Preface

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

This Preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for developers of applications that use Oracle Database Advanced Queuing.

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Development Guide*
- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Advanced Queuing Java API Reference*
- *Oracle Database Transactional Event Queues Java API Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle XML DB Developer's Guide*

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

Conventions

The following text conventions are used in this guide:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database Advanced Queuing User's Guide

This preface contains:

Changes in Oracle Database Advanced Queuing Release 23ai

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database Release 23ai.

New Features

The following features are new in this release:

- Transactional Event Queue (TxEventQ) Propagation

  Transactional Event Queue (TxEventQ) Propagation enables events at source to be sent to destination queues reliably honoring the ordering semantics, with JMS session level ordering.

  TxEventQ gets created with set of “Event Streams” which is configurable, and it allow the applications to parallelize the publish activity using multiple subscribers. “Event Streams” are viewed as a storage mechanism for TxEventQ. Applications publish events in TxEventQs under 'Event Streams'

  Facility of Propagation leads to multiple benefits, the ability to process published messages remotely, the ability to backup critical data at remote locations and any others.

  See Propagation Features for more information

- Enhancement of Kafka Implementation for Transactional Event Queues

  Enhancement to the client library now allows the Kafka applications to atomically produce and or consume messages from multiple queues of Transactional Event Queues (TxEventQ). Support for Kafka Rebalancing which facilitates distribution of topic partitions among available Kafka consumers is also now available.

  See Kafka APIs for Oracle Transactional Event Queues for more information

- Oracle Database Advanced Queuing (AQ) to Transactional Event Queue (TxEventQ) online Migration Tool

AQ to TxEventQ online Migration Tool enables online and automated migration of a AQ deployment to next generation messaging product TxEventQ. The migration interface works without blocking queuing operations. It creates an internal Oracle managed TxEventQ for the AQ that is being migrated. Migration framework will internally drain (or dequeue) messages from AQ and enqueue them into the internal TxEventQ. At an appropriate time, the TxEventQ will assume the name of the AQ, and the AQ will be dropped.

The online migration tool first help assess the feature mapping between AQ and TxEventQ to flag any mismatches. Once this phase passes an online migration is attempted, and a choice is given after some time to continue with the migration or cancel it.

See Migrating from AQ to TxEventQ and DBMS_AQMIGTOOL for more information

- Python, Node.js and REST drivers for Transactional Event Queues (TxEventQ)

  Starting with Oracle Database 23ai, Transactional Event Queues (TxEventQ) also have support for Python, Node.js and REST drivers.

  See Polyglot Programming with Transactional Event Queues for more information.

- Prometheus/Grafana for Oracle

  Oracle Database 23ai introduces Prometheus/Grafana for Oracle (PGO), which provides database metrics for developers running in a Kubernetes/Docker (K8S) environment. Database metrics are stored in Prometheus, a time-series database and metrics tailored for developers are displayed using Grafana dashboards.

  The metrics give a picture of database performance colored by various applications and the modules it comprises.

  See Monitoring Transactional Event Queues for more information.

Deprecated Features

The following features are deprecated in this release:

- Oracle Messaging Gateway
  - `DBMS_MGWADM`

See Oracle Messaging Gateway for more information.

Changes in Oracle Database Advanced Queuing Release 21c

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database Release 21c.

- New Features
- Deprecated Features

New Features

The following features are new in this release:

- Advanced Queuing: Kafka Java Client for Transactional Event Queues

  Kafka Java Client for Transactional Event Queues (TxEventQ) enables Kafka application compatibility with Oracle Database. This provides easy migration of Kafka applications to TxEventQ.

Starting from Oracle Database 21c, Kafka Java APIs can connect to Oracle Database server and use Transactional Event Queues (TxEventQ) as a messaging platform. Developers can migrate an existing Java application that uses Kafka to the Oracle Database. A client side library allows Kafka applications to connect to Oracle Database instead of Kafka cluster and use TxEventQ messaging platform transparently. Kafka interoperability is supported by configuring Kafka JMS Connectors to move messages between the two messaging systems.

See Kafka APIs for Oracle Transactional Event Queues for more information.

- PL/SQL Enqueue and Dequeue Support for JMS Payload in Transactional Event Queues

  PL/SQL APIs perform enqueue and dequeue operations for Java Message Service (JMS) payload in Transactional Event Queues. Similarly, the PL/SQL Array APIs are exposed to Transactional Event Queues JMS users. Since JMS support of heterogeneous messages, dequeue gets one of the five JMS message types back, but cannot predict what is the type of the next message received. Therefore, it can run into application errors with PL/SQL complaining about type mismatch. Oracle suggests that the application always dequeue from Transactional Event Queues using the generic type AQ$_JMS_MESSAGE. PL/SQL administration is also supported.

  See Transactional Event Queues and Enqueuing / Dequeuing Messages for more information

- PL/SQL Enqueue and Dequeue Support for non-JMS Payload in Transactional Event Queues

  To improve throughput and reduce overhead and latency, enqueues and dequeues are optimized to use the message cache, the rules engine, and background processing when possible.

  See Transactional Event Queues and Enqueuing / Dequeuing Messages for more information

- Transactional Event Queues for Performance and Scalability

  Oracle Database 21c introduces Transactional Event Queues (TxEventQ), which are partitioned message queues that combine the best of messaging, streaming, direct messages, and publish/subscribe. TxEventQs have their Queue tables partitioned into multiple Event Streams, which are distributed across multiple RAC nodes for high throughput messaging and streaming of events.

  See Oracle Transactional Event Queues and Advanced Queuing Performance and Scalability for more information

- Simplified Metadata and Schema in Transactional Event Queues

  Oracle Database 21c introduces Transactional Event Queues (TxEventQ), which are partitioned message queues that combine the best of messaging, streaming, direct messages, and publish/subscribe. TxEventQ operates at scale on the Oracle Database. TxEventQ provides transactional event streaming, and runs in the database in a scale of 10s to 100s of billions of messages per day on 2-node to 8-node Oracle RAC databases, both on-premise and on the cloud. TxEventQ has Kafka client compatibility, which means, Kafka producer and consumer can use TxEventQ in the Oracle Database instead of a Kafka broker.

- Support for Message Retention and Seekable Subscribers in Transactional Event Queues

  A user can specify a time for which the message can be retained, even after the subscribers have consumed the message. The retention time is specified in seconds by user. It can vary from `0` to `INIFINITE`. Without retention, when a message is dequeued by all subscribers in sharded queues, the message is permanently removed from the queuing system.

The typical way a subscriber can consume from the queue is through a dequeue operation, which now supports seeking an offset into the message queue. Many queueing applications require subscriber to consume messages that were enqueued prior to its creation. Using this seek capability for subscribers, applications can reposition dequeue point to messages that were enqueued prior to the subscriber creation. This offers flexibility for applications to make the communication truly asynchronous between the producer and the consumer of the message.

See Transactional Event Queues and Message Retention and Transactional Event Queues and Seekable Subscribers for more information.

- Native JSON data type support in AQ and Transactional Event Queues

  Starting from Oracle Database 21c, Advanced Queuing also supports JSON datatype. Along with RAW/ADT payload type we can also specify JSON payload type during queue table creation of classic queue and during queue creation of sharded queue. Users can also specify embedded element JSON with simple adt. `DBMS_AQ`, `DBMS_AQADM` procedures like `create_queue_table`/`create_sharded_queue`/`enqueue`/`dequeue` and procedures for OGG/DG replication also accepts JSON datatype.

Deprecated Features

The following features are deprecated in this release:

- Sharded Queue APIs
  - `CREATE_SHARDED_QUEUE`
  - `DROP_SHARDED_QUEUE`
  - `ALTER_SHARDED_QUEUE`
  - `ISSHARDEDQUEUE`
  - `VERIFY_SHARDED_QUEUE`
- Sharded Queue Views
  - `ALL_QUEUE_SHARDS`
  - `DBA_QUEUE_SHARDS`
  - `USER_QUEUE_SHARDS`
  - `GV$AQ_CACHED_SUBSHARDS`
  - `GV$AQ_CROSS_INSTANCE_JOBS`
  - `GV$AQ_DEQUEUE_SESSIONS`
  - `GV$AQ_INACTIVE_SUBSHARDS`
  - `GV$AQ_MESSAGE_CACHE`
  - `GV$AQ_MESSAGE_CACHE_ADVICE`
  - `GV$AQ_MESSAGE_CACHE_STAT`
  - `GV$AQ_NONDUR_SUBSCRIBER_LWM`
  - `GV$AQ_REMOTE_DEQUEUE_AFFINITY`
  - `GV$AQ_SHARDED_SUBSCRIBER_STAT`
  - `GV$AQ_SUBSCRIBER_LOAD`
  - `GV$AQ_UNCACHED_SUBSHARDS`

  - `GV$AQ_NONDUR_SUBSCRIBER`
  - `GV$AQ_PARTITION_STATS`
  - `GV$AQ_MESSAGE_CACHE_STAT`

See Deprecation of Sharded Queues for more information.

Changes in Oracle Database Advanced Queuing Release 19c, Version 19.2

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database Release 19c, Version 19.2

New Features

The following feature is new in this release:

- Enhanced Key-based Messaging

  AQ sharded queues perform substantially better when compared to AQ classic (non-sharded) queues. This is achieved by sharding the queue, where each shard is owned by a specific instance of the database. By default, sharding is completely transparent to the user when it comes to enqueue and dequeue operations. AQ internally puts the message in the appropriate shard to get maximum performance and session level ordering as required by the JMS specification. Session level ordering ensures that no two messages will be dequeued in the reverse order of their enqueue order if both the messages are enqueued by the same session and have the same priority and delivery mode.

  In some cases, user applications want to control the sharding. The user application can choose the shard where a message is enqueued. The users can decide the way they plan to shard their messages in the sharded queue to support the application logic as needed. The performance and ordering benefits of AQ sharded queues are still maintained even if the sharding is under control of the user. Applications can control the following:

  - The number of shards of the queue
  - Key based enqueues: The enqueue session can choose the shard of the queue where the message will be enqueued by providing a key with the message at the time of enqueue. AQ server ensures that all the messages of a key are enqueued in the same shard. A shard can have messages of different keys.
  - Sticky dequeues: A shard can have only one active dequeue session for a single-consumer queue or JMS Queue. Similarly, a shard can have only one dequeue session per subscriber for a multi-consumer queue or JMS Topic. That dequeue session will stick to that shard of the queue for the session’s lifetime. Such functionality is also available for JMS listeners.

> **See Also:**
>
> User Event Streaming

Desupported Features

The following feature is desupported in this release:

- Desupport of Oracle Streams

  Starting in Oracle Database 19c, the Oracle Streams feature is desupported. Use Oracle GoldenGate to replace all replication features of Oracle Streams.

Changes in Oracle Database Advanced Queuing 12*c* Release 2 (12.2.)

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database 12*c* Release 2 (12.2).

New Features

The following features are new in this release:

- PL/SQL enqueue and dequeue support for JMS and non-JMS (ADT or RAW) payload in Sharded Queues

  Oracle Database 12*c* Release 2 (12.2) extends and supports PL/SQL APIs to perform enqueue and dequeue operations for JMS, ADT, and RAW payload in sharded queues. The PL/SQL Array APIs also support sharded queues. Many existing non-JMS applications can now use sharded queues with little or no change.

  Starting from Oracle Database 12*c* Release 2 (12.2), JMS customers using sharded queues can make use of PL/SQL notification to register a PL/SQL procedure that gets automatically invoked by AQ Server on successful enqueue. PL/SQL notification can eliminate the need for clients to poll the queue for messages because messages can be automatically dequeued and processed at the server.

  See Managing Sharded Queues for more information.

- Sharded Queue Diagnosability and Manageability

  Starting from 12*c* Release 2 (12.2), AQ sharded queues provides an advisor, views, and automated management for its message cache to optimize `STREAMS_POOL` memory allocation and throughput.

  See Transactional Event Queues Tuning for more information.

