms.AQjmsBytesMessage



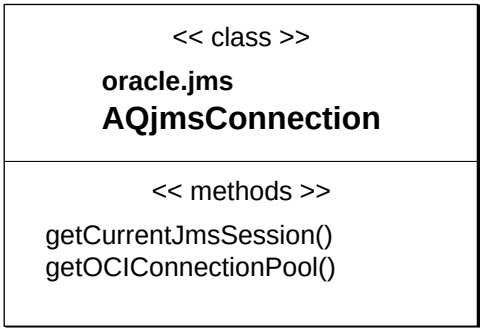
Use Cases

No use case available.

Class - oracle.jms.AQjmsConnection

Figure B–58 Class Diagram: oracle.jms.AQjmsConnection

Oracle JMS Classes : oracle.jms.AQjmsConnection

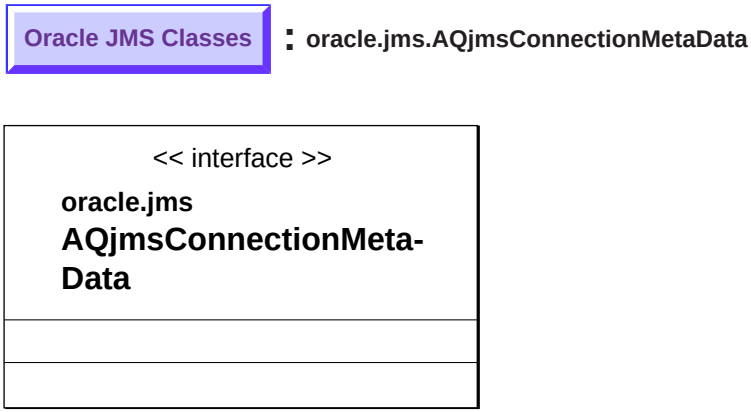


Use Cases

No use case available.

Interface - oracle.jms.AQjmsConnectionMetadata

Figure B–59 Interface - oracle.jms.AQjmsConnectionMetadata

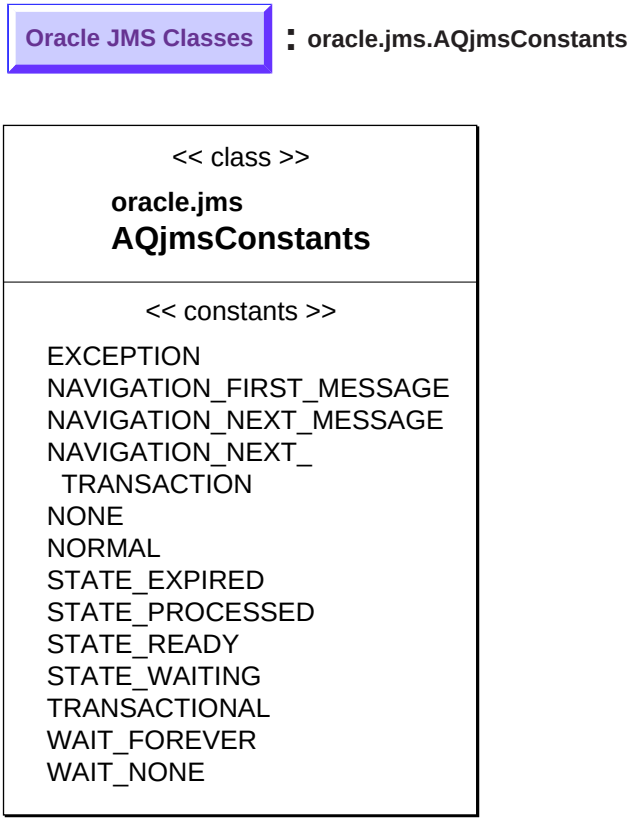


Use Cases

No use case available.

Class - oracle.jms.AQjmsConstants

Figure B–60 Class Diagram: Class - oracle.jms.AQjmsConstants

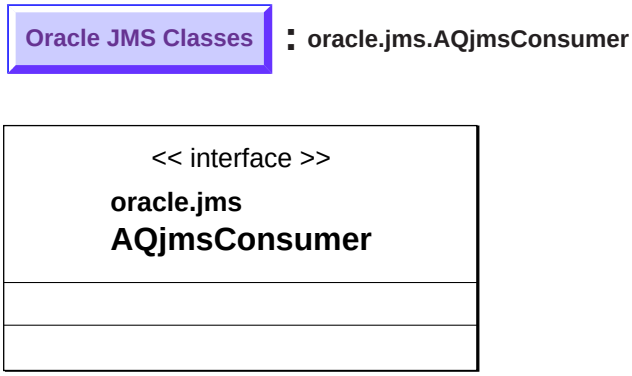


Use Cases

No use case available.

Interface - oracle.jms.AQjmsConsumer

Figure B–61 Interface - oracle.jms.AQjmsConsumer

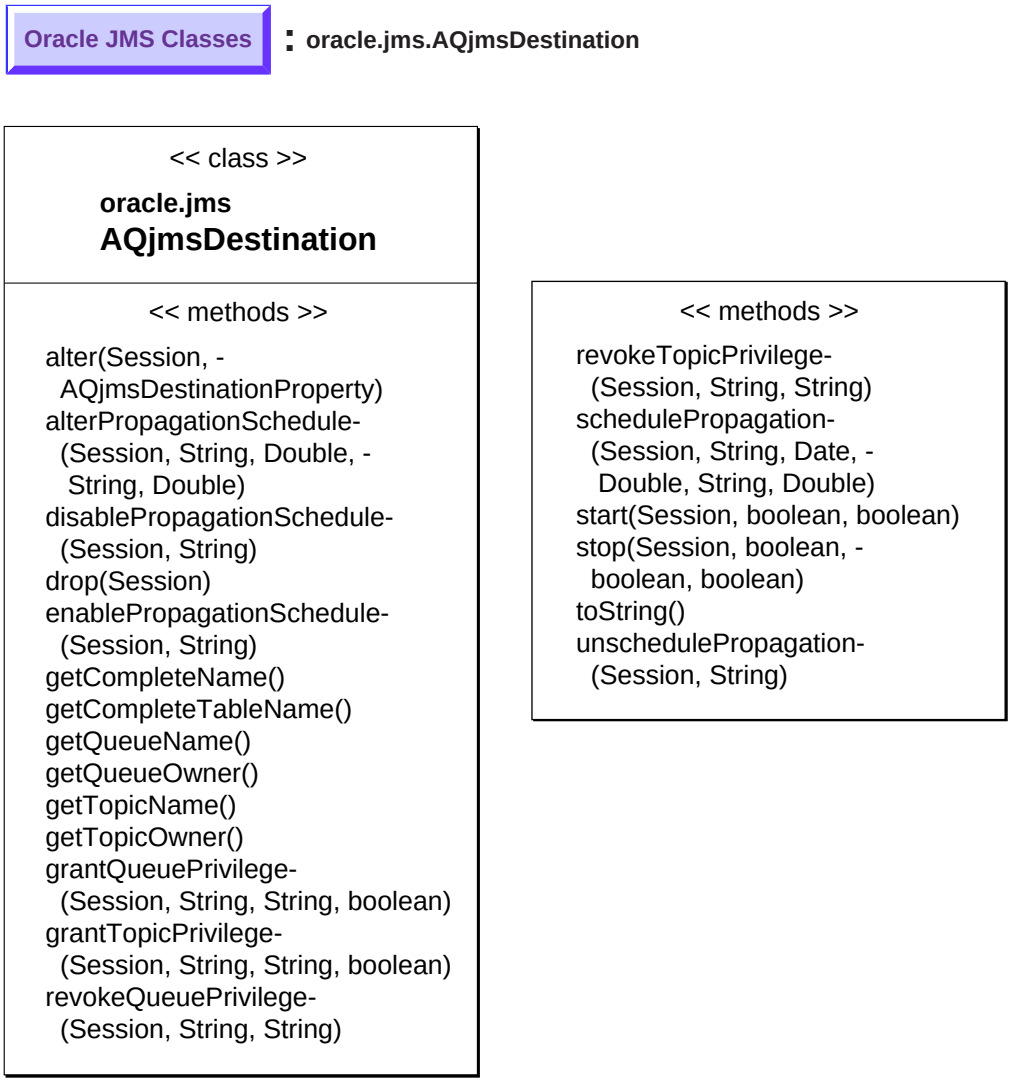


Use Cases

No use case available.

Class - oracle.jms.AQjmsDestination

Figure B–62 Class Diagram: Class - oracle.jms.AQjmsDestination



Use Cases

Publish-Subscribe - Granting Topic Privileges

Publish-Subscribe - Granting Topic Privileges

Starting a Destination

Stopping a Destination

Altering a Destination

Dropping a Destination

Scheduling a Propagation

Enabling a Propagation Schedule

Altering a Propagation Schedule

Disabling a Propagation Schedule

Unscheduling a Propagation

Class - oracle.jms.AQjmsDestinationProperty

Figure B–63 Interface - oracle.jms.AQjmsDestinationProperty

Use Cases

Specifying Destination Properties

Class - oracle.jms.AQjmsFactory

Figure B–64 Class Diagram: Class - oracle.jms. AQjmsFactory

Use Cases

Getting a Queue Connection Factory with JDBC URL

Getting a Queue Connection Factory with JDBC Connection Parameters

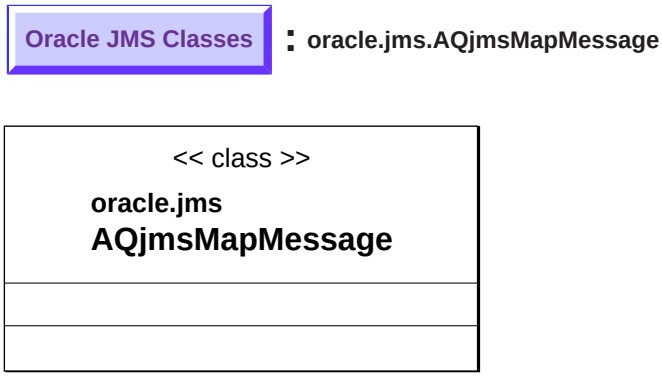
Publish-Subscribe - Two Ways to Create a Topic Connection Factory

Getting a Topic Connection Factory with JDBC URL

Getting a Topic Connection Factory with JDBC Connection Parameters

Class - oracle.jms.AQjmsMapMessage

Figure B–65 Class Diagram: Class - oracle.jms.AQjmsMapMessage

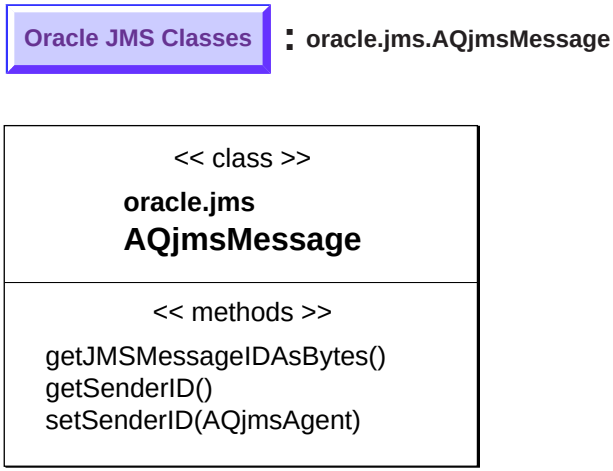


Use Cases

No use case available.

Class - oracle.jms.AQjmsMessage

Figure B–66 Class Diagram: Class - oracle.jms.AQjmsMessage



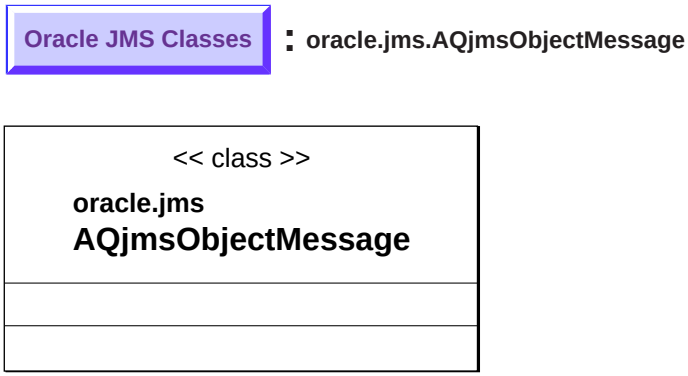
Use Cases

Two Ways of Getting the Message ID of a Message

Getting the Message ID of a Message as Bytes

Class - oracle.jms.AQjmsObjectMessage

Figure B–67 Class Diagram: Class - oracle.jms.AQjmsObjectMessage

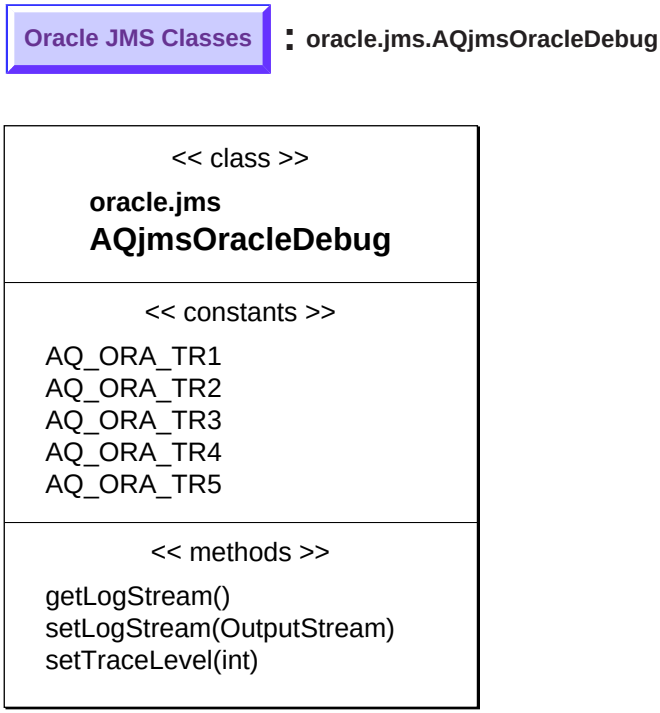


Use Cases

No use case available.