- Longer Identifiers for Oracle Database Advanced Queuing

  Starting from 12*c* Release 2 (12.2), the maximum length of AQ queue names has been increased to 122 bytes. The maximum length of subscriber and recipient names is increased to 128 characters. For the AQ Rules Engine, the maximum length of rule names and rule set names is now 128 bytes.

Changes in Oracle Database Advanced Queuing 12*c* Release 1 (12.1.0.2)

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database 12*c* Release 1 (12.1.0.2).

New Features

The following feature is new in this release:

- JMS Streaming

In Oracle Database 12*c* Release 1 (12.1.0.2), Advanced Queuing introduces JMS Streaming with enqueue and dequeue for sharded queues through `AQjmsBytesMessage` and `AQjmsStreamMessage`, for the applications interested in sending and receiving large message data or payload.

See "JMS Streaming" for more information.

Changes in Oracle Database Advanced Queuing 12*c* Release 1 (12.1)

The following are changes in *Oracle Database Advanced Queuing User's Guide* for Oracle Database 12*c* Release 1 (12.1).

New Features

The following features are new in this release:

- JMS Sharded Queues

  In Oracle Database 12*c* Release 1 (12.1), Advanced Queuing introduces high performing and scalable sharded JMS Queues. A sharded queue is a single logical queue that is divided into multiple, independent, physical queues through system-maintained partitioning. A sharded queue increases enqueue-dequeue throughput, especially across Oracle RAC instances, because ordering between two messages on different queue shards is best effort. Each shard is ordered based on enqueue time within a session. Sharded queues automatically manage table partitions so that enqueuers and dequeuers do not contend among themselves. In addition, sharded queues use an in-memory message cache to optimize performance and reduce the disk and CPU overhead of AQ-JMS enqueues and dequeues. Sharded queues are the preferred JMS queues for queues used across Oracle RAC instances, for queues with high enqueue or dequeue rates, or for queues with many subscribers.

  In 12.2, Sharded Queues have been enhanced to support more than JMS. See Sharded Queues for more information.

- Result Cache Enhancement

  In Oracle Database 12*c* Release 1 (12.1), the Rules Engine introduces a result cache to improve the performance of many commonly used rules. The result cache will bypass the evaluation phase if an expression with the same attributes has already been evaluated earlier. Not all rule results are cached, such as when rule results are potentially non-deterministic or when not all rules are evaluated or when attributes include non-scalar data types. For Advanced Queues, the cache is most useful when subscriptions and their dequeue sessions are long-lived.

- `LONG VARCHAR` Support

  The `LONG VARCHAR` data type is supported by Oracle Database Advanced Queuing in Oracle Database 12*c* Release 1 (12.1).

- 3-Tier Background Architecture

  Oracle Database 12c Release 1 (12.1) introduces the AQ background process architecture with a new a 3-tier design.

  See "AQ Background Architecture" for more information.

- Support for Data Guard Database Rolling Upgrade

Databases that use Oracle Database Advanced Queuing can now be upgraded to new Oracle database releases and patch sets in rolling fashion using Data Guard database rolling upgrades (transient logical standby database only). Rolling upgrades are supported beginning in Oracle Database 12*c* Release 1 (12.1).

Data Guard database rolling upgrades reduce planned downtime by enabling the upgrade to new database releases or patch sets in rolling fashion. Total database downtime for such an upgrade is limited to the small amount of time required to execute a Data Guard switchover.

The following packages will have support for rolling upgrade using logical standby:

– `DBMS_AQ`

– `DBMS_AQJMS`

– `DBMS_AQADM`, except for the following procedures:

  * `SCHECULE_PROPAGATION`
  * `UNSCHEDULE_PROPAGATION`
  * `ALTER_PROPAGATION_SCHEDULE`
  * `ENABLE_PROPAGATION_SCHEDULE`
  * `DISABLE_PROPAGATION_SCHEDULE`

See Also:

– *Oracle Database PL/SQL Packages and Types Reference* for more information on the Oracle Database AQ packages

1 Introduction to Transactional Event Queues and Advanced Queuing

Transactional Event Queues (TxEventQ) and Advanced Queuing (AQ) are robust and feature-rich message queuing systems integrated with Oracle database. When web, mobile, IoT, and other data-driven and event-driven applications stream events, or communicate with each other as part of a workflow, producer applications enqueue messages and consumer applications dequeue messages.

These topics discuss the newly introduced Transactional Event Queues (TxEventQ) that are highly optimized implementation of AQ previously called AQ Sharded Queues. Both TxEventQ and AQ in the Oracle database address the requirements from data-driven and event-driven architectures in modern enterprise applications.

What Is Queuing?

Transactional Event Queue (TxEventQ) and Advanced Queuing (AQ) stores user messages in abstract storage units called queues. When web, mobile, IoT, and other data-driven and event-driven applications stream events or communicate with each other as part of a workflow, producer applications enqueue messages and consumer applications dequeue messages.

At the most basic level of queuing, one producer enqueues one or more messages into one queue. Each message is dequeued and processed once by one of the consumers. A message stays in the queue until a consumer dequeues it or the message expires. A producer can stipulate a delay before the message is available to be consumed, and a time after which the message expires. Likewise, a consumer can wait when trying to dequeue a message if no message were available. An agent program or application could act as both a producer and a consumer.

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

At a slightly higher level of complexity, many producers enqueue messages into a queue, all of which are processed by one consumer. Or many producers enqueue messages, each message being processed by a different consumer depending on type and correlation identifier.

Enqueued messages are said to be propagated when they are reproduced on another queue, which can be in the same database or in a remote database.

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

Transactional Event Queues Leverage Oracle Database

Oracle Transactional Event Queues (TxEventQ) provide database-integrated message queuing functionality. This highly optimized and partitioned implementation leverages the functions of Oracle database so that producers and consumers can exchange messages with high throughput, by storing messages persistently, and propagate messages between queues on different databases. Oracle Transactional Event Queues (TxEventQ) are a high performance partitioned implementation with multiple event streams per queue, while Advanced Queuing (AQ) is a disk-based implementation for simpler workflow use cases.

Naming nomenclature for TxEventQ and AQ in Oracle Database Release 20c are as follows:

Message type	Old Name	New Name
Persistent messages	AQ classic queues	AQ queues
Persistent messages	AQ Sharded queues	TxEventQ queues
Buffered messages	AQ classic queues	AQ buffered queues

You can decide about which queue to use as follows:

- For buffered messages use AQ buffered queues.
- For persistent messages, use the high performance Transactional Event Queues.
- If you are currently using AQ classic queues, then consider moving to Transactional Event Queues with one event stream (to preserve total ordering in the queue) or consider taking advantage of multiple event streams where messages are ordered within each event stream. This is similar to Apache Kafka's approach of Topics consisting of multiple partitions to which producers and consumers can publish to or subscribe from.

AQ sharded queues are being deprecated in this release.

Because TxEventQs are implemented in database tables, all operational benefits of high availability, scalability, and reliability are also applicable to queue data. Standard database features such as recovery, restart, and security are supported by TxEventQ. You can use standard database development and management tools to monitor queues. Like other database tables, queue tables can be imported and exported. Similarly, TxEventQ queues are supported by Oracle Data Guard for high availability, which can be critical to preserve messages when using a stateless middle tier. By being in the database, enqueues and dequeues can be incorporated in database transactions without requiring distributed transactions

Messages can be queried using standard SQL. This means that you can use SQL to access the message properties, the message history, and the payload. With SQL access you can also audit and track messages. All available SQL technology, such as in-memory latches, table indices, are used to optimize access to messages in TxEventQ and AQ.

Note:

TxEventQ and AQ do not support data manipulation language (DML) operations on a queue table, or associated index-organized table (IOT) for AQ, or associated system-partitioned tables used by TxEventQs, if any. The only supported means of modifying queue tables is through the supplied APIs. Queue tables and IOTs can become inconsistent and therefore effectively ruined, if DML operations are performed on them.

System-Level Access Control

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

Queue-Level Access Control

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

Performance

Requests for service must be separated from the supply of services to increase efficiency and enable complex scheduling. TxEventQ and AQ deliver high performance as measured by:

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

Scalability

Queuing systems must be scalable. TxEventQ and AQ deliver high performance when the number of programs using the application increases, when the number of messages increases, and when the size of the message warehouse increases.

Persistence for Security

Messages that constitute requests for service must be stored persistently and processed exactly once for deferred execution to work correctly in the presence of network, computer, and application failures. TxEventQ and AQ can meet requirements in the following situations:

- Applications do not have the resources to handle multiple unprocessed messages arriving simultaneously from external clients or from programs internal to the application.
- Communication links between databases are not available all the time or are reserved for other purposes. If the system falls short in its capacity to deal with these messages

immediately, then the application must be able to store the messages until they can be processed.

- External clients or internal programs are not ready to receive messages that have been processed.

Persistence for Scheduling

Queuing systems must deal with priorities, and those priorities can change:

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

Persistence for Accessing and Analyzing Metadata

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

Object Type Support

TxEventQ and AQ support enqueue, dequeue, and propagation operations where the queue type is an abstract datatype, ADT. It also supports enqueue and dequeue operations if the types are inherited types of a base ADT. Propagation between two queues where the types are inherited from a base ADT is not supported.

TxEventQ and AQ also support `ANYDATA` queues, which enable applications to enqueue different message types in a single queue. TxEventQ and AQ support the `LONG VARCHAR` data type.

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

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

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

The object identifier need not match at each database.

Structured and XMLType Payloads

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

- Content-based routing

  TxEventQ and AQ can examine the content and automatically route the message to another queue based on the content.
- Content-based subscription

  A publish and subscribe system is built on top of a messaging system so that you can create subscriptions based on content.
- Querying

  The ability to run queries on the content of the message enables message warehousing.

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

- Store any type of message in a queue
- Store `m`ore than one type of payload in a queue
- Query XMLType columns using the operator `ExistsNode()`
- Specify the operators in subscriber rules or dequeue conditions

Integration with Oracle Internet Directory

You can register system events, user events, and notifications on queues with Oracle Internet Directory. System events are database startup, database shutdown, and system error events. User events include user log on and user log off, DDL statements (create, drop, alter), and DML statement triggers. Notifications on queues include OCI notifications, PL/SQL notifications, and e-mail notifications.

You can also create aliases for TxEventQ and AQ agents in Oracle Internet Directory. These aliases can be specified while performing TxEventQ and AQ enqueue, dequeue, and notification operations. This is useful when you do not want to expose an internal agent name.

> **Note:**
>
> Transactional Event Queues (TxEventQ) does not support OCI and thick drivers.

Support for Oracle Real Application Clusters(Oracle RAC)

Oracle Real Application Clusters can be used to improve TxEventQ and AQ performance by allowing different queues (and event streams in the case of TxEventQ) to be managed by different instances. You do this by specifying different instance affinities (preferences) for the queue tables that store the queues. This allows queue operations (enqueue and dequeue) on different queues to occur in parallel. TxEventQs are recommended for applications with enqueuers or dequeuers on multiple Oracle RAC instances. Refer to Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC) for more information.