Class - oracle.jms.AQjmsOracleDebug

Figure B–68 Class Diagram: Class - oracle.jms.AQjmsOracleDebug

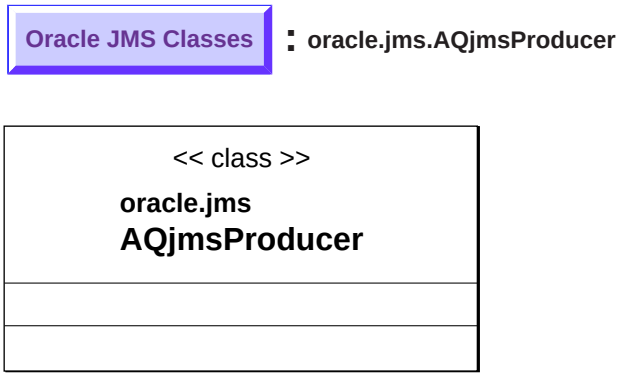


Use Cases

No use case available.

Class - oracle.jms.AQjmsProducer

Figure B–69 Class Diagram: Class - oracle.jms.AQjmsProducer

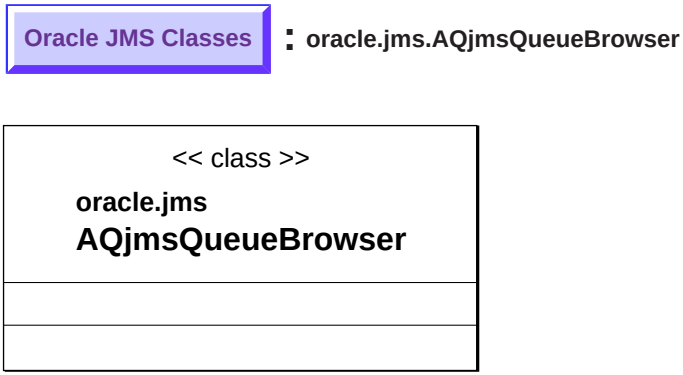


Use Cases

No use case available.

Class - oracle.jms.AQjmsQueueBrowser

Figure B–70 Class Diagram: Class - oracle.jms.AQjmsQueueBrowser

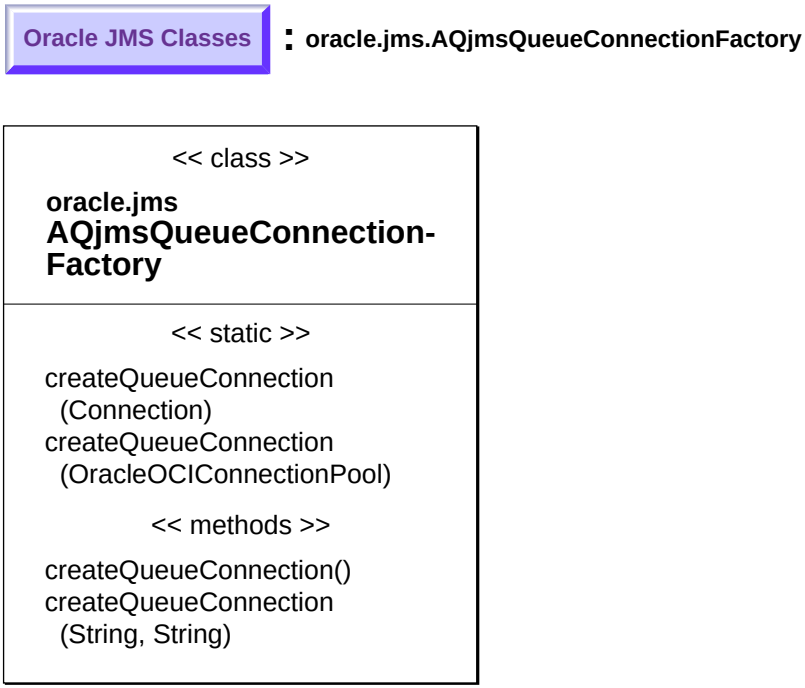


Use Cases

Browsing Messages Using a Queue Browser

Class - AQjmsQueueConnectionFactory

Figure B–71 Class Diagram: Class - oracle.jms.AQjmsQueueConnectionFactory

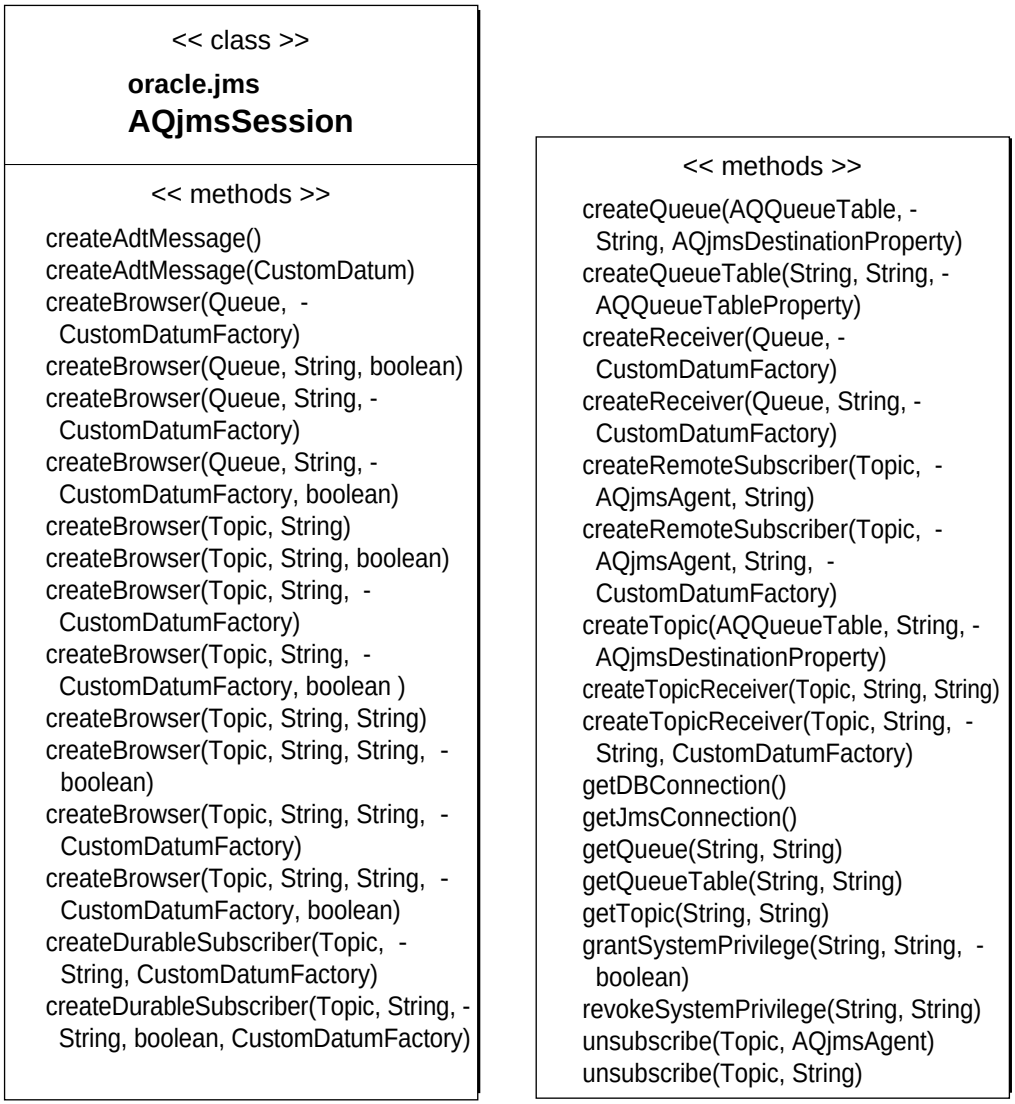


Use Cases

Creating a Queue Connection with Default Connection Factory Parameters

Class - oracle.jms.AQjmsSession

Figure B–72 Class Diagram: Class - oracle.jms.AQjmsSession



Use Cases

Creating a Queue Table

Getting a Queue Table

Publish-Subscribe - Creating a Topic

Granting System Privileges

Revoking System Privileges

Point-to-Point: Granting Queue Privileges

Point-to-Point: Revoking Queue Privileges

Two Ways of Creating a Queue Browser for Oracle Object Type (ADT) Messages Queues

Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages

Creating a Queue Browser for Queues of Oracle Object Type (ADT) Messages, Locking Messages While Browsing

Two Ways of Creating a Queue Receiver

Creating a Queue Receiver for Queues of Oracle Object Type (ADT) Messages

Two Ways of Creating a Durable Subscriber for a Topic of Oracle Object Type (ADT) Messages

Creating a Durable Subscriber for an ADT Topic without Selector

Creating a Durable Subscriber for an ADT Topic with Selector

Creating a Remote Subscriber for Topics of JMS Messages

Creating a Remote Subscriber for Topics of Oracle Object Type (ADT) Messages

Two Ways of Unsubscribing a Durable Subscription

Unsubscribing a Durable Subscription for a Local Subscriber

Unsubscribing a Durable Subscription for a Remote Subscriber

Two Ways of Creating a Topic Receiver

Creating a Topic Receiver for a Topic of Standard JMS Type Messages

Creating a Topic Receiver for a Topic of Oracle Object Type (ADT) Messages

Getting the JMS Connection from a Session

Getting the Underlying JDBC Connection from a JMS Session

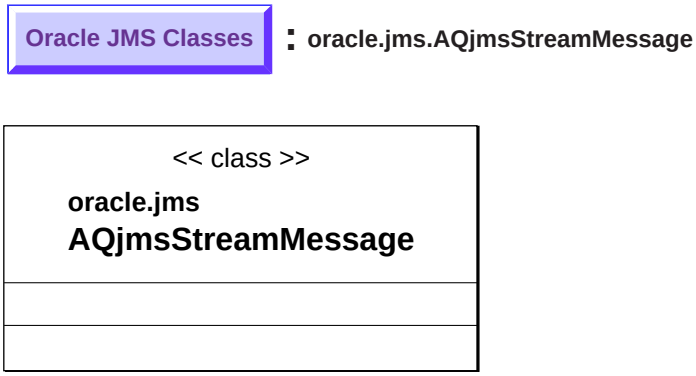
Creating an ADT Message

Two Ways of Creating a Topic Browser for JMS Message Queues

Two Ways of Creating a Topic Browser for Oracle Object Type (ADT) Message Topics

Class - oracle.jms.AQjmsStreamMessage

Figure B–73 Class Diagram: Class - oracle.jms.AQjmsStreamMessage

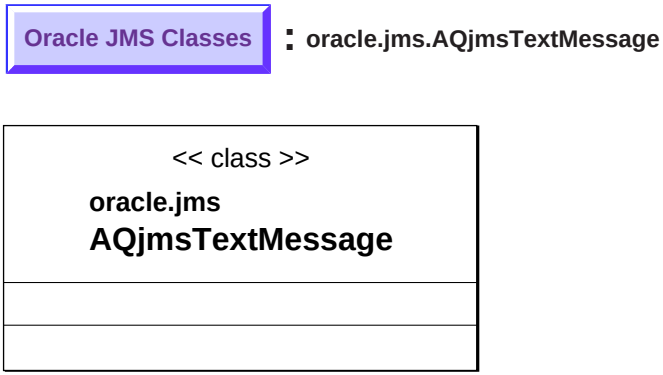


Use Cases

No use case available.

Class - oracle.jms.AQjmsTextMessage

Figure B–74 Class Diagram: Class - oracle.jms.AQjmsTextMessage

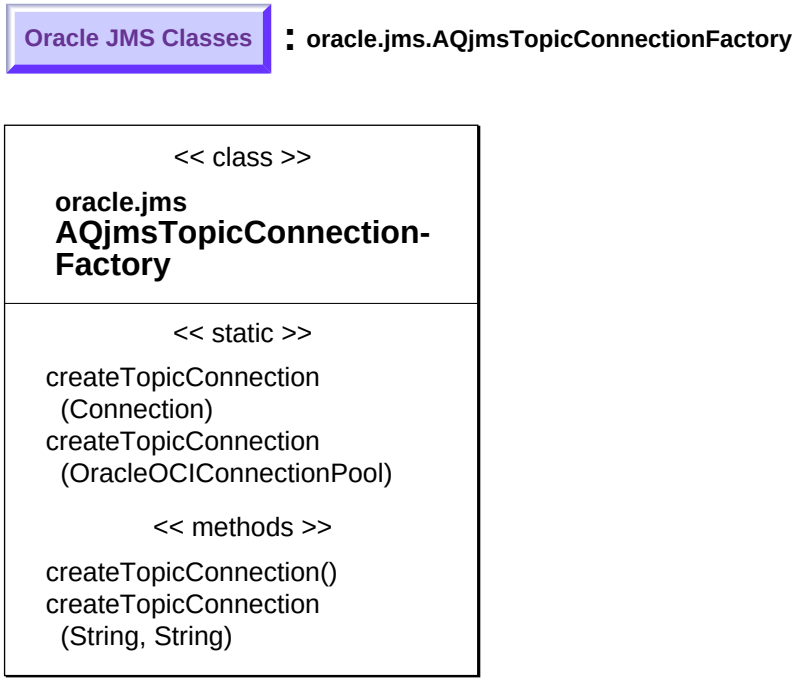


Use Cases

No use case available.

Class - oracle.jms.AQjmsTopicConnectionFactory

Figure B–75 Class oracle.jms.AQjmsTopicConnectionFactory



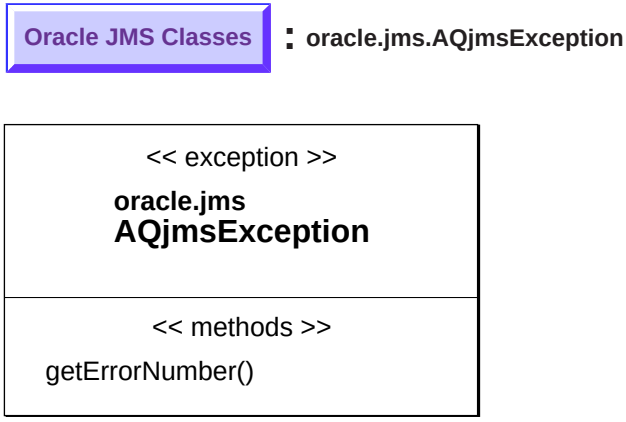
Use Cases

Creating a Topic Connection with Open JDBC Connection

Creating a Topic Connection with Default Connection Factory Parameters

Exception oracle.jms.AQjmsException

Figure B–76 Class oracle.jms.AQjmsException



Use Cases

Getting the Error Number for the JMS Exception

Exception oracle.jms.AQjmsInvalidDestinationException

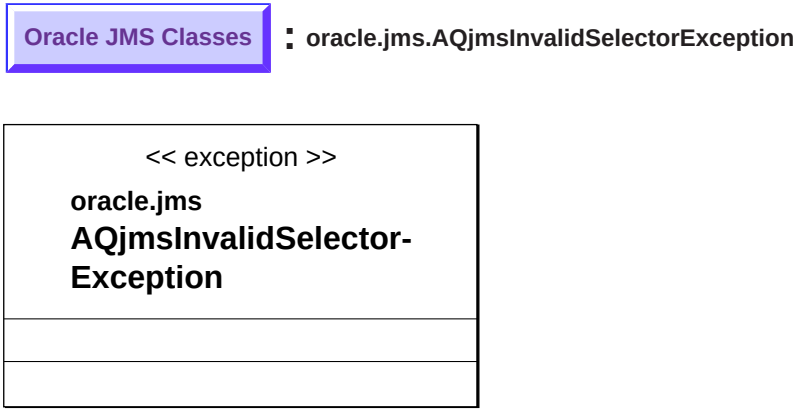
Figure B–77 Exception oracle.jms.AQjmsInvalidDestionationException

Use Cases

No use case available.

Exception oracle.jms.AQjmsInvalidSelectorException

Figure B–78 Exception oracle.jms.AQjmsInvalidSelectorException



Use Cases

No use case available.

Exception oracle.jms.AQjmsMessageEOFException

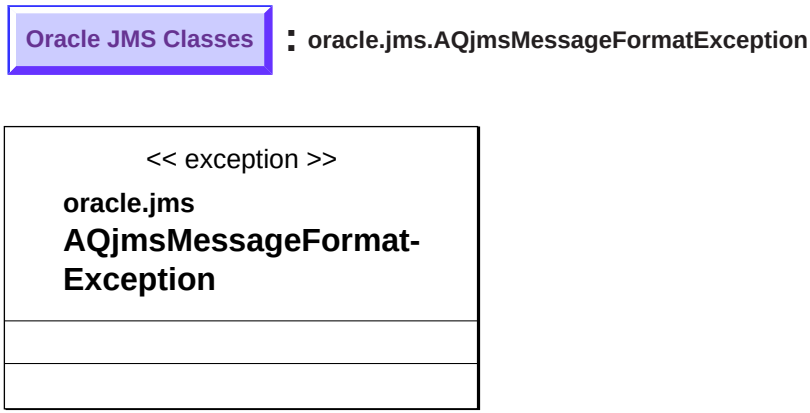
Figure B–79 Exception oracle.jms.AQjmsMessageEOFException

Use Cases

No use case available.

Exception oracle.jms.AQjmsMessageFormatException

Figure B–80 Exception oracle.jms.AQjmsMessageFormatException



Use Cases

No use case available.

Exception oracle.jms.AQjmsMessageNotReadableException

Figure B–81 Exception oracle.jms.AQjmsMessageNotReadableException

Use Cases

No use case available.

Exception oracle.jms.AQjmsMesssageNotWriteableException

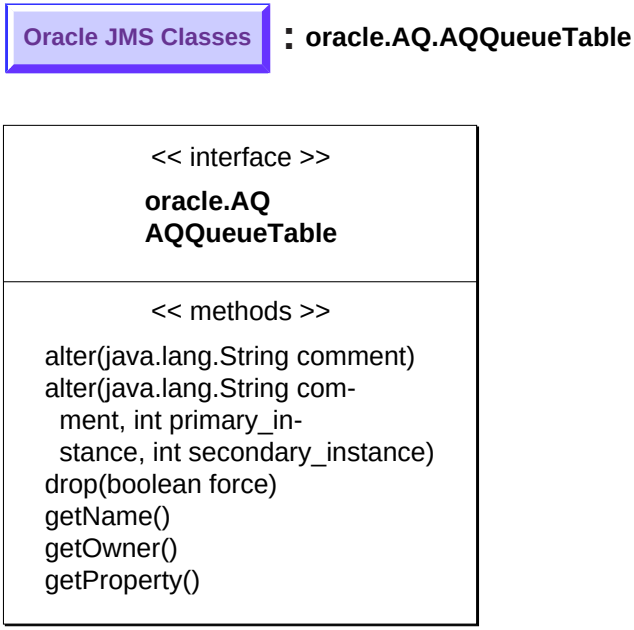
Figure B–82 Exception oracle.jms.AQjmsMessageNotWriteableException

Use Cases

No use case available.

Interface - oracle.AQ.AQQueueTable

Figure B–83 Interface - oracle.AQ.AQQueueTable

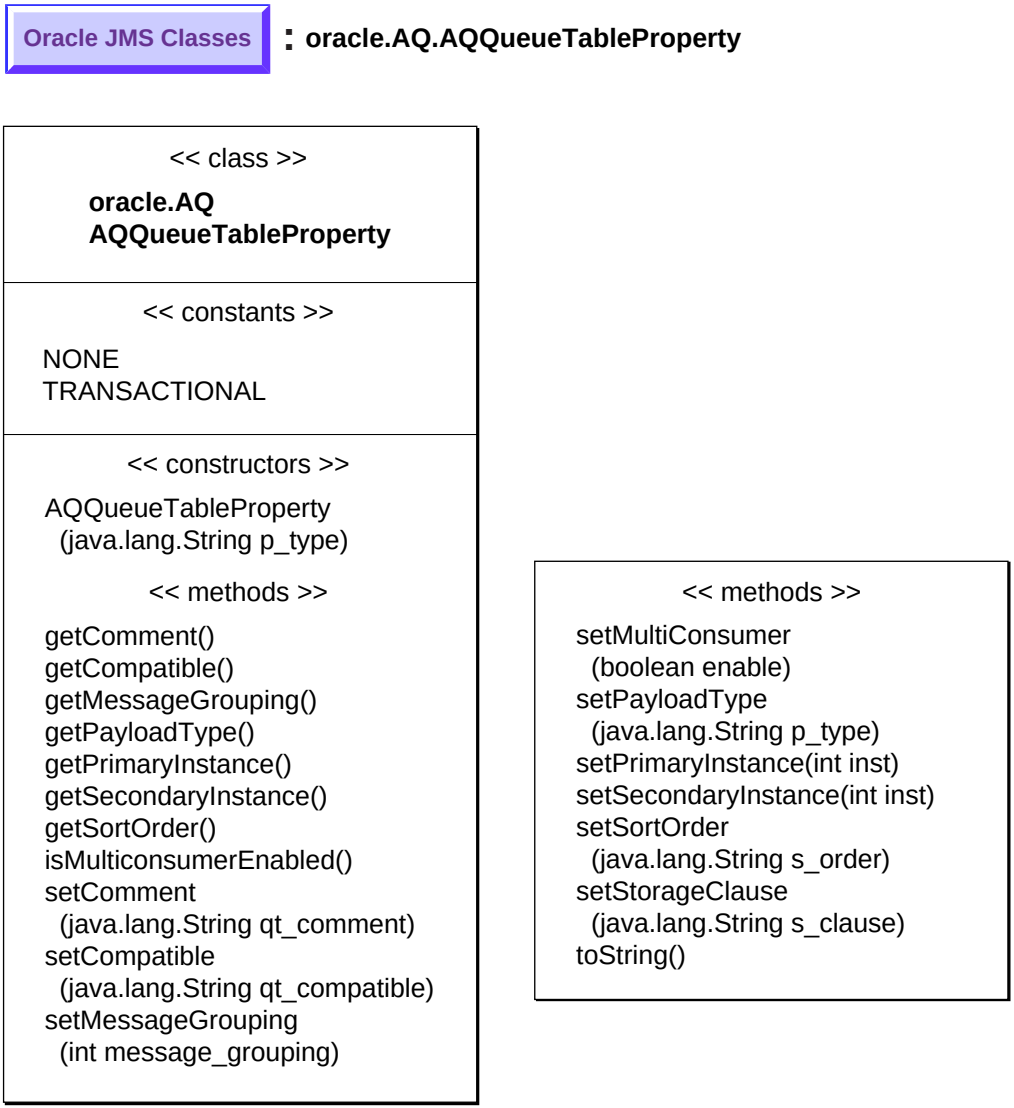


Use Cases

No use case available.

Class - oracle.AQ.AQQueueTableProperty

Figure B–84 Class Diagram: Class - oracle.AQ.AQQueueTableProperty

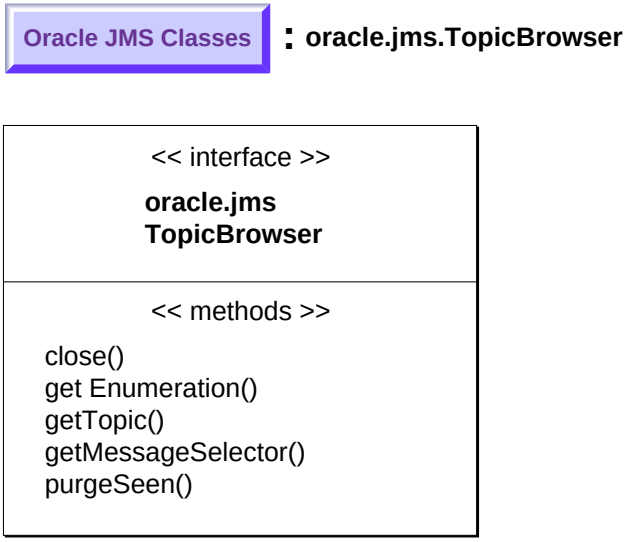


Use Cases

Creating a Queue Table [Specify Queue Table Property]

Interface - oracle.jms.TopicBrowser

Figure B–85 Interface - oracle.jms.TopicBrowser



Use Cases

No use case available.

Class - oracle.jms.AQjmsTopicBrowser

Figure B–86 Class Diagram: Class - oracle.jms.AQjmsTopicBrowser

C

Scripts for Implementing 'BooksOnLine'

This Appendix contains the following scripts:

- tkaqdoca.sql: Script to Create Users, Objects, Queue Tables, Queues & 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 & Subscribers

```
Rem $Header: tkaqdoca.sql 26-jan-99.17:50:37 aquser1 Exp $
Rem
Rem tkaqdoca.sql
Rem
Rem  Copyright (c) Oracle Corporation 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 OS identified by OS;
grant connect, resource to OS;
grant execute on dbms_aq to OS;
grant execute on dbms_aqadm to OS;

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 OS;
grant execute on orderitemlist_vartyp to OS;
grant execute on orderitem_typ to OS;
grant execute on book_typ to OS;
grant execute on customer_typ to OS;
execute dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','OS',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 OS_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 will dequeue from these priority
Rem queues according to the orders' shipping priorities, processes 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 back orders 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 back orders 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 OS/OS;

Rem Create a priority queue table for OS shipping
begin
dbms_aqadm.create_queue_table(
        queue_table => 'OS_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 OS shipping
begin
dbms_aqadm.create_queue_table(
        queue_table => 'OS_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              => 'OS_bookedorders_que',
        queue_table             => 'OS_orders_pr_mqtab');
end;
/

Rem Shipped orders and back orders are stored in the FIFO queue table
begin
dbms_aqadm.create_queue (
        queue_name              => 'OS_shippedorders_que',
        queue_table             => 'OS_orders_mqtab');
end;
/

begin
dbms_aqadm.create_queue (
        queue_name              => 'OS_backorders_que',
        queue_table             => 'OS_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 will return 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', 'OS.OS_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$_agent;
begin
  /* Subscribe to the WS back orders queue */
  subscriber := aq$_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 => 'OS.OS_bookedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'OS.OS_shippedorders_que');
execute dbms_aqadm.start_queue(queue_name => 'OS.OS_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 Corporation 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, etc.>
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.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 Corporation 1998, 1999. All Rights Reserved.
Rem


set echo on

Rem ===============================================================
Rem        Demonstrate enqueuing a backorder with delay time set
Rem        to 1 day.  This will guarantee that each backorder will
Rem        be 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 back order 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 := 'OS.OS_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 Corporation 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
        -- enqueueing 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 := 'OS.OS_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 := 'OS.OS_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 := 'OS.OS_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 OS;
grant execute on shipping_bookedorder_deq to CS;

grant execute on get_rushtitles to ES;

grant execute on get_northamerican_orders to OS;

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 OS/OS;
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 Corporation 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 OS cascade;
drop user CB cascade;
drop user CBADM cascade;
drop user CS cascade;
drop user OE cascade;
drop user boladm cascade;
```

D

JMS and AQ XML Servlet Error Messages

A list of error messages is provided to aid you in troubleshooting problems.