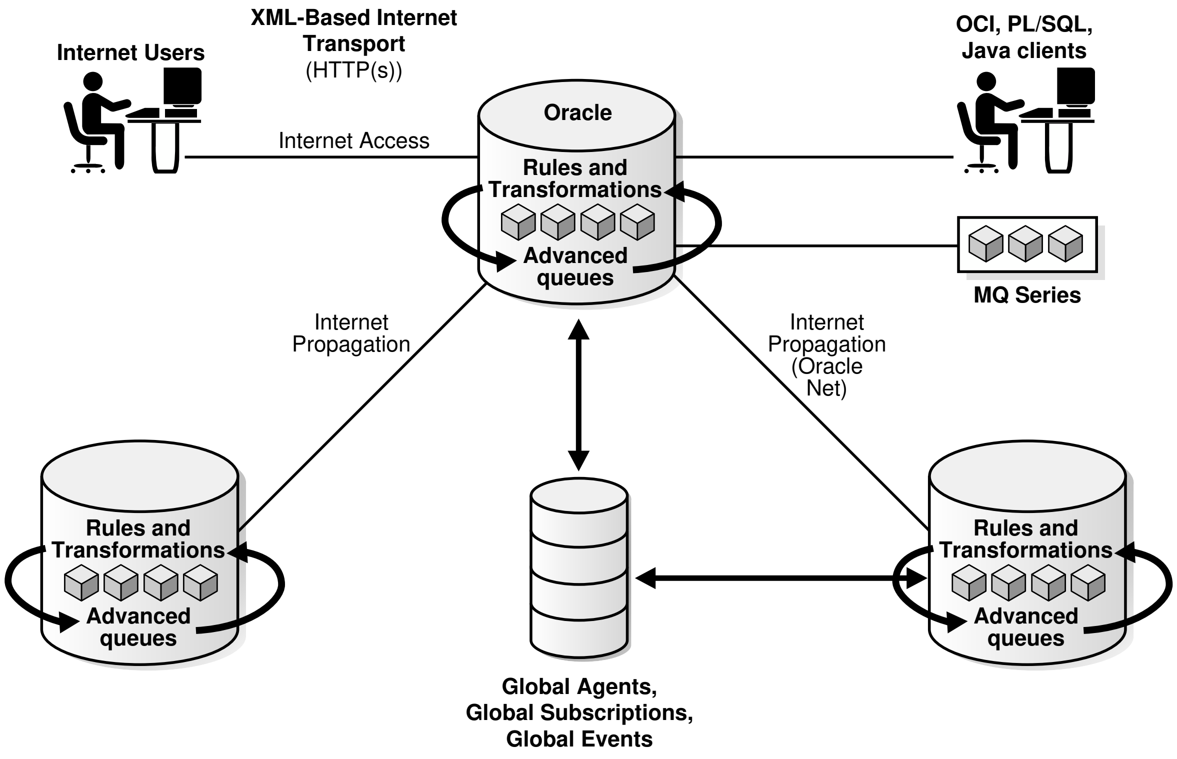
If compatibility is set to Oracle8*i* release 8.1.5 or higher, then an application can specify the instance affinity for a queue table. When TxEventQ and AQ is used with Oracle RAC and multiple instances, this information is used to partition the queue tables between instances for queue-monitor scheduling and, also for propagation. The queue table is monitored by the queue monitors of the instance specified by the user. If the owner of the queue table is terminated, then the secondary instance or some available instance takes over the ownership for the queue table.

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

Transactional Event Queues and Advanced Queuing in Integrated Application Environments

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

Figure 1-1 Integrated Application Environment Using TxEventQ and AQ


Internet Users
XML-Based Internet Transport (HTTP(s))
OCI, PL/SQL, Java clients
Oracle
Internet Access
Rules and Transformations
Advanced queues
MQ Series
Internet Propagation
Internet Propagation (Oracle Net)
Rules and Transformations
Advanced queues
Rules and Transformations
Advanced queues
Global Agents, Global Subscriptions, Global Events

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

Application integration also involves the integration of heterogeneous messaging systems. Oracle Database Advanced Queuing seamlessly integrates with existing non-Oracle Database messaging systems like IBM WebSphere MQ through Messaging Gateway, thus allowing existing WebSphere MQ-based applications to be integrated into an Oracle Database Advanced Queuing environment. Oracle Transactional Event Queues can interoperate with Apache Kafka using a Kafka JMS connector. TxEventQ can also work with a Kafka Java client. Both capabilities are described in Kafka APIs for Oracle Transactional Event Queues.

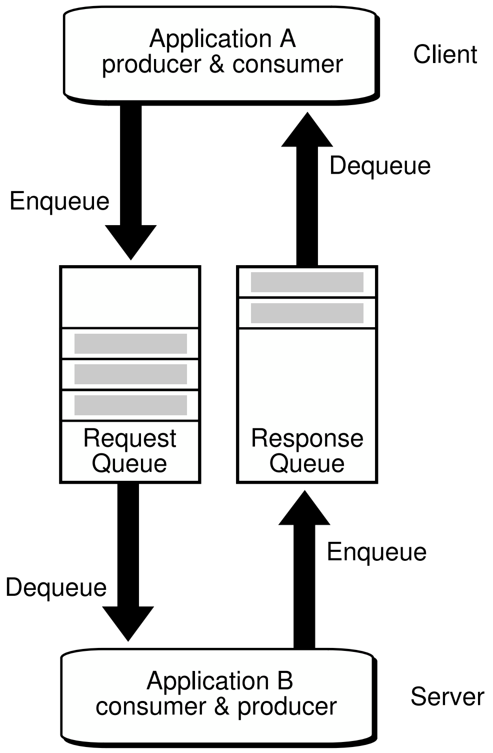
Topics:

- Oracle Database Advanced Queuing Client/Server Communication
- Multiconsumer Dequeuing of the Same Message
- Oracle Database Advanced Queuing Implementation of Workflows
- Oracle Database Advanced Queuing Implementation of Publish/Subscribe

Transactional Event Queues and Advanced Queuing Client/Server Communication

TxEventQ and AQ provide an asynchronous alternative to the synchronous manner in which Client/Server applications usually run. This figure exemplifies Client/Server Communication Using TxEventQ and AQ.
In this example Application B (a server) provides service to Application A (a client) using a request/response queue.

Figure 1-2   Client/Server Communication Using TxEventQ and AQ



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

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

Multiconsumer Dequeuing of the Same Message

A message can only be enqueued into one queue at a time. If a producer had to insert the same message into several queues in order to reach different consumers, then this would require management of a very large number of queues. TxEventQ and AQ provides for queue subscribers and message recipients to allow multiple consumers to dequeue the same message.
To allow for subscriber and recipient lists, the queue must reside in a queue table that is created with the multiple consumer option. Each message remains in the queue until it is consumed by all its intended consumers.

Queue Subscribers

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

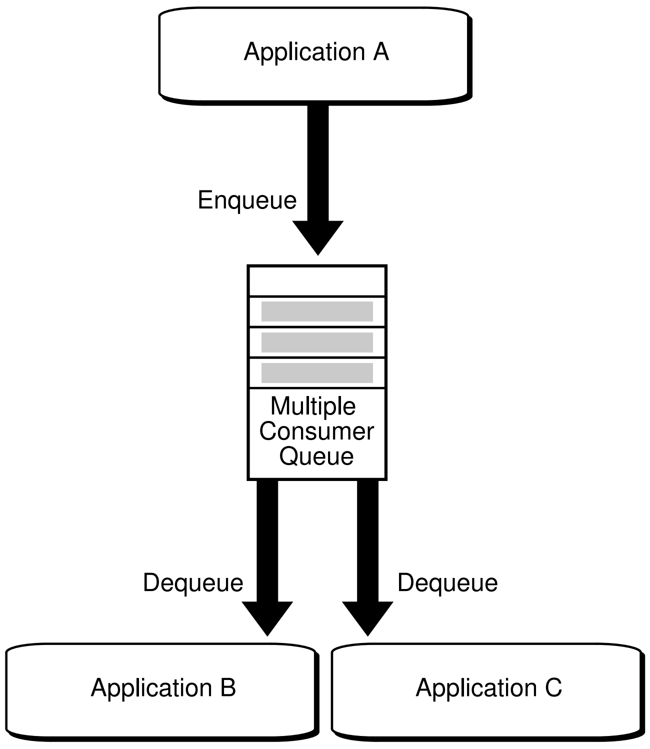
You cannot add subscriptions to single-consumer queues or exception queues. A consumer that is added as a subscriber to a queue is only able to dequeue messages that are enqueued after the subscriber is added. No two subscribers can have the same values for name, address, and protocol. At least one of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

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

Subscribers can also be rule-based. Similar in syntax to the `WHERE` clause of a SQL query, rules are expressed in terms of attributes that represent message properties or message content. These subscriber rules are evaluated against incoming messages, and those rules that match are used to determine message recipients.

In Figure 1-3, Application B and Application C each need messages produced by Application A, so a multiconsumer queue is specially configured with Application B and Application C as queue subscribers. Each receives every message placed in the queue.

Figure 1-3 Communication Using a Multiconsumer Queue



Message Recipients

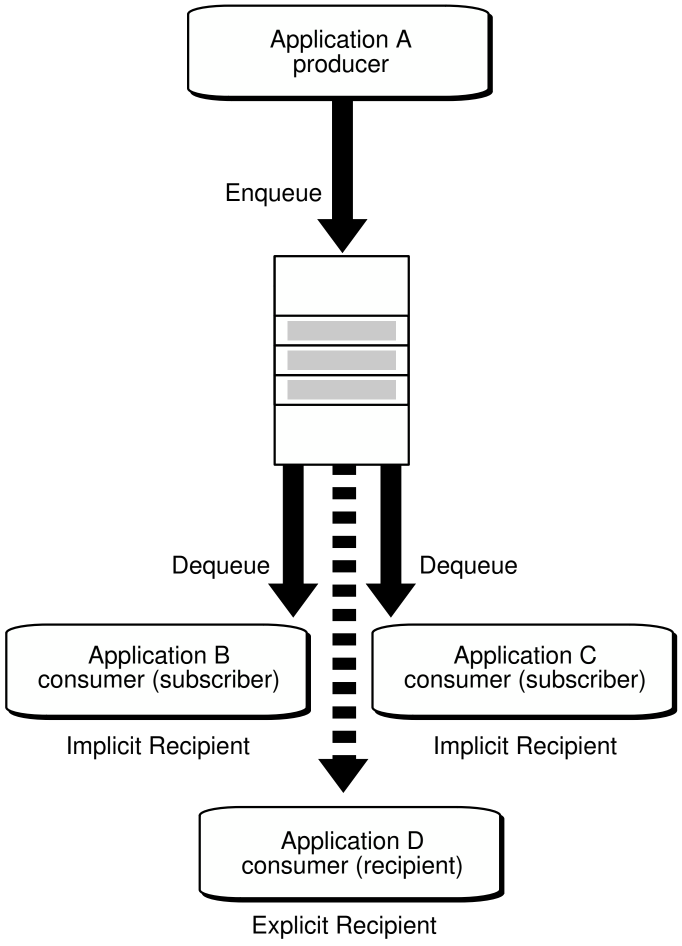
A message producer can submit a list of recipients at the time a message is enqueued into a TxEventQ or AQ queue. This allows for a unique set of recipients for each message in the queue. The recipient list associated with the message overrides the subscriber list associated with the queue, if there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

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

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

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

Figure 1-4 Explicit and Implicit Recipients of Messages



Note:

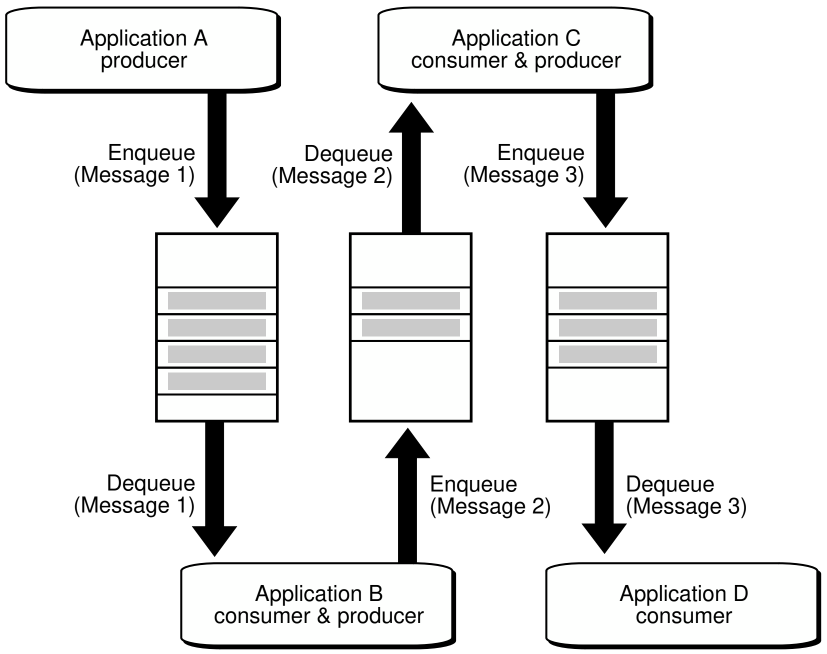
Multiple producers can simultaneously enqueue messages aimed at different targeted recipients.

Transactional Event Queues and Advanced Queuing Implementation of Workflows

TxEventQ and AQ allows us to implement a workflow, also known as a chained application transaction.The figure exemplifies how a workflow is implemented using TxEventQ and AQ.

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

Figure 1-5 Implementing a Workflow using TxEventQ and AQ



Note:

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

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

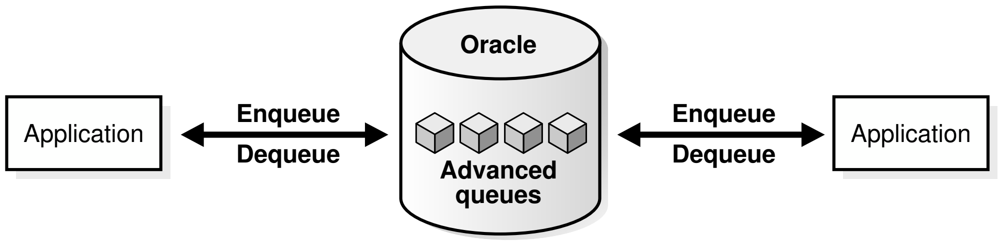
From a workflow perspective, knowledge of the volume and timing of message flows is a business asset quite apart from the value of the payload data. TxEventQ and AQ helps you gain this knowledge by supporting the optional retention of messages for analysis of historical patterns and prediction of future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

Transactional Event Queues and Advanced Queuing Implementation of Publish/Subscribe

A point-to-point message is aimed at a specific target. Senders and receivers decide on a common queue in which to exchange messages. Each message is consumed by only one receiver.

Figure 1-6 shows that each application has its own message queue, known as a single-consumer queue.

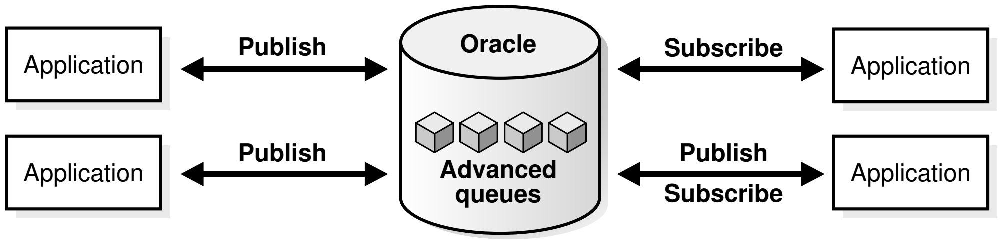
Figure 1-6 Point-to-Point Messaging



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

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

Figure 1-7 Publish/Subscribe Mode



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

You can implement a publish/subscribe model of communication using TxEventQ and AQ as follows:

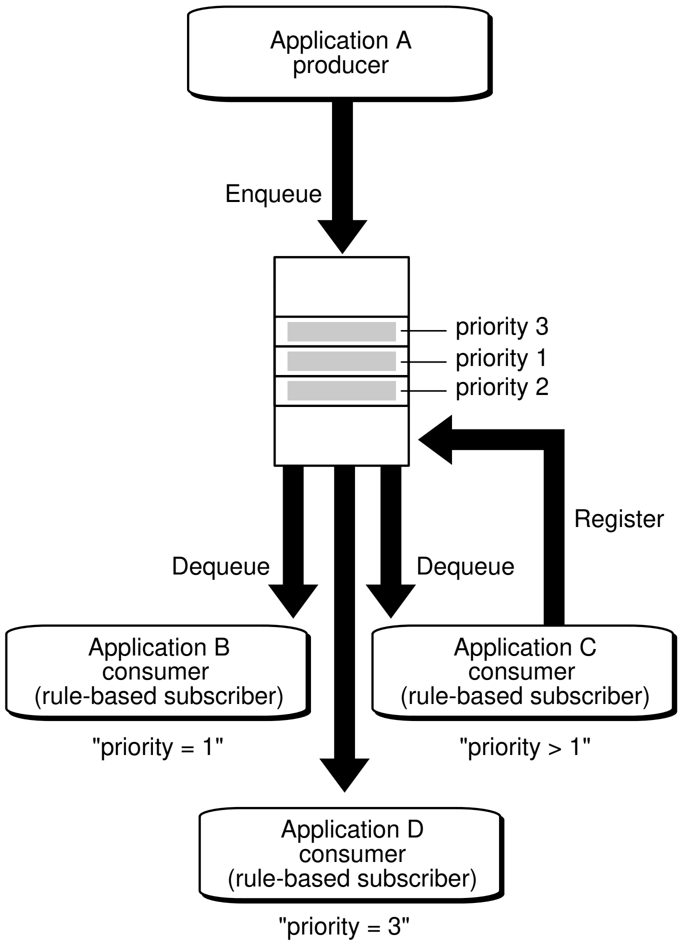
1. Set up one or more queues to hold messages. These queues should represent an area or subject of interest. For example, a queue can be used to represent billed orders.
2. Set up a set of rule-based subscribers. Each subscriber can specify a rule which represents a specification for the messages that the subscriber wishes to receive. A null rule indicates that the subscriber wishes to receive all messages.
3. Publisher applications publish messages to the queue by invoking an enqueue call.
4. Subscriber applications can receive messages with a dequeue call. This retrieves messages that match the subscription criteria.
5. Subscriber applications can also use a listen call to monitor multiple queues for subscriptions on different queues. This is a more scalable solution in cases where a subscriber application has subscribed to many queues and wishes to receive messages that arrive in any of the queues.

6. Subscriber applications can also use the Oracle Call Interface (OCI) notification mechanism. This allows a push mode of message delivery. The subscriber application registers the queues (and subscriptions specified as subscribing agent) from which to receive messages. This registers a callback to be invoked when messages matching the subscriptions arrive.

Figure 1-8 illustrates the use of TxEventQ and AQ for implementing a publish/subscribe relationship between publisher Application A and subscriber Applications B, C, and D:

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

Figure 1-8 Implementing Publish/Subscribe using TxEventQ and AQ



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

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

Buffered Messaging

Buffered messaging combines the rich functionality that this product has always offered with a much faster queuing implementation. Buffered messaging is ideal for applications that do not require the reliability and transaction support of Oracle Database Advanced Queuing persistent messaging.

Buffered messaging is faster than persistent messaging, because its messages reside in shared memory. They are usually written to disk only when the total memory consumption of buffered messages approaches the available shared memory limit.

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

Message retention is not supported for buffered messaging.

When using buffered messaging, Oracle recommends that you do one of the following:

- Set parameter `streams_pool_size`

  This parameter controls the size of shared memory available to Oracle Database Advanced Queuing. If unspecified, up to 10% of the shared pool size may be allocated for the Oracle Database Advanced Queuing pool from the database cache.

  Refer to manually tuning sharded queues for more information about setting the message cache for buffered messaging with TxEventQs.
- Turn on SGA autotuning

  Oracle will automatically allocate the appropriate amount of memory from the SGA for Oracle Database Advanced Queuing, based on Oracle Database Advanced Queuing usage and, also usage of other components that use the SGA. Examples of such other components are buffer cache and library cache. If `streams_pool_size` is specified, it is used as the lower bound.

Topics:

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

Enqueuing Buffered Messages

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

Recipient lists are supported for buffered messaging enqueue.

Buffered messaging is supported in all queue tables created with compatibility 8.1 or higher. Transaction grouping queues and array enqueues are not supported for buffered messages in this release. You can still use the array enqueue procedure to enqueue buffered messages, but the array size must be set to one.

Buffered messages can be queried using the `AQ$`*`Queue_Table_Name`* view. They appear with states `IN-MEMORY` or `SPILLED`.

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

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

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

See Also:

- "Enqueuing Messages"
- "AQ$<*Queue_Table_Name*>: Messages in Queue Table"
- "Priority and Ordering of Messages in Enqueuing"

Dequeuing Buffered Messages

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

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

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

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

See Also:

- "Adding a Subscriber"
- "Dequeue Options"

Propagating Buffered Messages

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

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

See Also:

- "DBA_QUEUE_SCHEDULES: All Propagation Schedules"
- "Buffered Messaging with Oracle Real Application Clusters (Oracle RAC)"

Flow Control

Oracle Database Advanced Queuing implements a flow control system that prevents applications from flooding the shared memory with messages. If the number of outstanding messages per sender exceeds a system-defined threshold, the enqueue call will block and timeout with an error message. A message sender is identified by *sender_id.name* in the enqueue options. A sender blocked due to flow control on a queue does not affect other message senders. The resolution is to dequeue messages, thereby resolving flow control, after which new messages can be enqueued.

Flow control threshold varies with memory pressure and could come down to the system-defined limit if streams pool usage becomes significant. Message senders will block on event `Streams AQ: enqueue blocked due to flow control` and time out with error `ORA-25307` if flow control is not resolved. Applications are expected to handle this error, and re-enqueue the failed message.

Even with flow control, slow consumers of a multiconsumer queue can cause the number of messages stored in memory to grow without limit. Provided there is at least one subscriber who is keeping pace, older messages are spilled to disk and removed from the pool to free up memory. This ensures that the cost of disk access is paid by the slower consumers, and faster subscribers can proceed unhindered.

Buffered Messaging with Oracle Real Application Clusters (Oracle RAC)

TxEventQ and AQ queues handle buffered messaging with Oracle RAC differently.

TxEventQs perform cross-instance communication but avoid simultaneous writes to the same block across Oracle RAC instances. Typically, dequeues occur on an event stream that is local to a message’s enqueuing instance, but in certain situations, Oracle will efficiently forward messages across instances for dequeuing on another instance. For example, if a TxEventQ has a single enqueuing session on one Oracle RAC instance and a single dequeuing session on another instance, then TxEventQs will forward messages between the Oracle RAC instances. The forwarding of messages is done asynchronously to the enqueuing transaction to improve performance. Dequeuers may get an ORA-25228 if they are connected to an instance whose event streams have no messages.

For AQ queues, an application can enqueue and dequeue buffered messages from any Oracle RAC instance as long as it uses password-based authentication to connect to the database. The structures required for buffered messaging are implemented on one Oracle RAC instance. The instance where the buffered messaging structures are implemented is the `OWNER_INSTANCE` of the queue table containing the queue. Enqueue and dequeue requests received at other instances are forwarded to the `OWNER_INSTANCE` over the interconnect. The `REMOTE_LISTENER` parameter in `listener.ora` must also be set to enable forwarding of buffered messaging requests to correct instance. Internally, buffered queues on Oracle RAC may use `dblinks` between instances. Definer's rights packages that enqueue or dequeue into buffered queues on Oracle RAC must grant `INHERIT REMOTE PRIVILEGES` to users of the package.

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

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

See Also:

- "ALL_QUEUE_TABLES: Queue Tables Accessible to the Current User" for more information on `OWNER_INSTANCE`
- "REMOTE_LISTENER" in *Oracle Database Reference* for more information on setting the `REMOTE_LISTENER` parameter
- "DBA_QUEUES: All Queues in Database" or "USER_QUEUES: Queues In User Schema"
- "Support for Oracle Real Application Clusters(Oracle RAC)"

Buffered Messaging Restrictions

The following Oracle Database Advanced Queuing features are only supported for buffered messaging on TxEventQs:

- Message delay
- Array enqueue
- Array dequeue
- PL/SQL Notification

The following Oracle Database Advanced Queuing features are not currently supported for buffered messaging:

- Message retention

- Transaction grouping
- Message export and import
- Messaging Gateway
- OCI notification

Error Handling

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

Asynchronous Notifications

Asynchronous notification allows clients to receive notifications of messages of interest.

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

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

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

Note:

TxEventQs only support PL/SQL notification.

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

Payload Delivery for RAW Queues

For RAW queues, Oracle Database Advanced Queuing clients can now specify that the message payload be delivered along with its notification.