JMS Error Messages

JMS-101 Invalid delivery mode (string)

Cause: The delivery mode is not supported

Action: The valid delivery mode is AQjmsConstants.PERSISTENT

JMS-102 Feature not supported (string)

Cause: This feature is not supported in the current release

Action: Self-explanatory

JMS-104 Message Payload must be specified

Cause: The message payload was null

Action: Specify a non-null payload for the message

JMS-105 Agent must be specified

Cause: AQjmsAgent object was null

Action: Specify a valid AQjmsAgent representing the remote subscriber

JMS-106 Cannot have more than one open Session on a JMSConnection

Cause: There is already one open JMS session on the connection. Cannot have more than one open session on a connection

Action: Close the open session and then open a new one

JMS-107 Operation not allowed on (string)

Cause: The specified operation is not allowed on this object

Action: Self-explanatory

JMS-108 Messages of type (string) not allowed with Destinations containing payload of type (string)

Cause: There was a mismatch between the message type being used and the payload type specified for the destination

Action: Use the message type that maps to the payload specified for the queue table that contains this destination

JMS-109 Class not found: (string)

Cause: The specified class was not found

Action: Make sure your CLASSPATH contains the class

JMS-110 Property (string) not writeable

Cause: An attempt was made to update a read-only message header field or property

Action: Self-explanatory

JMS-111 Connection must be specified

Cause: The connection object was null

Action: Specify a non-null JDBC connection

JMS-112 Connection is invalid

Cause: The JDBC connection is invalid

Action: Specify a non-null oracle JDBC connection

JMS-113 Connection is in stopped state

Cause: An attempt was made to receive messages on a connection that is in stopped state

Action: Start the connection

JMS-114 Connection is closed

Cause: An attempt was made to use a Connection that has been closed

Action: Create a new connection

JMS-115 Consumer is closed

Cause: An attempt was mode to use a Consumer that has been closed

Action: Create a new Message Consumer

JMS-116 Subscriber name must be specified

Cause: Subscriber name was null

Action: Specify a non-null subscription name

JMS-117 Conversion failed - invalid property type

Cause: An error occurred while converting the property to the requested type

Action: Use the method corresponding to the property data type to retrieve it

JMS-119 Invalid Property value

Cause: The property value specified is invalid

Action: Use the appropriate type of value for the property being set

JMS-120 Dequeue failed

Cause: An error occurred while receiving the message

Action: See message inside the JMSException and linked SQLException for more information

JMS-121 DestinationProperty must be specified

Cause: A null AQjmsDestinationProperty was specified while creating a queue/topic

Action: Specify a non-null AQjmsDestinationProperty for the destination

JMS-122 Internal error (string)

Cause: Internal error occurred

Action: Call Support

JMS-123 Interval must be at least (integer) seconds

Cause: An invalid interval was specified

Action: The interval must be greater than 30 seconds

JMS-124 Invalid Dequeue mode

Cause: Invalid dequeue mode was specified

Action: Valid Dequeue modes are AQConstants.DEQUEUE_BROWSE, AQConstants.DEQUEUE_REMOVE, AQConstants.DEQUEUE_LOCKED, AQConstants.DEQUEUE_REMOVE_NODATA

JMS-125 Invalid Queue specified

Cause: An invalid Queue object was specified

Action: Specify a valid Queue handle

JMS-126 Invalid Topic specified

Cause: An invalid Topic object was specified

Action: Specify a valid Topic handle

JMS-127 Invalid Destination

Cause: An invalid destination object was specified

Action: Specify a valid destination (Queue/Topic) object

JMS-128 Invalid Navigation mode

Cause: An invalid navigation mode was specified

Action: The valid navigation modes are AQjmsConstants.NAVIGATION_FIRST_MESSAGE, AQjmsConstants.NAVIGATION_NEXT_MESSAGE, AQjmsConstants.NAVIGATION_NEXT_TRANSACTION

JMS-129 Invalid Payload type

Cause: There was a mismatch between the message type being used and the payload type specified for the destination

Action: Use the message type that maps to the payload specified for the queue table that contains this destination. For ADT messages, use the appropriate CustomDatum factory to create the message consumer

JMS-130 JMS queue cannot be multi-consumer enabled

Cause: An attempt was made to get a AQ multi-consumer queue as a JMS queue

Action: JMS queues cannot be multi-consumer enabled

JMS-131 Session is closed

Cause: An attempt was made to use a session that has been closed

Action: Open a new session

JMS-132 Maximum number of properties (integer) exceeded

Cause: Maximum number of user defined properties for the message has been exceeded

Action: Self-explanatory

JMS-133 Message must be specified

Cause: Message specified was null

Action: Specify a non-null message

JMS-134 Name must be specified

Cause: Queue or Queue table Name specified was null

Action: Specify a non-null name

JMS-135 Driver (string) not supported

Cause: The specified driver is not supported

Action: Valid drivers are oci8 and thin. To use the kprb driver get the kprb connection using getDefaultConnection() and use the static createTopicConnection and createQueueConnection methods

JMS-136 Payload factory can only be specified for destinations with ADT payloads

Cause: A CustomDatumFactory was specified for consumers on destinations not containing ADT payloads

Action: This field must be set to null for destinations containing payloads of type SYS.AQ$_JMS_TEXT_MESSAGE, SYS.AQ$_JMS_BYTES_MESSAGE, SYS.AQ$_JMS_MAP_MESSAGE, SYS.AQ$_JMS_OBJECT_MESSAGE, SYS.AQ$_JMS_STREAM_MESSAGE

JMS-137 Payload factory must be specified for destinations with ADT payloads

Cause: CustomDatumFactory was not specified for destinations containing ADT payloads

Action: For destinations containing ADT messages, a CustomDatumFactory for a java class that maps to the SQL ADT type of the destination must be specified

JMS-138 Producer is closed

Cause: An attempt was made to use a producer that has been closed

Action: Create a new Message Producer

JMS-139 Property name must be specified

Cause: Property name was null

Action: Specify a non-null property name

JMS-140 Invalid System property

Cause: Invalid system property name specified.

Action: Specify one of the valid JMS system properties

JMS-142 JMS topic must be created in multi-consumer enabled queue tables

Cause: An attempt was made to create a JMS topic in a single-consumer queue table

Action: JMS topics can only be created in queue tables that are multi-consumer enabled

JMS-143 Queue must be specified

Cause: Null queue was specified

Action: Specify a non-null queue

JMS-144 JMS queue cannot be created in multiconsumer enabled queue tables

Cause: An attempt was made to create a JMS queue in a multi-consumer queue table

Action: JMS queues can only be created in queue tables that are not multi-consumer enabled

JMS-145 Invalid recipient list

Cause: The recipient list specified was empty

Action: Specify a recipient list with at least one recipient

JMS-146 Registration failed

Cause: An error occurred while registering the type in the type map

Action: Self-explanatory

JMS-147 Invalid ReplyTo destination type

Cause: The ReplyTo destination object type is invalid

Action: The ReplyTo destination must be of type AQjmsAgent

JMS-148 Property name size exceeded

Cause: The property name is greater than the maximum size

Action: Specify a property name that is less than 100 characters

JMS-149 Subscriber must be specified

Cause: Subscriber specified was null

Action: Specify a non-null subscriber

JMS-150 Property not supported

Cause: An attempt was made to use a property that is not supported

Action: Self-explanatory

JMS-151 Topics cannot be of type EXCEPTION

Cause: Topics cannot be of type AQjmsConstants.EXCEPTION

Action: Specify topics to be of type AQjmsConstants.NORMAL

JMS-153 Invalid System property type

Cause: The type of the value specified does not match the type defined for the system property being set

Action: Use the correct type for the setting the system property

JMS-154 Invalid value for sequence deviation

Cause: The sequence deviation is invalid

Action: Valid values are AQEnqueueOption.DEVIATION_BEFORE, AQEnqueueOption.DEVIATION_TOP

JMS-155 AQ Exception (string)

Cause: An error occurred in the AQ java layer

Action: See the message inside the JMSException and the linked exception for more information

JMS-156 Invalid Class (string)

Cause: Class specified is invalid

Action: make sure your CLASSPATH has the specified class

JMS-157 IO Exception (string)

Cause: IO exception

Action: See message is JMSException for details

JMS-158 SQL Exception (string)

Cause: SQL Exception

Action: See message inside linked SQLException for details

JMS-159 Invalid selector (string)

Cause: The selector specified is either invalid or too long

Action: Check the syntax of the selector

JMS-160 EOF Exception (string)

Cause: EOF exception occurred while reading the byte stream

Action: Self-explanatory

JMS-161 MessageFormat Exception: (string)

Cause: An error occurred while converting the stream data to specified type

Action: check the type of data expected on the stream and use the appropriate read method

JMS-162 Message not Readable

Cause: Message is in write-only mode

Action: Call the reset method to make the message readable

JMS-163 Message not Writeable

Cause: Message is in read-only mode

Action: Use the clearBody method to make the message writeable

JMS-164 No such element

Cause: Element with specified name was not found in the map message

Action: Self-explanatory

JMS-165 Maximum size of property value exceeded

Cause: The property value exceeded the maximum length allowed

Action: Values for JMS defined properties can be a maximum of length of 100, Values for User defined properties can have a maximum length of 2000

JMS-166 Topic must be specified

Cause: Topic specified was null

Action: Specify a non-null topic

JMS-167 Payload factory or Sql_data_class must be specified

Cause: Payload factory or Sql_data_class not specified for queues containing object payloads

Action: Specify a CustomDatumFactory or the SQLData class of the java object that maps to the ADT type defined for the queue.

JMS-168 Cannot specify both payload factory and sql_data_class

Cause: Both CustomDatumFactory and SQLData class were specified during dequeue

Action: Specify either the CustomDatumFactory or the SQLData class of the java object that maps to the ADT type defined for the queue.

JMS-169 Sql_data_class cannot be null

Cause: SQLData class specified is null

Action: Specify the SQLData class that maps to the ADT type defined for the queue

JMS-171 Message is not defined to contain (string)

Cause: Invalid payload type in message

Action: Check if the queue is defined to contain RAW or OBJECT payloads and use the appropriate payload type in the message

JMS-172 More than one queue table matches query (string)

Cause: More than one queue table matches the query

Action: Specify both owner and queue table name

JMS-173 Queue Table (string) not found

Cause: The specified queue table was not found

Action: Specify a valid queue table

JMS-174 Class must be specified for queues with object payloads\n. Use dequeue(deq_option,payload_fact) or dequeue(deq_option, sql_data_cl)

Cause: This dequeue method cannot be used to dequeue from queues with OBJECT payloads

Action: use the either dequeue(deq_option, payload_fact) or dequeue(deq_option, sql_data_cl)

JMS-175 DequeueOption must be specified

Cause: DequeueOption specified is null

Action: Specify a non-null dequeue option

JMS-176 EnqueueOption must be specified

Cause: EnqueueOption specified is null

Action: Specify a non-null enqueue option

JMS-177 Invalid payload type: Use dequeue(deq_option) for raw payload queues

Cause: This method cannot be used to dequeue from queues with RAW payload

Action: use the dequeue(deq_option) method

JMS-178 Invalid Queue name - (string)

Cause: The queue name specified is null or invalid

Action: Specify a queue name that is not null. The queue name must not be qualified with the schema name. The schema name must be specified as the value of the owner parameter

JMS-179 Invalid Queue Table name - (string)

Cause: The queue table name specified is null or invalid

Action: Specify a queue table name that is not null. The queue table name must not be qualified with the schema name. The schema name must be specified as the value of the owner parameter

JMS-180 Invalid Queue Type

Cause: Queue type is invalid

Action: Valid types are AQConstants.NORMAL or AQConstants.EXCEPTION

JMS-181 Invalid value for wait_time

Cause: Invalid value for wait type

Action: Wait time can be AQDequeueOption.WAIT_FOREVER, AQDequeueOption.WAIT_NONE or any value greater than 0