See Also:

"AQ Registration Information Type"

Reliable Notification

Clients can specify persistent message notification. If an Oracle RAC instance fails, its notifications are delivered by another Oracle RAC node. If a standalone instance fails, its notifications are delivered when the instance restarts.

Note:

Notification reliability refers only to server failures. If Oracle Database Advanced Queuing cannot deliver client notifications for any other reason, then the notifications are purged along with the client registration.

Designated Port Notification

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

Note:

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

See Also:

"Publish-Subscribe Registration Functions in OCI" in *Oracle Call Interface Programmer's Guide*

IPv6 Compliance and Designated IP Support

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

Registration Timeout

In earlier releases of Oracle Database Advanced Queuing, registrations for notification persisted until explicitly removed by the client or purged in case of extended client failure. From

Oracle Database Advanced Queuing 10*g* Release 2 (10.2) onwards, clients can register for a specified time, after which the registration is automatically purged.

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

See Also:

"AQ Registration Information Type" for information on the `timeout` parameter

Purge on Notification

Clients can also register to receive only the first notification, after which the registration is automatically purged.

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

Buffered Message Notification

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

See Also:

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

Reliable notification is not supported.

Views on Registration

The dictionary views `DBA_SUBSCR_REGISTRATIONS` and `USER_SUBSCR_REGISTRATIONS` display the various registrations in the system.

The diagnostic view `GV$SUBSCR_REGISTRATION_STATS` may be used to monitor notification statistics and performance.

Event-Based Notification

Event-based notifications are processed by a set of coordinator (`EMNC`) and subordinate processes.

The event notification load is distributed among these processes. These processes work on the system notifications in parallel, offering a capability to process a larger volume of notifications, a faster response time and lower shared memory use for staging notifications.

Notification Grouping by Time

Notification applications may register to receive a single notification for all events that occur within a specified time interval. Notification Clients may specify a start time for the notifications. Additionally, they must specify a time as the grouping class and the time interval as the grouping value.

A repeat count may be used to limit the number of notifications delivered. Clients can receive two types of grouping events, Summary or Last. A summary notification is a list of Message Identifiers of all the messages for the subscription. If last was specified as a grouping type, notification would have information about the last message in the notification interval. A count of the number of messages in the interval is also sent. The registration interfaces in PLSQL and OCI allow for specification of the `START_TIME`, `REPEAT_COUNT`, `GROUPING CLASS`, `GROUPING VALUE`, `GROUPING TYPE` in the `AQ$_REGISTRATION_INFO` and the OCI subscription Handle.

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

See Also:

- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Call Interface Programmer's Guide*

Enqueue Features

This topic describes the enqueue features Enqueuing an Array of Messages, Correlation Identifiers, Priority and Ordering of Messages in Enqueuing, Message Grouping, Sender Identification, and Time Specification and Scheduling.

The following features apply to enqueuing messages:

- Enqueue an Array of Messages
- Correlation Identifiers
- Priority and Ordering of Messages in Enqueuing
- Message Grouping
- Sender Identification
- Time Specification and Scheduling

Enqueue an Array of Messages

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

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

Correlation Identifiers

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

Priority and Ordering of Messages in Enqueuing

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

- A priority can be assigned to each message.
- A sort order specifies which properties are used to order all messages in a queue. This is set when the queue table is created and cannot be changed. You can choose to sort messages by priority, enqueue time, or commit time.

  See Also:

  – "Creating a Queue Table" for more information on sort order

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

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

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

Message Grouping

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

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

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

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

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

Sender Identification

Applications can mark the messages they send with a custom identification. Oracle Database Advanced Queuing also automatically identifies the queue from which a message was dequeued. This allows applications to track the pathway of a propagated message or a string message within the same database.

Time Specification and Scheduling

Messages can be enqueued with an expiration that specifies the interval of time the message is available for dequeuing. The default for expiration is never. When a message expires, it is moved to an exception queue. Expiration processing requires that the queue monitor be running.

Dequeue Features

This topic discusses the dequeue features Concurrent Dequeues, Dequeue Methods, Dequeue Modes, Dequeue an Array of Messages, Message States, Navigation of Messages in Dequeuing, Waiting for Messages, Retries with Delays, Optional Transaction Protection, and Exception Queues.

The following features apply to dequeuing messages:

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

Concurrent Dequeues

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

Dequeue Methods

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

- Specifying a correlation identifier

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

- Specifying a message identifier

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

- Specifying a dequeue condition

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

- Default dequeue

  A default dequeue retrieves the first available message.

> **Note:**
>
> Dequeuing with correlation identifier, message identifier, or dequeue condition does not preserve the message grouping property.

Dequeue Modes

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

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

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

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

> **See Also:**
>
> - "Creating a Queue Table"

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

Dequeue an Array of Messages

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

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

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

Message States

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

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

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

Note:

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

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

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

Navigation of Messages in Dequeuing

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

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

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

Transaction grouping has no effect if you dequeue by specifying a correlation identifier or message identifier, or if you dequeue some of the messages of a transaction and then commit.

If you reach the end of the queue while using the `NEXT_MESSAGE` or `NEXT_TRANSACTION` option, and you have specified a nonzero wait time, then the navigating position is automatically changed to the beginning of the queue. If a zero wait time is specified, then you can get an exception when the end of the queue is reached.

Waiting for Messages

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

Note:

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

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

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

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

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

Retries with Delays

If the transaction dequeuing a message from a queue fails, then it is regarded as an unsuccessful attempt to consume the message. Oracle Database Advanced Queuing records the number of failed attempts to consume the message in the message history. Applications can query the `RETRY_COUNT` column of the queue table view to find out the number of unsuccessful attempts on a message. In addition, Oracle Database Advanced Queuing allows the application to specify, at the queue level, the maximum number of retries for messages in the queue. The default value for maximum retries is 5. If the number of failed attempts to remove a message exceeds this number, then the message is moved to the exception queue and is no longer available to applications.

Note:

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

A bad condition can cause the transaction receiving a message to end. Oracle Database Advanced Queuing allows users to hide the bad message for a specified retry delay interval, during which it is in the WAITING state. After the retry delay, the failed message is again available for dequeue. The Oracle Database Advanced Queuing time manager enforces the retry delay property. The default value for retry delay is 0.

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

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

Note:

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

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

Optional Transaction Protection

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

> **Note:**
>
> Transaction protection is not supported for buffered messaging.

Exception Queues

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

> **Note:**
>
> Expired or unserviceable buffered messages are moved to an exception queue as persistent messages.
>
> Messages intended for single-consumer queues, or for 8.0-style multiconsumer queues, can only be dequeued by their message identifiers once the messages have been moved to an exception queue.
>
> Queues created in a queue table with `compatible` set to `8.0` (referred to in this guide as 8.0-style queues) are deprecated in Oracle Database Advanced Queuing 10*g* Release 2 (10.2). Oracle recommends that any new queues you create be 8.1-style or newer and that you migrate existing 8.0-style queues at your earliest convenience.

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

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

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

- It was not dequeued within the specified expiration interval.

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

- The message was dequeued successfully, but the application that dequeued it rolled back the transaction because of an error that arose while processing the message. If the message has been dequeued but rolled back more than the number of times specified by the retry limit, then the message is moved to the exception queue.

  For a message intended for multiple recipients, a separate retry count is kept for each recipient. The message is moved to the exception queue only when retry counts for all recipients of the message have exceeded the specified retry limit.

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

  Note:

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

- The statement processed by the client contains a dequeue that succeeded but the statement itself was undone later due to an exception.

  If the dequeue procedure succeeds but the PL/SQL procedure raises an exception, then Oracle Database Advanced Queuing increments the retry count of the message returned by the dequeue procedure.

- The client program successfully dequeued a message but terminated before committing the transaction.

Propagation Features

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

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

Starting from 23, TxEventQ propagation creates number of schedules equal to the number of shards(Event Streams) present in the source queue. If the source queue has 5 shards, there will be 5 propagation schedules created with the same frequency, duration, and `next_time` defined by the user when `schedule_propagation` was executed. This one-to-one mapping of shard-schedule propagation increases the total throughput. This change can be seen in propagation related tables and views, for each shard there will be a row available at `sys.aq$_schedules` and `DBA_QUEUE_SCHEDULES`.

Note:

- You can propagate messages from a multi-consumer queue to a single-consumer queue. Propagation from a single-consumer queue to a multi-consumer queue is not possible.
- For AQ queues, you can propagate messages from a multi-consumer queue to a single-consumer queue. Propagation from a single-consumer queue to a multi-consumer queue is not possible.
- For TxEventQs, you can propagate between single-consumuer and multi-consumer queues.
- You cannot propagate between TxEventQ and AQ queues.

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

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

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

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

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

Oracle Database Advanced Queuing offers two kinds of propagation:

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

Queue-to-dblink propagation delivers messages or events from the source queue to all subscribing queues at the destination database identified by the dblink. Queue-to-dblink propagation feature is not supported for sharded queues. Propagation is always a queue-to-queue, and a destination queue that is remote can be accesses by a dblink.

A single propagation schedule is used to propagate messages to all subscribing queues. Hence any changes made to this schedule will affect message delivery to all the subscribing queues.

Queue-to-queue propagation delivers messages or events from the source queue to a specific destination queue identified on the dblink. This allows the user to have fine-grained control on the propagation schedule for message delivery.

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

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

Remote Consumers

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

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

- The address refers to a queue in the same database.

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

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

  You must refer to the documentation of the third party software to determine how to specify the address and the protocol database link and schedule propagation.

Propagation to Remote Subscribers

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

Oracle Database AQ offers two kinds of propagation:

A) **Queue-to-dblink propagation** - specified by providing a (source) queue and (destination) databaselink. Messages from the source queue for any queues at the destination specified by the dblink will be handled by this propagation. Queue-to-dblink propagation is not supported for sharded queues.

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

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

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

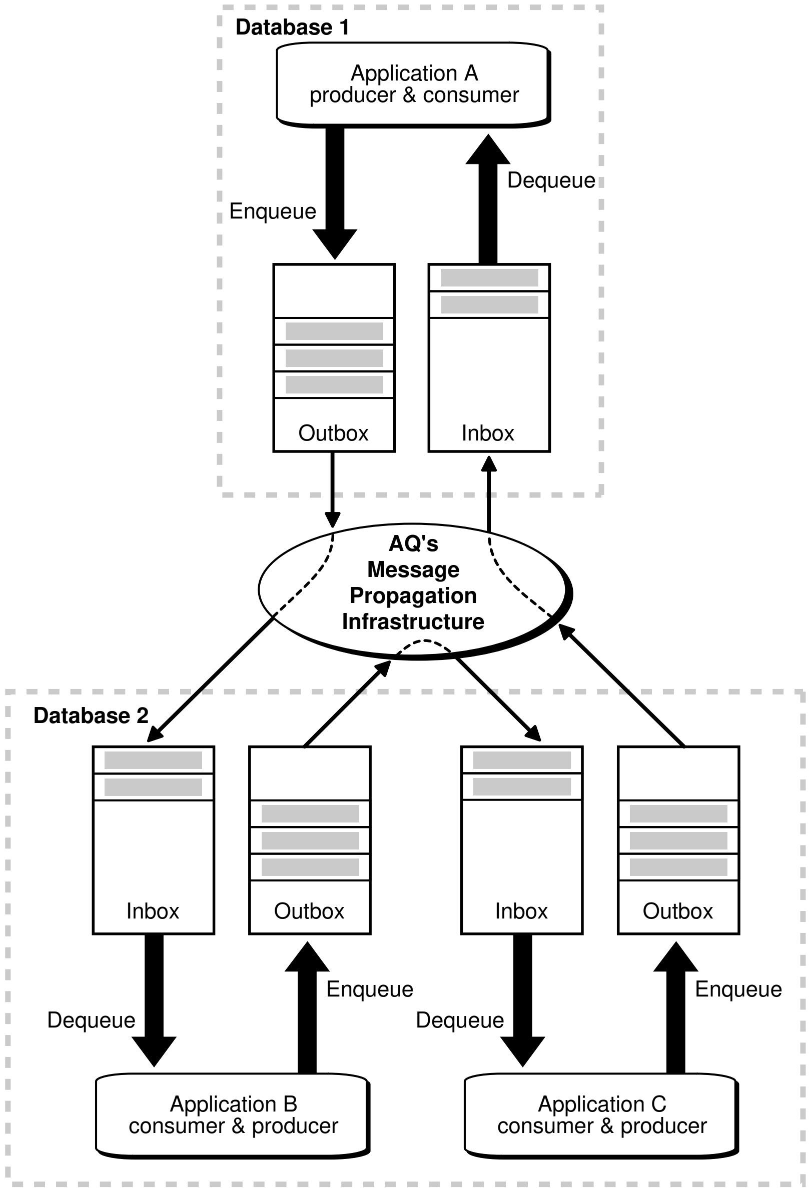
Priority and Ordering of Messages in Propagation

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

Inboxes and Outboxes

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

Figure 1-9   Message Propagation in Oracle Database Advanced Queuing



Propagation Scheduling

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

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

When a schedule is created, a job is automatically submitted to the job queue facility to handle propagation.