JMS-182 More than one queue matches query

Cause: More than one queue matches query

Action: Specify both the owner and name of the queue

JMS-183 No AQ driver registered

Cause: No AQDriver registered

Action: Make sure that the AQ java driver is registered. Use Class.forName("oracle.AQ.AQOracleDriver")

JMS-184 Queue object is invalid

Cause: The queue object is invalid

Action: The underlying JDBC connection may have been closed. Get the queue handle again

JMS-185 QueueProperty must be specified

Cause: AQQueueProperty specified is null

Action: Specify a non-null AQQueueProperty

JMS-186 QueueTableProperty must be specified

Cause: QueueTableProperty specified is null

Action: Specify a non-null AQQueueTableProperty

JMS-187 Queue Table must be specified

Cause: Queue Table specified is null

Action: Specify a non-null queue table

JMS-188 QueueTable object is invalid

Cause: The queue table object is invalid

Action: The underlying JDBC connection may have been closed. Get the queue table handle again

JMS-189 Byte array too small

Cause: The byte array given is too small to hold the data requested

Action: Specify a byte array that is large enough to hold the data requested or reduce the length requested

JMS-190 Queue (string) not found

Cause: The specified queue was not found

Action: Specify a valid queue

JMS-191 sql_data_cl must be a class that implements SQLData interface

Cause: The class specified does not support the java.sql.SQLData interface

Action: Self-explanatory

JMS-192 Invalid Visibility value

Cause: Visibility value specified is invalid

Action: Valid values areAQConstants.VISIBILITY_ONCOMMIT, AQConstants.VISIBILITY_IMMEDIATE

JMS-193 JMS queues cannot contain payload of type RAW

Cause: An attempt was made to create a JMS queue with RAW payload

Action: JMS queues/topics cannot contain RAW payload

JMS-194 Session object is invalid

Cause: Session object is invalid

Action: The underlying JDBC connection may have been closed. Create a new session

JMS-195 Invalid object type: object must implement CustomDatum or SQLData interface

Cause: Invalid object type specified

Action: object must implement CustomDatum or SQLData interface

JMS-196 Cannot have more than one open QueueBrowser for the same destination on a JMS Session

Cause: There is already an open QueueBrowser for this queue on this session

Action: There cannot be more than one queue browser for the same queue in a particular session. Close the existing QueueBrowser and then open a new one

JMS-197 Agent address must be specified for remote subscriber

Cause: Address field is null for remote subscriber

Action: The address field must contain the fully qualified name of the remote topic

JMS-198 Invalid operation: Privileged message listener set for the Session

Cause: The client tried to use a message consumer to receive messages when the session message listener was set.

Action: Use the session's message listener to consume messages. The consumer's methods for receiving messages must not be used.

JMS-199 Registration for notification failed

Cause: Listener Registration failed

Action: See error message in linked Exception for details

JMS-200 Destination must be specified

Cause: Destination is null

Action: Specify a non-null destination

JMS-201 All Recipients in recipient_list must be specified

Cause: One or more elements in the recipient list are null

Action: All AQjmsAgents in the recipient list must be specified

JMS-202 Unregister for asynchronous receipt of messages failed

Cause: An error occurred while removing the registration of the consumer with the database for asynchronous receipt

Action: Check error message in linked exception for details

JMS-203 Payload Factory must be specified

Cause: Null Payload Factory was specified

Action: Specify a non null payload factory

JMS-204 An error occurred in the AQ JNI layer

Cause: JNI Error

Action: Check error message in linked exception for details

JMS-205 Naming Exception

Cause: Naming exception

Action: Check error message in linked exception for details

JMS-206 XA Exception XAError-{0} :: OracleError-{1}

Cause: An error occurred in the XA layer

Action: See the message inside the linked XAException for more information

JMS-207 JMS Exception {0}

Cause: An error occurred in the JMS layer

Action: See the message inside the linked JMSException for more information

JMS-208 XML SQL Exception

Cause: An error occurred in the XML SQL layer

Action: See the message inside the linked AQxmlException for more information

JMS-209 XML SAX Exception

Cause: An error occurred in the XML SAX layer

Action: See the message inside the linked AQxmlException for more information

JMS-210 XML Parse Exception

Cause: An error occurred in the XML Parser layer

Action: See the message inside the linked AQxmlException for more information

JMS-220 Connection no longer available

Cause: Connection to the database no longer available.

Comment: This may happen if the database / network / machine is not accessible. This may be a transient failure.

JMS-221 Free physical database connection unavailable in connection pool

Cause: A free physical database connection was not available in the OCI connection pool in order to perform the specified operation.

Action: Try performing the operation later

AQ XML Servlet Error Messages

JMS-400 Destination name must be specified

Cause: A null Destination name was specified

Action: Specify a non-null destination name

JMS-402 Class not found: {0}

Cause: The specified class was not found

Action: Make sure your CLASSPATH contains the class specified in the error message

JMS-403 IO Exception {0}

Cause: IO exception

Action: See the message inside the linked AQxmlException for more information

JMS-404 XML Parse Exception

Cause: An error occurred in the XML Parser layer

Action: See the message inside the linked AQxmlException for more information

JMS-405 XML SAX Exception

Cause: An error occurred in the XML SAX layer

Action: See the message inside the linked AQxmlException for more information

JMS-406 JMS Exception {0}

Cause: An error occurred in the JMS layer

Action: See the message inside the linked JMSException for more information

JMS-407 Operation not allowed on {0}

Cause: The specified operation is not allowed on this object

Action: Check that the user performing the operation has the required privileges

JMS-408 Conversion failed - invalid property type

Cause: An error occurred while converting the property to the requested type

Action: Use the method corresponding to the property data type to retrieve it

JMS-409 No such element

Cause: Element with specified name was not found in the map message

Action: Specify a valid element name

JMS-410 XML SQL Exception

Cause: An error occurred in the JDBC SQL layer

Action: See the message inside the linked SQLException for more information

JMS-411 Payload body cannot be null

Cause: An invalid body string or document was specified

Action: Specify a non-null body string or document for the payload

JMS-412 Byte conversion failed

Cause: An invalid username/password was specified

Action: Specify a non-null username and password

JMS-413 Autocommit not allowed for operation

Cause: The autocommit flag cannot be set for this operation

Action: Do not set the autocommit flag

JMS-414 Destination owner must be specified

Cause: A null Destination owner was specified

Action: Specify a non-null destination name

JMS-415 Invalid Visibility value

Cause: Visibility value specified is invalid

Action: Valid values are AQxmlConstants.VISIBILITY_ONCOMMIT, AQxmlConstants.VISIBILITY_IMMEDIATE

JMS-416 Invalid Dequeue mode

Cause: Invalid dequeue mode was specified

Action: Valid Dequeue modes are AQxmlConstants.DEQUEUE_BROWSE, AQxmlConstants.DEQUEUE_REMOVE, AQxmlConstants.DEQUEUE_LOCKED, AQxmlConstants.DEQUEUE_REMOVE_NODATA

JMS-417 Invalid Navigation mode

Cause: An invalid navigation mode was specified

Action: The valid navigation modes are:
AQxmlConstants.NAVIGATION_FIRST_MESSAGE
AQxmlConstants.NAVIGATION_NEXT_MESSAGE
AQxmlConstants.NAVIGATION_NEXT_TRANSACTION

JMS-418 Invalid value for wait_time

Cause: Invalid value for wait type

Action: Wait time can be AQDequeueOption.WAIT_FOREVER, AQDequeueOption.WAIT_NONE, or any value greater than 0

JMS-419 Invalid ConnectionPoolDataSource

Cause: A null or invalid ConnectionPoolDataSource was specified

Action: Specify a valid OracleConnectionPoolDataSource object with the correct URL and user/password

JMS-420 Invalid value for cache_size

Cause: An invalid cache_size was specified

Action: Cache size must be greater than 0

JMS-421 Invalid value for cache_scheme

Cause: An invalid cache scheme was specified

Action: The valid cache schemes are:
OracleConnectionCacheImpl.DYNAMIC_SCHEME
OracleConnectionCacheImpl.FIXED_WAIT_SCHEME

JMS-422 Invalid tag - {0}

Cause: An invalid tag was encountered in the XML document

Action: Verify that the XML document conforms to the AQ schema

JMS-423 Invalid value

Cause: An invalid value was specified

Action: Verify that the value specified in the XML document conforms to those specified in the AQ schema

JMS-424 Invalid message header

Cause: The message header specified is null or invalid

Action: Specify a valid message header

JMS-425 Property name must be specified

Cause: Property name was null

Action: Specify a non-null property name

JMS-426 Property does not exist

Cause: Invalid property name specified. The property does not exist

Action: The property does not exist

JMS-427 Subscriber name must be specified

Cause: Subscriber name was null

Action: Specify a non-null subscription name

JMS-428 Valid message must be specified

Cause: message was null

Action: Specify a non-null message

JMS-429 Register Option must be specified

Cause: Register option is null

Action: Specify a non-null Register Option

JMS-430 Database Link must be specified

Cause: DB Link is null

Action: Specify a non-null Register Option

JMS-431 Sequence Number must be specified

Cause: Register option is null

Action: Specify a non-null Register Option

JMS-432 Status must be specified

Cause: status option is null

Action: Specify a non-null Register Option

JMS-433 User not authenticated

Cause: User is not authenticated

Action: Check that the user was authenticated by the webserver before connecting to the Servlet

JMS-434 Invalid data source

Cause: Data source is null or invalid

Action: Specify a valid data source for connecting to the database

JMS-435 Invalid schema location

Cause: Schema location is null or invalid

Action: Specify a valid URL for the schema

JMS-436 AQ Exception

Cause: An error occurred in the AQ java layer

Action: See the message inside the AQxmlException and the linked exception for more information

JMS-437 Invalid Destination

Cause: An invalid destination object was specified

Action: Specify a valid destination (Queue/Topic) object

JMS-438 AQ agent {0} not mapped to a valid database user

Cause: The AQ agent specified does not map to a database user which has privileges to perform the requested operation

Action: Use dbms_aqadm.enable_db_access to map the agent to a database user with the required queue privileges

JMS-439 Invalid schema document

Cause: The schema document specified is not valid

Action: Specify a valid URL for the schema document

JMS-440 Invalid operations - agent {0} maps to more than one database user

Cause: The AQ agent mapped to more than one database user in the same session

Action: Map the AQ agent to only one database user. Check the `aq$internet_users` view for database users that map to this agent.

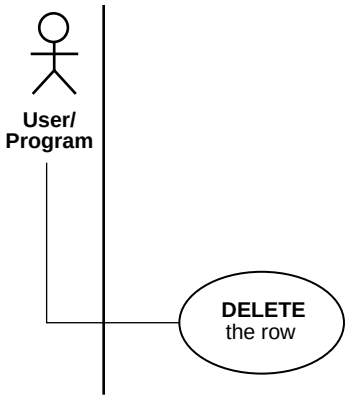
E

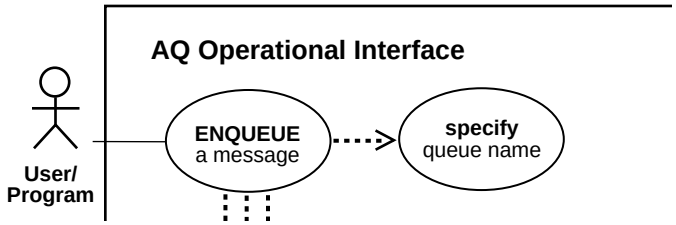
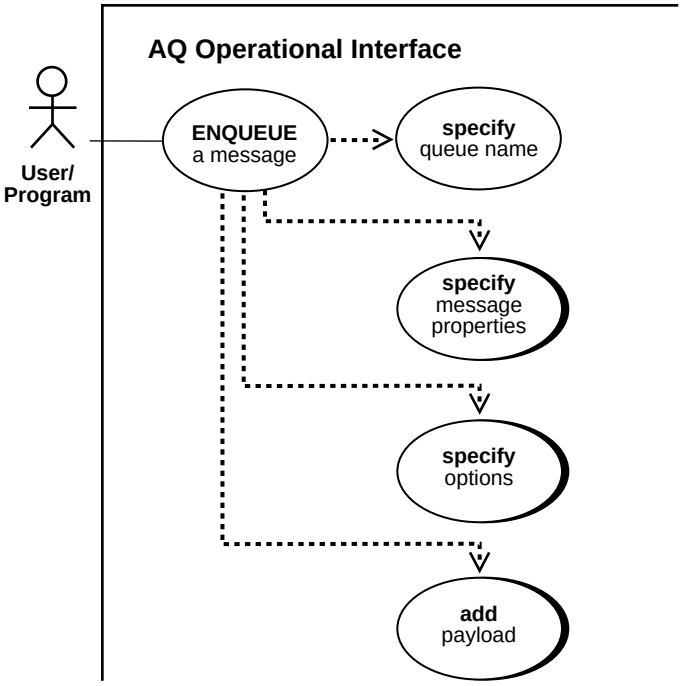
Interpreting Unified Modeling Language Diagrams

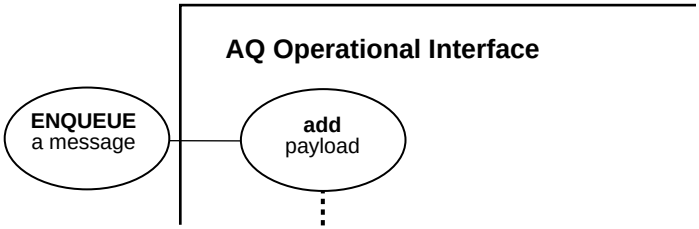
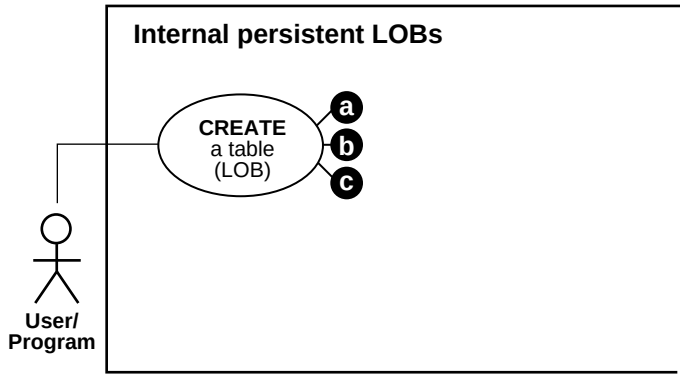
This manual uses the Unified Modeling Language (UML) to present use case diagrams as a way of explaining technology. A brief explanation of use case diagrams and UML notation follows.