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

Propagation of Messages with LOBs

Large Objects can be propagated using Oracle Database Advanced Queuing using two methods:

- Propagation from `RAW` queues

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

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

  Note:

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

  See Also:

  Oracle Database SecureFiles and Large Objects Developer's Guide

Propagation Statistics

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

For each schedule, detailed propagation statistics are maintained:

- Total number of messages propagated in a schedule
- Total number of bytes propagated in a schedule
- Maximum number of messages propagated in a window

- Maximum number of bytes propagated in a window
- Average number of messages propagated in a window
- Average size of propagated messages
- Average time to propagated a message

Propagation Error Handling

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

If a schedule encounters failures, then the retry happens at every minute for 24 hours. Once we exhaust all the retry attempts, the frequency of retry will be changed to hourly once, and the max retries are set to `65535` by default. Users can change the max retry attempts by setting a queue level parameter `PROP_MAXRETRY_VAL` with a positive value within the range of `0`-`65535`.

Note:

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

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

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

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

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

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

Note:

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

Propagation with Oracle Real Application Clusters

Propagation has support built-in for Oracle Real Application Clusters. It is transparent to the user and the queue administrator. The job that handles propagation is submitted to the same instance as the owner of the queue table where the queue resides.

If there is a failure at an instance and the queue table that stores the source queue is migrated to a different instance, then the propagation job is also migrated to the new instance. This minimizes pinging between instances and thus offers better performance.

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

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

See Also:

"Scheduling a Queue Propagation" for more information on queue-to-queue propagation

Propagation has been designed to handle any number of concurrent schedules. The number of job queue processes is limited to a maximum of 4000, and some of these can be used to handle jobs unrelated to propagation. Hence, propagation has built-in support for multitasking and load balancing.

If propagation found to be starving other database client that are waiting for the job queue processes to complete their task, users can tweak the scheduling parameters. `SCHEDULE_PROPAGATION` or `ALTER_PROPAGATION_SCHEDULE` APIs provide duration and `next_time` parameters, by setting these parameters users can control the amount of time the `job_queue_processes` are going to execute a schedule. For instance, a 10 seconds duration and a `next_time` to `SYSTIMESTAMP`, that is, immediate time, will make the scheduler to yield the `job_queue_process` once 10 seconds elapsed, and re-schedule immediately after.

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

Transactional Event Queues (TxEventQ) Propagation with Oracle RAC

When the destination queue is created on an Oracle RAC setup, then schedule propagation using a Queue-to-dblink model. In a Queue-to-dblink propagation, single dblink is created and passed as a parameter to the `schedule_propagation` API. In case of a destination with Oracle RAC setup, create set of unique dblinks for each node at the destination.

The dblinks to the destination nodes should follow certain naming convention. The instance-ID is appended to the dblink name which is used in the `schedule_propagation` API. For example, if the propagation setup involves 2 nodes at the destination, a dblink with name '`samplelnk`' is passed to `schedule_propagation` API, and the instance IDs are `1` and `2`, then the rest of the dblinks will be '`samplelnk1`' and '`samplelnk2`'.

Create and validate these dblinks before scheduling the propagation. Internally these dblinks are used to access each nodes involved in the propagation schedule.

If any node at the destination is temporarily down for more than three minutes, then the current propagation schedule is stopped and the corresponding error message is logged in an alert log.

TxEventQ propagation on Oracle RAC needs to know the metadata from destination to redirect the propagation schedules in case of any instance going down in the destination database. This look up requires granting `SELECT_CATALOG_ROLE` to the propagation user.

Third-Party Support

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

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

Propagation Using HTTP

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

Message Format Transformation

Applications often use data in different formats. A transformation defines a mapping from one Oracle data type to another.

The transformation is represented by a SQL function that takes the source data type as input and returns an object of the target data type. Only one-to-one message transformations are supported.

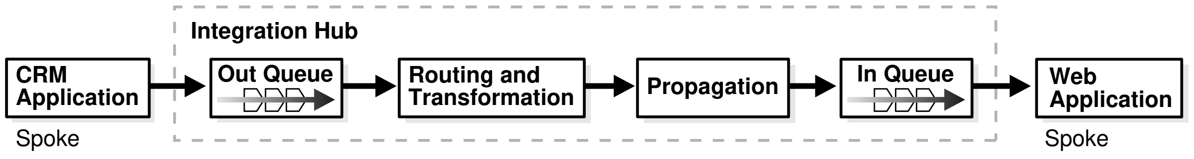
To transform a message during enqueue, specify a mapping in the enqueue options. To transform a message during dequeue, specify a mapping either in the dequeue options or when you add a subscriber. A dequeue mapping overrides a subscriber mapping. To transform a message during propagation, specify a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

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

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

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

Figure 1-10 Transformations in Application Integration



XML Data Transformation

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

Other Oracle Database Advanced Queuing Features

This topic describes the AQ features Queue Monitor Coordinator, Integration with Oracle Internet Directory, Integration with Oracle Enterprise Manager, Retention and Message History, Cleaning Up Message Queues, Tracking and Event Journals, Non-repudiation, and Internet Integration.

- Queue Monitor Coordinator
- Integration with Oracle Internet Directory
- Integration with Oracle Enterprise Manager
- Retention and Message History
- Cleaning Up Message Queues

- Tracking and Event Journals
- Non-repudiation
- Internet Integration

Queue Monitor Coordinator

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

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

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

Although it is no longer necessary to set `init.ora` parameter `AQ_TM_PROCESSES`, it is still supported. If you do set it (up to a maximum of 40), then QMNC still autotunes the number of processes. But you are guaranteed at least the set number of processes for persistent queues. Processes for a buffered queue however, are not affected by this parameter.

Note:

If you want to disable the Queue Monitor Coordinator, then you must set `AQ_TM_PROCESSES = 0` in your `pfile` or `spfile`. Oracle strongly recommends that you do NOT set `AQ_TM_PROCESSES = 0`.

Integration with Oracle Internet Directory

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

Integration with Oracle Enterprise Manager

You can use Oracle Enterprise Manager to:

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

Retention and Message History

The systems administrator specifies the retention duration to retain messages after consumption. Oracle Database 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, data mining operations, and, also specific auditing functions.

Message retention is not supported for buffered messaging.

Cleaning Up Message Queues

The Oracle Database Advanced Queuing retention feature can be used to automatically clean up messages after the user-specified duration after consumption.

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

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

Tracking and Event Journals

Retained messages can be related to each other to form sequences. These sequences represent event journals, which are often constructed by applications. Oracle Database Advanced Queuing is designed to let applications create event journals automatically.

Non-repudiation

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

The following information is kept at enqueue for non-repudiation of the enqueuer:

- Oracle Database Advanced Queuing agent doing the enqueue
- Database user doing the enqueue
- Enqueue time
- Transaction ID of the transaction doing enqueue

The following information is kept at dequeue for non-repudiation of the dequeuer:

- Oracle Database Advanced Queuing agent doing dequeue
- Database user doing dequeue
- Dequeue time
- Transaction ID of the transaction doing dequeue

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

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

Internet Integration

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

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

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

Figure 1-11 shows the architecture for performing Oracle Database Advanced Queuing operations over HTTP. The major components are:

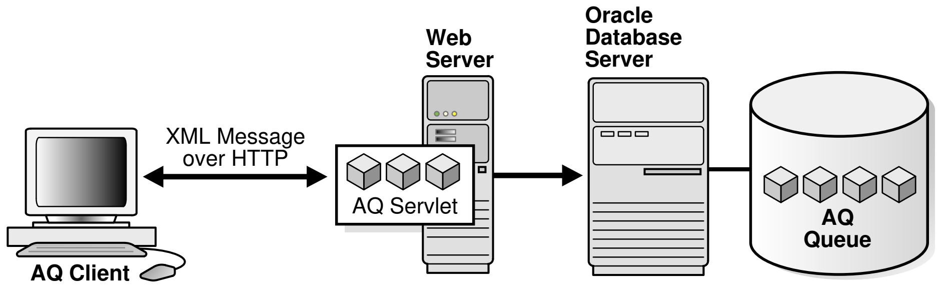
- Oracle Database Advanced Queuing client program
- Web server/servlet runner hosting the Oracle Database Advanced Queuing servlet
- Oracle Database server

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

Note:

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

Figure 1-11 Architecture for Performing Oracle Database Advanced Queuing Operations Using HTTP



Polyglot Programming with Transactional Event Queues

AQ and TxEventQ support multiple languages to use the messaging and streaming features in the database.

In addition to multiple languages, the queues support multiple message types like RAW, ADT, JMS Types, and JSON. For applications that use multiple languages to exchange messages, JSON is the recommended message type to use. It is the most widely supported format with the drivers of many languages. Message header, message body, and message properties are supported in this format by encoding them as the key:value structure in JSON (or nested JSON) format.

The following languages are supported by TxEventQ and AQ:

- PL/SQL using `DBMS_AQ`, `DBMS_AQADM`, and `DBMS_AQELM`
- Java (with JDBC driver)
- Java Message Service (JMS) using the `oracle.jms` Java package
- Python using the python-oracledb thick driver
- JavaScript using the `Node.js` node-oracledb driver
- .NET using the AQ classes
- Kafka Java Client APIs using the `oracle.kafka` Java package

See Also:

- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Database Advanced Queuing Java API Reference*
- python-oracledb API Documentation
- node-oracledb API Documentation
- *Oracle Data Provider for .NET Developer's Guide.*
- *Oracle Database Transactional Event Queues Java API Reference*

2

Basic Components of Oracle Transactional Event Queues and Advanced Queuing

This topic lists the basic components of Oracle Database Advanced Queuing.

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

See Also:

– Oracle Database Advanced Queuing Administrative Interface
– Oracle Database Advanced Queuing Operations Using PL/SQL

Object Name

This component names database objects.

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

This naming convention applies to queues, queue tables, and object types.

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, then the current schema is assumed. The name must follow the reserved character guidelines in *Oracle Database SQL Language ReferenceOracle Database SQL*

Language Reference. The schema name, agent name, and the object type name can each be up to 128 bytes long. However starting from 12c Release 2 (12.2.), queue names and queue table names can be a maximum of 122 bytes.

Type Name

This component defines queue types.

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

The maximum number of attributes in the object type is limited to 900.

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

Note:

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

AQ Agent Type

This component identifies a producer or consumer of a message.

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

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

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

This type has three attributes:

- `name`

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

- `address`

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

- `protocol`

  This attribute specifies the protocol to interpret the address and propagate the message. The default value is 0.

AQ Recipient List Type

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

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

AQ Agent List Type

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

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

AQ Subscriber List Type

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

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

AQ Registration Information List Type

This component identifies the list of registrations to a queue.

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

AQ Post Information List Type

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

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

AQ Registration Information Type

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

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

Its attributes are described in the following list.