See Also: *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

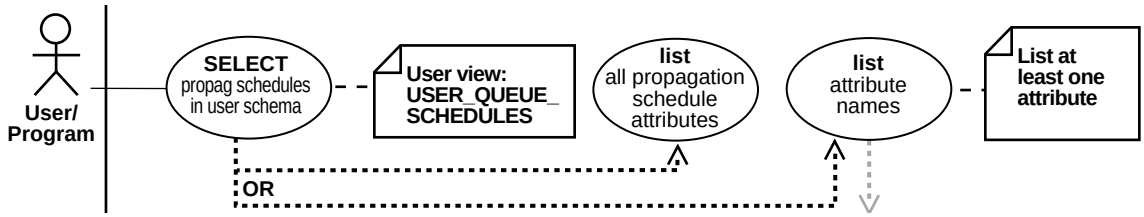
Use Case Diagrams

Graphic Element	Description
	In a use case diagram, each primary use case is instigated by an actor (stickman), which can be a human user, an application, or a subprogram. The actor is connected to the primary use case, which is depicted as an oval (bubble) enclosing the use case action. The totality of primary use cases is described in a use case model diagram.

Graphic Element	Description
	Primary use cases may require other operations to complete them. In this diagram fragment ■ **`specify`** `queue name` is one of the suboperations, or secondary use cases, needed to complete ■ **`ENQUEUE`** `a message` The downward lines from the primary use case lead to the other required operations (not shown).
	A secondary use case with a drop shadow expands into its own use case diagram, thus making it easier to: ■ Understand the logic of the operation ■ Continue a complex operation across multiple pages In this example ■ **`specify`** `message properties,` ■ **`specify`** `options` ■ **`add`** `payload` are all expanded in separate use case diagrams. In the online versions of these diagrams, these are clickable areas that link to the related operation.

Graphic Element	Description
	This diagram fragment shows the expanded use case diagram. While the standard diagram has the actor as the initiator, here the use case itself is the point of departure for the suboperation. In this example, the expanded view of ■ **`add`** `payload` represents a constituent operation of ■ **`ENQUEUE`** `a message`
	The a, b, c convention shows that there are three ways to create a table containing LOBs.
	This fragment shows one of the uses of a note box, here presenting one of the three ways to create a table containing LOBs.

Graphic Element

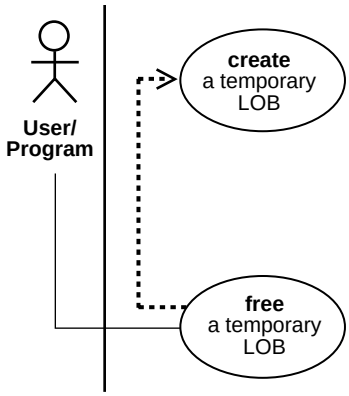


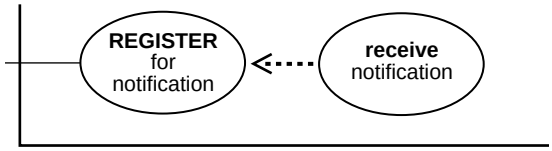
Description

This drawing shows other uses of note boxes:

- Note boxes can present an alternative name. In this case, the action **`SELECT`** `propagation schedules` in `the user schema` is represented by the view `USER_QUEUE_SCHEDULES.`
- Note boxes can qualify the use case action. In this case, the **`list`** `attribute names` action is qualified by the note that you must list at least one attribute if you do not list all the attributes of the propagation schedule.

Graphic Element





Description

The dotted arrow in the use case diagram indicates dependency. In this example

- **`free`** a `temporary` `LOB`

requires that you first

- **`create`** a `temporary` `LOB`

Put another way: you should not execute the **`free`** operation on a LOB that is not temporary.

The target of the arrow shows the operation that must be performed first.

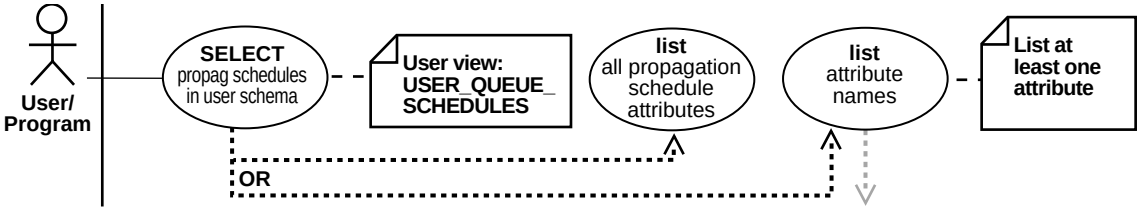
Use cases and their suboperations can be linked in complex relationships. In this example of a callback, you must first

- **`REGISTER`** `for` `notification`

in order to later

- **`receive`** a `notification`

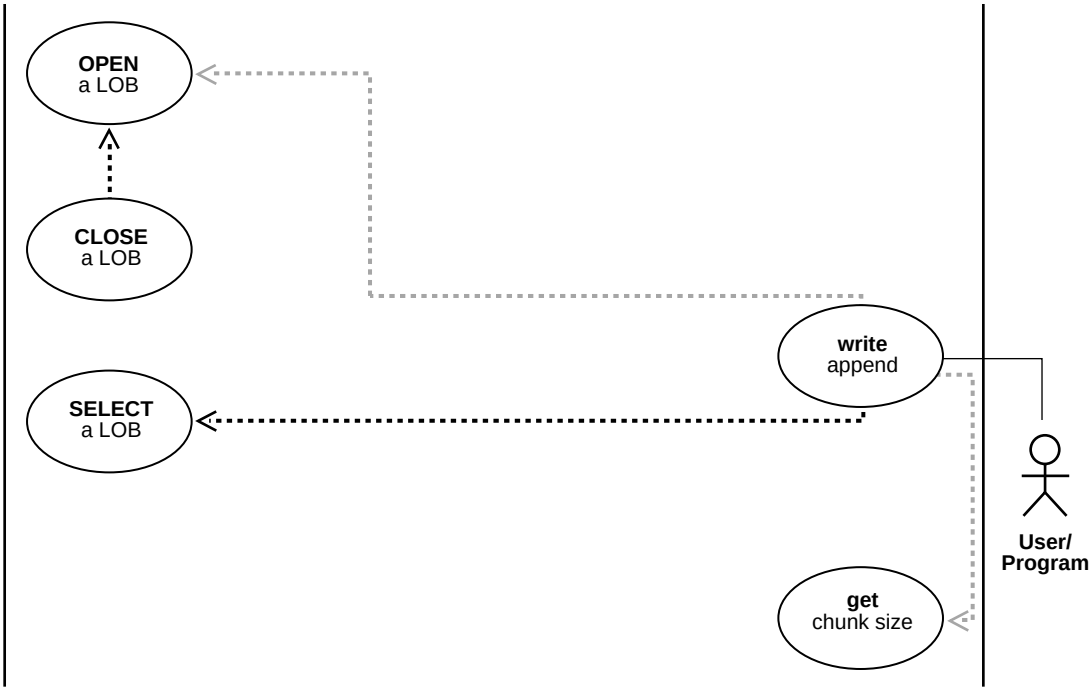
Graphic Element



Description

In this case the branching paths of an OR condition are shown. In invoking the view, you can choose to list all the attributes or view one or more attributes. The grayed arrow indicates that you can stipulate which attributes you want to view.

Graphic Element



Description

The black dashed line and arrow indicate that the targeted operation is required. The gray dashed line and arrow indicate that the targeted operation is optional. In this example, executing

- **`write`** `append`

on a `LOB` requires that you first

- **`SELECT`** `a LOB`

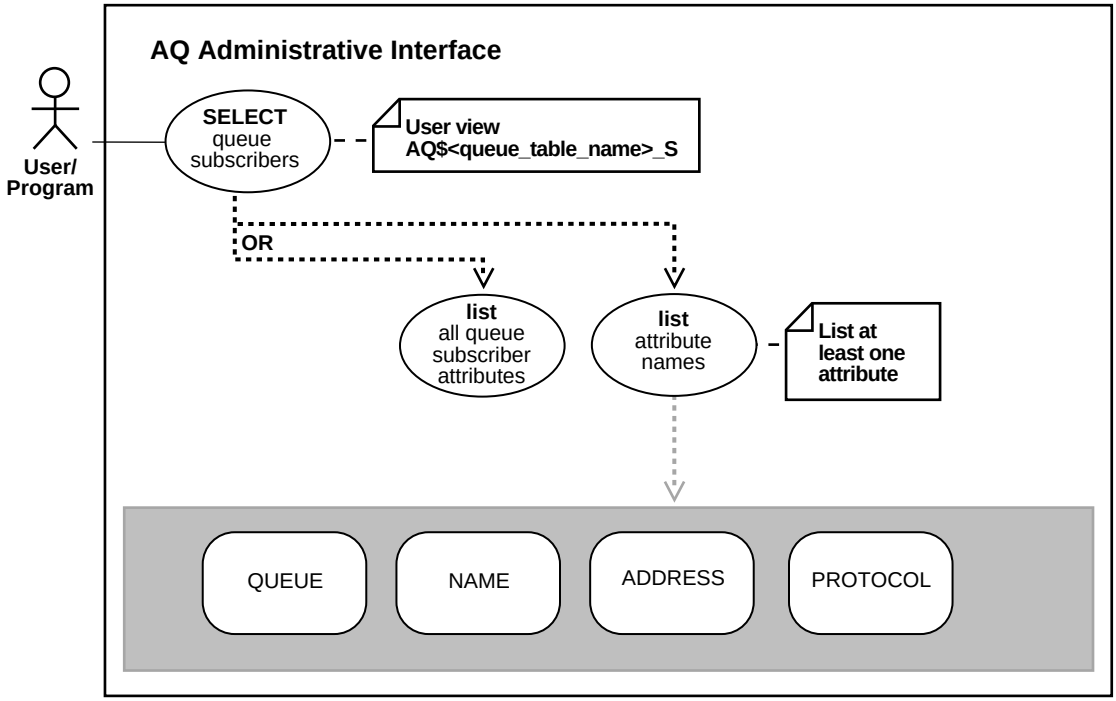
You may optionally choose to

- **`OPEN`** `a LOB` or **`get`** `chunk size`

Note that if you do **`OPEN`** `a LOB`, you must **`CLOSE`** `it.`

State Diagrams

Graphic Element



Description

The previous notes have discussed use case diagrams. Here we introduce the basic application of a state diagram, which we use to present the attributes of view. Attributes of a view have only two states—visible or invisible. We are not interested in showing the permutations of state, but in showing what you can make visible in invoking a view. Accordingly, we have extended the UML to join a partial state diagram onto a use case diagram to show the totality of attributes, and thereby all the substates of the view that you can see. We have demarcated the use case from the view state by coloring the background gray.

In this example, the view allows you to query queue subscribers. You can stipulate one attribute, some combination of the four attributes, or all four attributes.

Graphic Element	Description
Internal temporary LOBs (part 1 of 2) **continued on next page**	Use case model diagrams summarize all the use cases in a particular domain, such as `Internal temporary LOBs`. Where diagrams are too complex to fit on one page, we have divided the diagram into two parts. No sequence is implied in this division. This marker indicates that the diagram is continued.

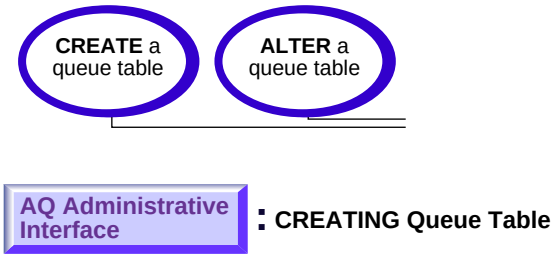
Links in Online Versions of this Document

The online (HTML and PDF) versions of these diagrams include active areas that have blue perimeters or look like buttons. You can use these links to traverse the following relationships:

- To move between Use Case Model Diagrams that encompass all the possible use cases for a given interface, and the Use Case Diagrams that detail the individual cases.
- To traverse different branches of a use case that can be implemented in more than one way. The branching Use Case Diagrams have titles such as "Three Ways to..." and buttons marked by "a", "b", "c".
- To access the sub-use cases that are entailed as part of a more primary use case while retaining context.
- To view details of the classes that underlie use cases accessible in Java.
- To view the class structure in which specific Java classes are located (see Appendix B, "Oracle JMS Interfaces, Classes and Exceptions").

The following examples illustrate these relationships.

Graphic Element

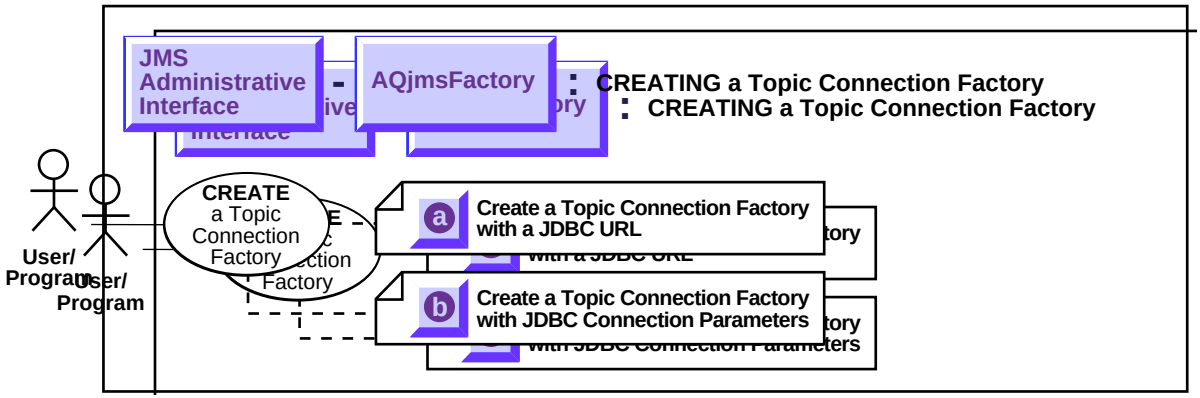


Description

Use Case Model Diagrams, which summarize all the use cases in a particular domain, have active areas that link to the individual use cases.

Buttons in the individual Use Case Diagrams lead back to the Use Case Model Diagram

Graphic Element



Description

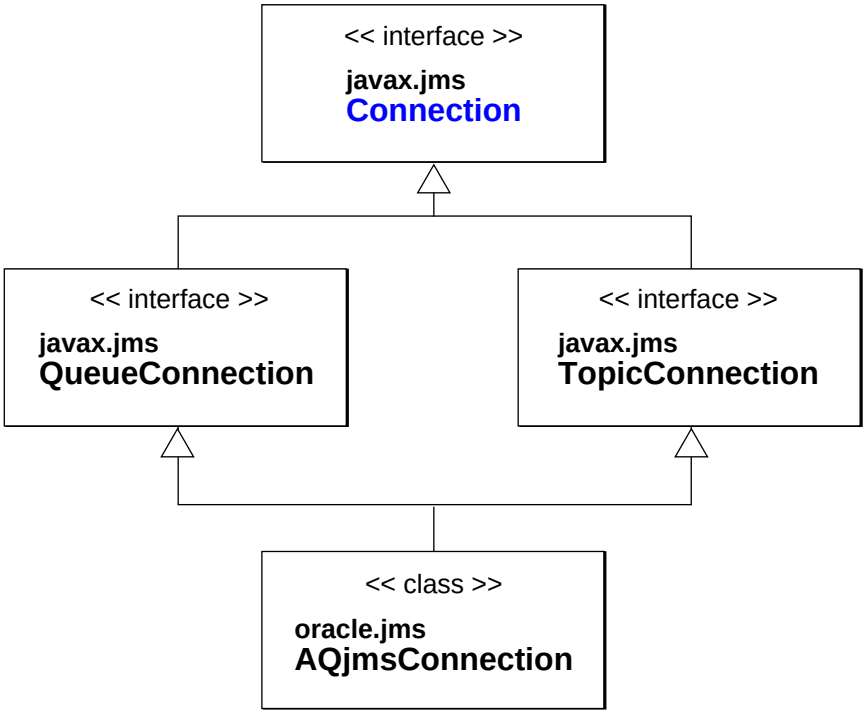
This Use Case Diagram combines a number of the elements:

- **JMS Administrative Interface** - this button leads back to the Use Case Model Diagram
- **AQjmsFactory** - this button leads to the class diagram, which contains the method by which the use case is implemented

  a. Create a Topic Connection Factory with JDBC URL is the "a" branch of the use case

  b. Create a Topic Connection Factory with JDBC Connection Parameters is the "b" branch of the use case

Graphic Element

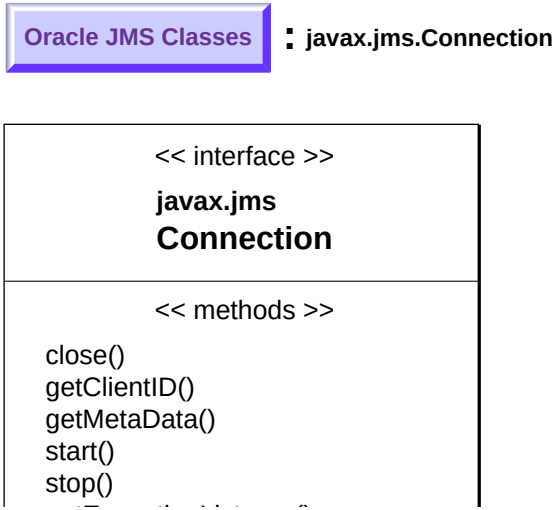


Description

This class diagram has links to the individual class diagrams that form its components. This reduced view of the classes shows:

- Whether classes, interfaces, and exceptions are entailed in the interrelationship by means of the <<>>, stereotype, such as **<<interface>>**
- The name of the package in which the class is found, such as **oracle.jms**
- The name of the class, such as **AQjmsConnection**

Graphic Element



Description

The expanded view of the class diagram:

- Links to the class structure diagram using the **Oracle JMS Classes** button, which describes its interrelationships. Although many classes share this button, each class is linked to the part of the structure specific to it.
- Contains the names of the attributes (fields) if they exist and are exposed (there are none in this case)
- Contains the name of the methods that comprise the public interface to the class

Index

A

access control. *See* system-level access control, 4-4, 8-2
adding a subscriber, 9-61
administrative interface, 4-4, 9-1
  basic operations, 13-2
  JMS, 13-4
  use cases, 13-2
  view, 10-2
  views, 10-1
Adt message, 12-27
Advanced Queuing
  operations over the Internet, 17-2
agent, 1-22
  identifying, 2-3, 2-4
agent. *See* AQ agent, 9-97
AQ agent
  altering, 9-97
  creating, 9-94
  dropping, 9-99
  registering, 17-50
AQ servlet, 17-2
AQ XML
  document, 17-6
  requests, using SMTP, 17-53
  schema, 17-33, 17-35
  servlet, 17-47, 17-52
AQ XML servlet
  registering for notifications, 8-106
AQ_TM_PROCESSES, 2-10
AQjmsQueueConnectionFactory, B-78
AQXmlPublish method, 17-7
AQXmlReceive method, 17-21
AQXmlSend method, 17-7
asynchronous notification, 1-17, 8-97
asynchronously receiving message, 12-74

B

BooksOnLine sample application, 8-1
  using JMS, 12-2
bytes message, 12-25

C

C. *See* Oracle Call Interface (OCI)
Class - AQjmsQueueConnectionFactory, B-78
Class - oracle.AQ.AQQueueTableProperty, B-93
Class - oracle.jms.AQjmsAdtMessage, B-61
Class - oracle.jms.AQjmsAgent, B-62
Class - oracle.jms.AQjmsBytesMessage, B-63
Class - oracle.jms.AQjmsConnection, B-64
Class - oracle.jms.AQjmsConstants, B-66
Class - oracle.jms.AQjmsDestination, B-68
Class - oracle.jms.AQjmsDestinationProperty, B-70
Class - oracle.jms.AQjmsFactory, B-71
Class - oracle.jms.AQjmsMapMessage, B-72
Class - oracle.jms.AQjmsObjectMessage, B-74
Class - oracle.jms.AQjmsOracleDebug, B-75
Class - oracle.jms.AQjmsProducer, B-76
Class - oracle.jms.AQjmsQueueBrowser, B-77
Class - oracle.jms.AQjmsSession, B-79
Class - oracle.jms.AQjmsStreamMessage, B-82
Class - oracle.jms.AQjmsTextMessage, B-83
Class - oracle.jms.AQjmsTopicConnectionFactory, B-84

Class oracle.jms.AQjmsException, B-85
classes, B-2
Classes, JMS, B-6
commit response, 17-32
commit transaction, 17-26
commonly asked questions. *See* frequently asked questions, 6-1
compositing, 7-15
connection factory
  registering queue/topic, 13-5
  unregistering in LDAP through database, 13-16
  unregistering in LDAP through LDAP, 13-18
consumers, 7-3
correlation identifier, 1-15
creating
  queue, 9-22
  queue tables and queues, examples, A-4
creation of prioritized message queue table and queue, A-5
creation of queue table and queue of object type, A-4
creation of queue table and queue of RAW type, A-5

D

database
  design and modeling, 7-1
  tuning, 5-2
database access
  enabling, 9-101
database session, 17-51
DBA_ATTRIBUTE_TRANSFORMATIONS, 10-41
DBA_QUEUE_TABLES, 10-5, 10-7, 10-23
DBA_QUEUES, 10-8
DBA_TRANSFORMATIONS, 10-40
DBMS_AQADM package, 4-2
DBMS_AQADM.DROP_QUEUE, 9-20
delay, 2-9
  time specification, 12-57
delay interval
  retry with, 8-77
  time specification, 8-46
dequeue
  client request for, 17-21
dequeue mode, 2-9
dequeue of messages after preview, A-33
dequeue request
  server response, 17-29
dequeuing, 11-47
  features, 8-58
  message navigation, 8-66
  methods, 8-59
  modes, 1-18, 8-70
  multiple-consumer dequeuing of one message, 7-6
  navigation of messages, 1-18
  same message, multiple-consumer, 7-6
  using HTTP, 8-107
destination-level access control, 12-15
disabling
  propagation schedule, 9-91
dropping
  queue table, 9-19
dropping AQ objects, A-66
durable subscriber, 12-43

E

email server, 17-2
  setup, 17-54
enqueue
  client request for, 17-7
  server response, 17-27
enqueue and dequeue of messages
  by Correlation and Message Id Using Pro*C/C++, A-38
  by priority, A-14, A-16, A-18
  examples, A-11
  of object type, A-11
  of RAW type, A-14, A-16, A-18
  of RAW type using Pro*C/C++, A-22, A-25
  to/from multiconsumer queues, A-44, A-47
  with time delay and expiration, A-37
enqueuing, 11-5, 11-13
  features, 8-36
  specify message properties, 11-10
  specify options, 11-7
enqueuing, priority and ordering of messages, 1-16
Enterprise Manager, 1-7

enumerated constants
administrative interface, 2-8
operational interface, 2-9
error messages, D-1
examples
AQ operations, A-1
exception handling, 1-19, 8-81, 12-78
Exception Handling During Propagation, 12-93, 12-94
exception handling during propagation, 12-90, 12-91
Exception javax.jms.InvalidDestinationException, B-47
Exception javax.jms.InvalidSelectorException, B-48
Exception javax.jms.JMSException, B-49
Exception javax.jms.MesageNotWriteableException, B-53
Exception javax.jms.MessageEOFException, B-50
Exception javax.jms.MessageFormatException, B-51
Exception javax.jms.MessageNotReadableException, B-52
Exception oracle.jms.AQjmsInvalidDestinationException, B-86
Exception oracle.jms.AQjmsInvalidSelectorException, B-87
Exception oracle.jms.AQjmsMessageEOFException, B-88
Exception oracle.jms.AQjmsMessageFormatException, B-89
Exception oracle.jms.AQjmsMessageNotReadableException, B-90
Exception oracle.jms.AQjmsMesssageNotWriteableException, B-91
exception queue, 1-22
exceptions, B-2
expiration, 2-9
time specification, 8-49
exporting
incremental, 4-6
queue table data, 4-5

F

fanning-out of messages, 7-15
FAQs. *See* frequently asked questions, 6-1
features, new, lvii
frequently asked questions, 6-1
general questions, 6-1
installation questions, 6-10
Internet access questions, 6-7
JMS questions, 6-6
Oracle Internet Directory, 6-9
performance questions, 6-10
transformation questions, 6-9
funneling-in of messages. *See* compositing, 7-15

G

global agents, 6-9
global events, 6-9
global queues, 6-9
granting
system privilege, 9-50
grouping
message, 12-60

H

HTTP, 1-12, 17-2, 17-6, 17-49, 17-50, 17-58
accessing AQ XML servlet, 17-56
AQ operations over, 17-2
headers, 17-5
propagation, 17-60
response, 17-6
HTTPS
propagation, 17-60