Table 2-1 AQ$_REG_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 or *`schema.queue:consumer_name`* if the registration is for a multiconsumer queues.
`namespace`	Specifies the namespace of the subscription. To receive notification from Oracle Database AQ queues, the namespace must be `DBMS_AQ.NAMESPACE_AQ.` To receive notifications from other applications through `DBMS_AQ.POST` or `OCISubscriptionPost(),` the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS.`
`callback`	Specifies the action to be performed on message notification. For HTTP notifications, use `http://www.company.com:8080`. For e-mail notifications, use `mailto://xyz@company.com`. For raw message payload for the `PLSQLCALLBACK` procedure, use `plsql://`*`schema.procedure`*`?PR=0`. For user-defined type message payload converted to XML for the `PLSQLCALLBACK` procedure, use `plsql://`*`schema.procedure`*`?PR=1`
`context`	Specifies the context that is to be passed to the callback function
`qosflags`	Can be set to one or more of the following values to specify the notification quality of service: • `NTFN_QOS_RELIABLE`- This value specifies that reliable notification is required. Reliable notifications persist across instance and database restarts. • `NTFN_QOS_PAYLOAD` - This value specifies that payload delivery is required. It is supported only for client notification and only for `RAW` queues. • `NTFN_QOS_PURGE_ON_NTFN` - This value specifies that the registration is to be purged automatically when the first notification is delivered to this registration location.
`ntfn_grouping_class`	Currently, only the following flag can be set to specify criterion for grouping. The default value will be 0. If `ntfn_grouping_class` is 0, all other notification grouping attributes must be 0. • `NTFN_GROUPING_CLASS_TIME` - Notifications grouped by time, that is, the user specifies a time value and a single notification gets published at the end of that time.
`ntfn_grouping_value`	Time-period of grouping notifications specified in seconds, meaning the time after which grouping notification would be sent periodically until `ntfn_grouping_repeat_count` is exhausted.
`ntfn_grouping_type`	• `NTFN_GROUPING_TYPE_SUMMARY` - Summary of all notifications that occurred in the time interval. (Default) • `NTFN_GROUPING_TYPE_LAST` - Last notification that occurred in the interval.
`ntfn_grouping_start_time`	Notification grouping start time. Notification grouping can start from a user-specified time that should a valid timestamp with time zone. If `ntfn_grouping_start_time` is not specified when using grouping, the default is to current timestamp with time zone

Table 2-1 (Cont.) AQ$_REG_INFO Type Attributes

Attribute	Description
`ntfn_grouping_repeat_count`	Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. The ntfn_grouping_repeat_count, if not specified, will default to • `NTFN_GROUPING_FOREVER` - Keep sending grouping notifications forever.

AQ Notification Descriptor Type

This component specifies the Oracle Database Advanced Queuing descriptor received by AQ PL/SQL callbacks upon notification.

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

It has the following attributes:

Table 2-2 AQ$_DESCRIPTOR Attributes

Attribute	Description
`queue_name`	Name of the queue in which the message was enqueued which resulted in the notification
`consumer_name`	Name of the consumer for the multiconsumer queue
`msg_id`	Identification number of the message
`msg_prop`	Message properties specified by the `MSG_PROP_T` type
`gen_desc`	Indicates the timeout specifications
`msgid_array`	Group notification message ID list
`ntfnsRecdInGrp`	Notifications received in group

AQ Message Properties Type

The message properties type `msg_prop_t` has these components.

```
TYPE AQ$_MSG_PROP_T IS OBJECT(
   priority          number,
   delay             number,
   expiration        number,
   correlation       varchar2(128),
   attempts          number,
   recipent_list     aq$_recipient_list_t,
   exception_queue   varchar2(51),
   enqueue_time      date,
   state             number,
   sender_id        aq$_agent,
   original_msgid   raw(16),
   delivery_mode    number);
```

The timeout specifications type `AQ$_NTFN_DESCRIPTOR` has a single component:

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

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

See Also:

"MESSAGE_PROPERTIES_T Type" in *Oracle Database PL/SQL Packages and Types Reference*

AQ Post Information Type

This component specifies anonymous subscriptions to which you want to post messages.

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

It has three attributes:

- `name`

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

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

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

AQ$_NTFN_MSGID_ARRAY Type

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

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

Enumerated Constants for AQ Administrative Interface

When enumerated constants such as `INFINITE`, `TRANSACTIONAL`, and `NORMAL_QUEUE` are selected as values, the symbol must be specified with the scope of the packages defining it.

All types associated with the administrative interfaces must be prepended with `DBMS_AQADM`. For example:

```
DBMS_AQADM.NORMAL_QUEUE
```

Table 2-3 lists the enumerated constants in the Oracle Database Advanced Queuing administrative interface.

Table 2-3 Enumerated Constants in the Oracle Database Advanced Queuing Administrative Interface

Parameter	Options
`retention`	`0,1,2...INFINITE`
`message_grouping`	`TRANSACTIONAL`, `NONE`
`queue_type`	`NORMAL_QUEUE`, `EXCEPTION_QUEUE`,`NON_PERSISTENT_QUEUE`
`delivery_mode`	`BUFFERED`, `PERSISTENT`, `PERSISTENT_OR_BUFFERED`

Note:

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

Enumerated Constants for AQ Operational Interface

When using enumerated constants such as `BROWSE`, `LOCKED`, and `REMOVE`, the PL/SQL constants must be specified with the scope of the packages defining them.

All types associated with the operational interfaces must be prepended with `DBMS_AQ`. For example:

```
DBMS_AQ.BROWSE
```

Table 2-4 lists the enumerated constants in the Oracle Database Advanced Queuing operational interface.

Table 2-4 Enumerated Constants in the Oracle Database Advanced Queuing 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`
`wait`	`FOREVER`, `NO_WAIT`
`delay`	`NO_DELAY`
`expiration`	`NEVER`
`namespace`	`NAMESPACE_AQ`, `NAMESPACE_ANONYMOUS`

Table 2-4 (Cont.) Enumerated Constants in the Oracle Database Advanced Queuing Operational Interface

Parameter	Options
`delivery_mode`	`BUFFERED`, `PERSISTENT`, `PERSISTENT_OR_BUFFERED`
`quosflags`	`NTFN_QOS_RELIABLE`, `NTFN_QOS_PAYLOAD`, `NTFN_QOS_PURGE_ON_NTFN`
`ntfn_grouping_class`	`NFTN_GROUPING_CLASS_TIME`
`ntfn_grouping_type`	`NTFN_GROUPING_TYPE_SUMMARY`, `NTFN_GROUPING_TYPE_LAST`
`ntfn_grouping_repeat_count`	`NTFN_GROUPING_FOREVER`

AQ Background Processes

These topics describe the background processes of Oracle Database Advanced Queuing.

- Queue Monitor Processes
- Job Queue Processes
- AQ Background Architecture

Queue Monitor Processes

Oracle recommends leaving the `AQ_TM_PROCESSES` parameter unspecified and let the system autotune.

Many Oracle Database Advanced Queuing tasks are executed in the background. These include converting messages with `DELAY` specified into the `READY` state, expiring messages, moving messages to exception queues, spilling and recovering of buffered messages, and similar operations.

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

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

Note:

If you want to disable the Queue Monitor Coordinator, then you must set `AQ_TM_PROCESSES = 0` in your `pfile` or `spfile`. Oracle strongly recommends that you do NOT set `AQ_TM_PROCESSES = 0`.

Job Queue Processes

Propagation and PL/SQL notifications are handled by job queue (Jnnn) processes.

The parameter `JOB_QUEUE_PROCESSES` no longer needs to be specified. The database scheduler automatically starts the job queue processes that are needed for the propagation and notification jobs.

AQ Background Architecture

Oracle Database Advanced Queuing introduces a new AQ background architecture with a 3-tier design.

- Tier1 (AQPC): Asingle background process called the Advanced Queuing Process Coordinator is created at instance startup. It will be responsible for creating and managing various primary processes. The coordinator statistics can be viewed using `GV$AQ_BACKGROUND_COORDINATOR`.
- Tier2 (QM**): There will be many primary processes named Queue Monitors. Each will be responsible for handling a distinct type of job. Jobs could be of type notification(Emon pool), queue monitors (handling TxEventQ time manager etc) , cross process etc.

  > **Note:**
  >
  > The old processes like QMNC and EMNC will be subsumed within one of new primary processes.

  A job can be defined as a type of work which needs own scheduling mechanism across multiple server processes (Q***) to perform its task . The primary process statistics and their jobs can be viewed using `GV$AQ_JOB_COORDINATOR`.
- Tier3(Q***): There will be a single pool of server processes for all above mentioned primary processes. Each process will be associated to a single primary process at a time. But can be rescheduled to another once original primary relinquishes its need to use it. These servers will perform jobs for respective primary processes providing performance and scalability. The server process statistics and its current primary association can be viewed using `GV$AQ_SERVER_POOL`.

3 Oracle Transactional Event Queues and Advanced Queuing: Programmatic Interfaces

These topics describe the various language options and elements you must work with and issues to consider in preparing your Oracle Database Advanced Queuing (AQ) application environment.

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

Topics:

- Programmatic Interfaces for Accessing Oracle Database Advanced Queuing
- Using PL/SQL to Access Oracle Database Advanced Queuing
- Using OCI to Access Oracle Database Advanced Queuing
- Using OCCI to Access Oracle Database Advanced Queuing
- Using Oracle Java Message Service (Oracle JMS) to Access Oracle Database Advanced Queuing
- Using Oracle Database Advanced Queuing XML Servlet to Access Oracle Database Advanced Queuing
- Comparing Oracle Database Advanced Queuing Programmatic Interfaces

Programmatic Interfaces for Accessing Oracle Database Advanced Queuing

The table lists Oracle Database Advanced Queuing programmatic interfaces, functions supported in each interface, and syntax references.

Table 3-1 Oracle Database Advanced Queuing Programmatic Interfaces

Language	Precompiler or Interface Program	Functions Supported	Syntax References
PL/SQL	`DBMS_AQADM` and `DBMS_AQ` Packages	Administrative and operational	*Oracle Database PL/SQL Packages and Types Reference*

Table 3-1 (Cont.) Oracle Database Advanced Queuing Programmatic Interfaces

Language	Precompiler or Interface Program	Functions Supported	Syntax References
C, Python, Javascript, Node.js	Oracle Call Interface (OCI)	Operational only	• *Oracle Call Interface Programmer's Guide* • *https://cx-oracle.readthedocs.io/en/latest/user_guide/aq.html* • https://oracle.github.io/node-oracledb/doc/api.html#aq
Java (JMS)	`oracle.JMS` package using JDBC API	Administrative and operational	*Oracle Database Advanced Queuing Java API Reference*
AQ XML servlet	Internet Data Access Presentation (IDAP)	Operational only	*Oracle XML DB Developer's Guide*

Using PL/SQL to Access Oracle Database Advanced Queuing

The PL/SQL packages `DBMS_AQADM` and `DBMS_AQ` support access to Oracle Database Advanced Queuing administrative and operational functions using the native Oracle Database Advanced Queuing interface.

These functions include:

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

- Add or remove aliases to Oracle Database Advanced Queuing objects in a LDAP server

Available PL/SQL `DBMS_AQADM` and `DBMS_AQ` functions are listed in detail in Table 3–2 through Table 3–9.

See Also:

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

Using OCI and the Thin JDBC Driver to Access Oracle Database Advanced Queuing

An Oracle Call Interface (OCI) provides an interface to Oracle Database Advanced Queuing functions using the native Oracle Database Advanced Queuing interface.

The OCI interface is used to access AQ in C, Python, and Node.js. See Table 3-1 for links to Python and Node.js documentation.

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`. Transactional event queues (TxEventQ) do not support OCI clients.

Oracle Type Translator

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

See Also:

"OCI and Advanced Queuing" and "Publish-Subscribe Notification" in *Oracle Call Interface Programmer's Guide* for syntax details

Python and Node.js programming interfaces for Advanced Queuing

The OCI thin driver client is used to develop the Python and Node.js implementations for operations on AQ. These calls are documented in a separate manual as shown in the links in Table 3-1 .