I

IDAP, 17-33
body, 17-4
envelope, 17-4
headers, 17-4

message structure, 17-4
method invocation, 17-5
transmitted over Internet, 17-1
IDAP schema, 17-34
IDAP. *See Internet Data Access Presentation*, 1-13, 17-3
INIT.ORA parameter, 2-9
Interface - javax.jms.BytesMessage, B-18
Interface - javax.jms.Connection, B-19
Interface - javax.jms.ConnectionFactory, B-20
Interface - javax.jms.ConnectionMetaData, B-21
Interface - javax.jms.DeliveryMode, B-22
interface - javax.jms.Destination, B-23
Interface - javax.jms.MapMessage, B-24
Interface - javax.jms.Message, B-25
Interface - javax.jms.MessageConsumer, B-27
Interface - javax.jms.MessageListener, B-28
Interface - javax.jms.MessageProducer, B-29
Interface - javax.jms.ObjectMessage, B-30
Interface - javax.jms.Queue, B-31
Interface - javax.jms.QueueBrowser, B-32
Interface - javax.jms.QueueConnection, B-33
Interface - javax.jms.QueueConnectionFactory, B-34
Interface - javax.jms.QueueReceiver, B-35
Interface - javax.jms.QueueSender, B-36
Interface - javax.jms.QueueSession, B-37
Interface - javax.jms.Session, B-38
Interface - javax.jms.StreamMessage, B-39
Interface - javax.jms.TextMessage, B-40
Interface - javax.jms.Topic, B-41
Interface - javax.jms.TopicSession, B-45
Interface - javax.jms.TopicSubscriber, B-46
Interface - oracle.AQ.AQQueueTable, B-92
Interface - oracle.jms.AdtMessage, B-54
Interface - oracle.jms.AQjmsConnectionMetadata, B-65
Interface - oracle.jms.AQjmsConsumer, B-67
Interface - oracle.jms.AQjmsQueueReceiver, B-55
Interface - oracle.jms.AQjmsQueueSender, B-56
Interface - oracle.jms.AQjmsTopicPublisher, B-57
Interface - oracle.jms.AQjmsTopicReceiver, B-60
Interface - oracle.jms.AQjmsTopicSubscriber, B-59
Interface - oracle.jms.TopicReceiver, B-58
interfaces, classes, and exceptions, B-2
Internet
access, 8-36
Advanced Queuing operations, 17-2
Advanced Queuing operations over, 17-1
AQ operations over, li, 1-2, 1-12
Internet Data Access Presentation (IDAP), 1-13, 17-3

J

Java API, 2-10
Java. *See* JDBC
javax.jms.BytesMessage, B-18
javax.jms.Connection, B-19
javax.jms.ConnectionFactory, B-20
javax.jms.ConnectionMetaData, B-21
javax.jms.DeliveryMode, B-22
javax.jms.Destination, B-23
javax.jms.InvalidDestinationException, B-47
javax.jms.InvalidSelectorException, B-48
javax.jms.JMSException, B-49
javax.jms.MapMessage, B-24
javax.jms.MesageNotWriteableException, B-53
javax.jms.Message, B-25
javax.jms.MessageConsumer, B-27
javax.jms.MessageEOFException, B-50
javax.jms.MessageFormatException, B-51
javax.jms.MessageListener, B-28
javax.jms.MessageNotReadableException, B-52
javax.jms.MessageProducer, B-29
javax.jms.ObjectMessage, B-30
javax.jms.Queue, B-31
javax.jms.QueueBrowser, B-32
javax.jms.QueueConnection, B-33
javax.jms.QueueConnectionFactory, B-34
javax.jms.QueueReceiver, B-35
javax.jms.QueueSender, B-36
javax.jms.QueueSession, B-37
javax.jms.Session, B-38
javax.jms.StreamMessage, B-39
javax.jms.TextMessage, B-40
javax.jms.Topic, B-41
javax.jms.TopicConnection, B-42
javax.jms.TopicSession, B-45
javax.jms.TopicSubscriber, B-46

JDBC, 3-6
connection parameters, registering through LDAP, 13-11
connection parameters, registering through the database, 13-6
JDBC URL
registering through LDAP, 13-14
registering through the database, 13-8
JMS
administrative interface, 13-4
examples payload, 12-29
JMS classes, B-2
JMS exceptions, B-2
JMS Extension, 3-8
JMS interfaces, B-2
JMS Type queues/topics, 17-10
JMS types, 17-10
JMSClasses, B-6
JOB_QUEUE_PROCESSES parameter, 2-10

L

LDAP
registering, 13-10, 13-14
unregistering, 13-16, 13-18
LDAP server, 17-2
with an AQ XML Servlet, 17-52
listen capability, 8-90

M

map message, 12-26
message
error, AQ XML servlet, D-1
error, JMS, D-1
fanning-out, 7-15
grouping, 8-52
history, 8-28
navigation in dequeue, 8-66
ordering, 8-39, 12-54
priority and ordering, 8-39, 12-54
propagation, 7-15
recipient, 7-6
message enqueuing, 11-5
message format transformation, 1-7
message grouping, 1-16, 12-60
message history and retention, 12-16
message navigation in receive, 12-67
message payloads, 17-9
message producer features, 12-53, 12-91
message_grouping, 2-9
messages
producers and consumers, 1-22
messages, definition, 1-21
modeling
queue entities, 7-2
modeling and design, 7-1
multiple recipients, 1-18

N

navigation, 2-9
new features, lvii
nonpersistent queue, 1-10, 1-22, 6-3
creating, 9-28
normal queues. *See* user queue, 1-22
notification, 17-32
asynchronous, 8-97

O

object message, 12-26
object types, 4-3, 4-17
object_name, 2-2
OO4O. *See* Oracle Objects for OLE (OO4O)
operational interface
basic operations, 11-1
use cases, 11-2
optimization
arrival wait, 8-75
optimization of waiting for messages, 1-18
Oracle Extension, 3-8
Oracle Internet Directory, 6-9, 17-2
Oracle JMSClasses, B-6
Oracle object (ADT) type queues, 17-9
Oracle Real Application Clusters, 1-11, 8-31, 12-17
oracle.AQ.AQQueueTable, B-92
oracle.AQ.AQQueueTableProperty, B-93
oracle.jms.AdtMessage, B-54
oracle.jms.AQjmsAdtMessage, B-61

oracle.jms.AQjmsAgent, B-62
oracle.jms.AQjmsBytesMessage, B-63
oracle.jms.AQjmsConnection, B-64
oracle.jms.AQjmsConstants, B-66
oracle.jms.AQjmsConsumer, B-67
oracle.jms.AQjmsDestination, B-68
oracle.jms.AQjmsDestinationProperty, B-70
oracle.jms.AQjmsException, B-85
oracle.jms.AQjmsFactory, B-71
oracle.jms.AQjmsInvalidDestinationException, B-86
oracle.jms.AQjmsInvalidSelectorException, B-87
oracle.jms.AQjmsMapMessage, B-72
oracle.jms.AQjmsMessageEOFException, B-88
oracle.jms.AQjmsMessageFormatException, B-89
oracle.jms.AQjmsMessageNotReadableException, B-90
oracle.jms.AQjmsMesssageNotWriteableException, B-91
oracle.jms.AQjmsObjectMessage, B-74
oracle.jms.AQjmsOracleDebug, B-75
oracle.jms.AQjmsProducer, B-76
oracle.jms.AQjmsQueueBrowser, B-77
oracle.jms.AQjmsQueueReceiver, B-55
oracle.jms.AQjmsQueueSender, B-56
oracle.jms.AQjmsSession, B-79
oracle.jms.AQjmsStreamMessage, B-82
oracle.jms.AQjmsTextMessage, B-83
oracle.jms.AQjmsTopicBrowser, B-96
oracle.jms.AQjmsTopicConnectionFactory, B-84
oracle.jms.AQjmsTopicPublisher, B-57
oracle.jms.AQjmsTopicReceiver, B-60
oracle.jms.AQjmsTopicSubscriber, B-59
oracle.jms.TopicBrowser, B-95
oracle.jms.TopicReceiver, B-58

P

payload, 17-9
  structured, 8-10
performance, 5-2
persistent queue, 1-22
ping period for Exception Listener, 16-104, 16-105
PL/SQL, 3-2
priority and ordering of messages, 12-54
privileges, 4-4
  revoking, A-67
producers, 7-3
programmatic environments, 2-7, 3-2
propagation, 1-16, 2-10, 8-107, 12-81, 17-60
  exception handling, 12-90, 12-91, 12-93
  exception handling during, 12-90
  failures, 4-12
  features, 8-107
  issues, 4-11
  LOB attributes, 8-111
  message, 4-5, 7-15
  messages with LOB attributes, 8-111
  schedule, 12-86
  scheduling, 1-20, 8-108, 8-114
  using HTTP, 8-119
propagation schedule, 12-88
  altering, 9-84
  disabling, 9-91
  enabling, 9-88
  selecting, 10-10
  selecting all, 10-10
  selecting in user schema, 10-26
Propagation, Exception Handling During, 12-94
publish-subscribe, 7-12, 8-29
  topic, 12-42

Q

QMN. *See* queue monitor (QMN), 1-24, 2-10
queue, 1-22
  altering, 9-31
  creating, 9-22
  creating, example, A-4
  dropping, 9-34
  nonpersistent, 1-10, 6-3, 9-28
  point-to-point, 12-36
  selecting all, 10-8
  selecting in user schema, 10-24
  selecting, in user schema, 10-24
  selecting, user has any privilege, 10-14
  selecting, user has queue privilege, 10-16
  staring, 9-44
  starting, 9-44
  stopping, 9-47

subscriber rules, 10-32
subscriber, selecting, 10-30
subscribers, 7-6
subscribers, selecting, 10-30
queue monitor, 1-24
queue monitor (QMN), 2-10
queue privilege
granting, 9-55
revoking, 9-58
queue propagation
scheduling, 9-74
unscheduling, 9-78
queue subscribers
selecting, rules, 10-32
queue table, 1-22
altering, 9-16
creating, 9-6
creating prioritized message, 9-25
creating, example, 9-10, 9-11, 9-24, 9-25
creating, example, XMLType attributes, 9-10
dropping, 9-19
messages, selecting, 10-18
selecting all, 10-4, 10-18
selecting messages, 10-18
selecting user tables, 10-6
queue table data
exporting, 4-5
queue tables
creating, example, A-4
selecting all in user schema, 10-22
queue type
verifying, 9-81
queue_type, 2-9
queue-level access control, 1-10, 8-16
queue/topic
connection factory, registering, 13-5
connection factory, registering through LDAP, 13-10
connection factory, unregistering in LDAP through the database, 13-16
connection factory, unregistering in LDAP through the LDAP, 13-18
queuing
basic, 7-3
basic, one producer, one consumer, 7-3

R

RAW queues, 17-9
Real Application Clusters. *See* Oracle Real Application Clusters, 1-11
receiving messages, 12-64
recipient, 1-23, 7-6
list, 2-4, 8-37, 12-48
local and remote, 1-18, 8-64
multiple, 8-63
register request
server response, 17-32
registering
AQ Agent, 17-50
JDBC connection parameters through LDAP, 13-11
JDBC URL through LDAP, 13-14
queue/topic connection factory, 13-5
queue/topic connection factory through LDAP, 13-10
through the database, JDBC connection parameters, 13-6
through the database, JDBC URL, 13-8
registration
client request for, 17-7
to a queue, 2-5
registration for notification vs. listener, 6-3
retention, 2-9
retention and message history, 1-9, 8-28, 12-16
retries with delays, 1-19
retry
delay interval, 8-77
revoking roles and privileges, A-67
role
revoking, A-67
user, 4-2
rollback a transaction, 17-27
rollback response, 17-32
rule, 1-24
selecting subscriber, 10-32
rule-based subscriber, 1-17
example, 9-64
rule-based subscription, 8-86

S

scheduling
  propagation, 1-20, 12-86
schema
  AQ XML, 17-33, 17-35
  IDAP, 17-34
security, 4-2, 4-3
sender identification, 1-17
sequence_deviation, 2-9
servlet
  AQ XML, 17-47, 17-52
queue table
  creating, 9-15
SMTP, 1-12, 17-53
  operations over the Internet, 17-2
SQL access, 1-8
state parameter, 2-9
statistics views, 8-35
statistics views support, 12-19
stream message, 12-24
structured payload, 1-9, 8-10
structured payload / message types, 12-19
subscriber, 2-4
  adding, 9-61
  altering, 9-67
  durable, 12-43
  removing, 9-71
  rule-based, 1-17, 9-64
  selecting, 10-30
subscription, 8-37
  anonymous, 2-5
  rule-based, 8-86
subscription and recipient list, 1-15
subscription and recipient lists, 1-15
system privilege
  granting, 9-50
  revoking, 9-53
system-level access control, 8-2, 12-14

T

text message, 12-26
time specification, 1-17
  delay, 8-46, 12-57
  expiration, 8-49, 12-58
topic publisher, 12-46
tracking and event journals, 1-10
transaction protection, 1-19
transformation. *See* message format
    transformation, 1-7, 1-24
tuning. *See* database tuning, 5-2
type_name, 2-2
types
  object, 4-3, 4-17

U

Unified Modeling Language (UML), E-i
unregistering
  queue / topic connection factory in LDAP, 13-16, 13-18
use case diagrams, E-i
user authentication, 17-49
user authorization, 17-50
user queue, 1-22
user role, 4-2
USER_ATTRIBUTE_TRANSFORMATIONS, 10-40
USER_TRANSFORMATIONS, 10-39

V

view, 10-1, 10-2
  attributes, 10-1
views
  statistics, 8-35
visibility, 2-9
Visual Basic. *See* Oracle Objects for OLE(OO4O)

W

wait, 2-9
waiting
  for message arrival, 8-75
Web server
  setup, 17-54

X

XML, 17-1

components, 17-10
document, 17-6
schema, 17-33, 17-35
servlet, 17-47, 17-52
servlet, HTTP, 17-56