Comparing Oracle Database Advanced Queuing Programmatic Interfaces

These topics list and compare the Oracle Database Advanced Queuing Administrative Interfaces and the Oracle Database Advanced Queuing Operational Interfaces.

Available functions for the Oracle Database Advanced Queuing programmatic interfaces are listed by use case in Table 3-2 through Table 3-9. Use cases are described in Oracle Database Advanced Queuing Administrative Interface through Oracle Database Advanced Queuing Operations Using PL/SQL and Oracle Java Message Service Basic Operations through Oracle Java Message Service Shared Interfaces.

Oracle Transactional Event Queues and Advanced Queuing Administrative Interfaces

The table lists the equivalent Oracle Transactional Event Queues(TxEventQ) and Advanced Queuing(AQ) administrative functions for the PL/SQL and Java (JMS) programmatic interfaces.

Table 3-2 Comparison of Oracle Transactional Event Queues and Advanced Queuing Programmatic Interfaces: Administrative Interface

Use Case	PL/SQL	Java (JMS)
Create a transactional event queue	`DBMS_AQADM.CREATE_TRANSACTIONAL_EVENT_QUEUE`	
Drop a transactional event queue	`DBMS_AQADM.DROP_TRANSACTIONAL_EVENT_QUEUE`	
Alter a transactional event queue	`DBMS_AQADM.ALTER_TRANSACTIONAL_EVENT_QUEUE`	
Create a connection factory	N/A	`AQjmsFactory.getQueueConnectionFactory` `AQjmsFactory.getTopicConnectionFactory`
Register a ConnectionFactory in an LDAP server	N/A	`AQjmsFactory.registerConnectionFactory`
Create a queue table	`DBMS_AQADM.CREATE_QUEUE_TABLE`	`AQjmsSession.createQueueTable`
Get a queue table	Use *`schema.queue_table_name`*	`AQjmsSession.getQueueTable`
Alter a queue table	`DBMS_AQADM.ALTER_QUEUE_TABLE`	`AQQueueTable.alter`
Drop a queue table	`DBMS_AQADM.DROP_QUEUE_TABLE`	`AQQueueTable.drop`

Table 3-2 (Cont.) Comparison of Oracle Transactional Event Queues and Advanced Queuing Programmatic Interfaces: Administrative Interface

Use Case	PL/SQL	Java (JMS)
Create a queue	`DBMS_AQADM.CREATE_QUEUE`	`AQjmsSession.createQueue`
Get a queue	Use *`schema.queue_name`*	`AQjmsSession.getQueue`
Create a multiconsumer queue/topic in a queue table with multiple consumers enabled	`DBMS_AQADM.CREATE_QUEUE`	`AQjmsSession.createTopic`
Get a multiconsumer queue/topic	Use *`schema.queue_name`*	`AQjmsSession.getTopic`
Alter a queue/topic	`DBMS_AQADM.ALTER_QUEUE`	`AQjmsDestination.alter`
Start a queue/topic	`DBMS_AQADM.START_QUEUE`	`AQjmsDestination.start`
Stop a queue/topic	`DBMS_AQADM.STOP_QUEUE`	`AQjmsDestination.stop`
Drop a queue/topic	`DBMS_AQADM.DROP_QUEUE`	`AQjmsDestination.drop`
Grant system privileges	`DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE`	`AQjmsSession.grantSystemPrivilege`
Revoke system privileges	`DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE`	`AQjmsSession.revokeSystemPrivilege`
Grant a queue/topic privilege	`DBMS_AQADM.GRANT_QUEUE_PRIVILEGE`	`AQjmsDestination.grantQueuePrivilege` `AQjmsDestination.grantTopicPrivilege`
Revoke a queue/topic privilege	`DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE`	`AQjmsDestination.revokeQueuePrivilege` `AQjmsDestination.revokeTopicPrivilege`
Verify a queue type	`DBMS_AQADM.VERIFY_QUEUE_TYPES`	Not supported
Add a subscriber	`DBMS_AQADM.ADD_SUBSCRIBER`	See Table 3-6

Table 3-2 (Cont.) Comparison of Oracle Transactional Event Queues and Advanced Queuing Programmatic Interfaces: Administrative Interface

Use Case	PL/SQL	Java (JMS)
Alter a subscriber	`DBMS_AQADM.ALTER_SUBSCRIBER`	See Table 3-6
Remove a subscriber	`DBMS_AQADM.REMOVE_SUBSCRIBER`	See Table 3-6
Schedule propagation	`DBMS_AQADM.SCHEDULE_PROPAGATION`	`AQjmsDestination.schedulePropagation`
Enable a propagation schedule	`DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE`	`AQjmsDestination.enablePropagationSchedule`
Alter a propagation schedule	`DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE`	`AQjmsDestination.alterPropagationSchedule`
Disable a propagation schedule	`DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE`	`AQjmsDestination.disablePropagationSchedule`
Unschedule a propagation	`DBMS_AQADM.UNSCHEDULE_PROPAGATION`	`AQjmsDestination.unschedulePropagation`
Create an Oracle Database Advanced Queuing Internet Agent	`DBMS_AQADM.CREATE_AQ_AGENT`	Not supported
Alter an Oracle Database Advanced Queuing Internet Agent	`DBMS_AQADM.ALTER_AQ_AGENT`	Not supported
Drop an Oracle Database Advanced Queuing Internet Agent	`DBMS_AQADM.DROP_AQ_AGENT`	Not supported
Grant database user privileges to an Oracle Database Advanced Queuing Internet Agent	`DBMS_AQADM.ENABLE_AQ_AGENT`	Not supported
Revoke database user privileges from an Oracle Database Advanced Queuing Internet Agent	`DBMS_AQADM.DISABLE_AQ_AGENT`	Not supported
Add alias for queue, agent, ConnectionFactory in a LDAP server	`DBMS_AQADM.ADD_ALIAS_TO_LDAP`	Not supported
Delete alias for queue, agent, ConnectionFactory in a LDAP server	`DBMS_AQADM.DEL_ALIAS_FROM_LDAP`	Not supported

Oracle Database Advanced Queuing Operational Interfaces

These tables list equivalent Oracle Database Advanced Queuing operational functions for the programmatic interfaces PL/SQL, OCI, Oracle Database Advanced Queuing XML Servlet, and JMS, for various use cases.

Table 3-3 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Create Connection, Session, Message Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Create a connection	N/A	`OCIServer Attach`	Open an HTTP connection after authenticating with the Web server	`AQjmsQueueConnectionFactory .createQueueConnection`
				
				`AQjmsTopicConnectionFactory .createTopicConnection`

Create a session	N/A	`OCISession Begin`	An HTTP servlet session is automatically started with the first SOAP request	`QueueConnection.createQueue Session`
				
				`TopicConnection.createTopic Session`

Create a RAW message	Use SQL RAW type for message	Use OCIRaw for Message	Supply the hex representation of the message payload in the XML message. For example, `<raw>023f4523</raw>`	Not supported

Create a message with structured data	Use SQL Oracle object type for message	Use SQL Oracle object type for message	For Oracle object type queues that are not JMS queues (that is, they are not type `AQ$_JMS_*`), the XML specified in `<message payload>` must map to the SQL type of the payload for the queue table.
			For JMS queues, the XML specified in the `<message_payload>` must be one of the following: `<jms_text_message>`, `<jms_map_message>`, `<jms_bytes_message>`, `<jms_object_message>`
				`Session.createTextMessage`
				`Session.createObjectMessage`
				`Session.createMapMessage`
				`Session.createBytesMessage`
				`Session.createStreamMessage`
				`AQjmsSession.createAdtMessage`

Create a message producer	N/A	N/A	N/A	`QueueSession.createSender` `TopicSession.createPublisher`

Table 3-4 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Enqueue a message to a single-consumer queue	`DBMS_AQ.enqueue`	`OCIAQEnq`	`<AQXmlSend>`	`QueueSender.send`

Table 3-4 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Enqueue a message to a queue and specify visibility options	`DBMS_AQ.enqueue`
	
	Specify visibility in
	
	`ENQUEUE_OPTIONS`
		`OCIAQEnq`
		
		Specify `OCI_ATTR_VISIBILITY` in `OCIAQEnqOptions`
		
		`OCIAQEnqOptions`
			`<AQXmlSend>`
			
			Specify `<visibility>` in
			
			`<producer_options>`
				Not supported

Enqueue a message to a single-consumer queue and specify message properties priority and expiration	`DBMS_AQ.enqueue`
	
	Specify priority, expiration in
	
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		
		Specify `OCI_ATTR_PRIORITY`, `OCI_ATTR_EXPIRATION` in
		
		`OCIAQMsgProperties`
			`<AQXmlSend>`
			
			Specify `<priority>`, `<expiration>` in
			
			`<message_header>`
				Specify `priority` and `TimeToLive` during
				
				`QueueSender.send`
				
				or
				
				`.setTimeToLive`
				
				and
				
				`MessageProducer.setPriority`
				
				followed by
				
				`QueueSender.send`

Enqueue a message to a single-consumer queue and specify message properties correlationID, delay, and exception queue	`DBMS_AQ.enqueue`
	
	Specify correlation, delay, exception_queue in
	
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		
		Specify `OCI_ATTR_CORRELATION`, `OCI_ATTR_DELAY`, `OCI_ATTR_EXCEPTION_QUEUE` in
		
		`OCIAQMsgProperties`
			`<AQXmlSend>`
			
			Specify `<correlation_id>`, `<delay>`, `<exception_queue>` in
			
			`<message_header>`
				`Message.setJMSCorrelationI`
				
				Delay and exception queue specified as provider specific message properties
				
				`JMS_OracleDelay`
				`JMS_OracleExcpQ`
				
				followed by
				
				`QueueSender.send`

Table 3-4 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Enqueue Messages to a Single-Consumer Queue, Point-to-Point Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Enqueue a message to a single-consumer queue and specify user-defined message properties	Not supported
	Properties should be part of payload
		Not supported
		Properties should be part of payload
			`<AQXmlSend>`
			Specify `<name>` and `<int_value>`, `<string_value>`, `<long_value>`, and so on in
			`<user_properties>`
				`Message.setInt Property`
				`Message.setString Property`
				`Message.setBoolean Property`
				and so forth, followed by
				`QueueSender.send`

Enqueue a message to a single-consumer queue and specify message transformation	`DBMS_AQ.enqueue`
	Specify transformation in
	`ENQUEUE_OPTIONS`
		`OCIAQEnq`
		Specify `OCI_ATTR_TRANSFORMATION` in
		`OCIAQEnqOptions`
			`<AQXmlSend>`
			Specify `<transformation>` in
			`<producer_options>`
				`AQjmsQueueSender. setTransformation`
				followed by
				`QueueSender.send`

Table 3-5 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Publish a message to a multiconsumer queue/topic using default subscription list	`DBMS_AQ.enqueue`
	Set `recipient_list` to `NULL` in
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		Set `OCI_ATTR_RECIPIENT_LIST` to `NULL` in
		`OCIAQMsgProperties`
			`<AQXmlPublish>`	`TopicPublisher. publish`

Publish a message to a multiconsumer queue/topic using specific recipient list
See footnote-1
	`DBMS_AQ.enqueue`
	Specify recipient list in
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		Specify `OCI_ATTR_RECIPIENT_LIST` in
		`OCIAQMsgProperties`
			`<AQXmlPublish>`
			Specify `<recipient_list>` in
			`<message_header>`
				`AQjmsTopic Publisher.publish`
				Specify recipients as an array of `AQjmsAgent`

Table 3-5 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Publish a message to a multiconsumer queue/topic and specify message properties priority and expiration	`DBMS_AQ.enqueue`
	
	Specify priority, expiration in
	
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		
		Specify `OCI_ATTR_PRIORITY`, `OCI_ATTR_EXPIRATIO N` in
		
		`OCIAQMsgProperties`
			`<AQXmlPublish>`
			
			Specify `<priority>`, `<expiration>` in
			
			`<message_header>`
				Specify `priority` and `TimeToLive` during
				
				`TopicPublisher. publish`
				
				or
				
				`MessageProducer. setTimeToLive`
				
				and
				
				`MessageProducer. setPriority`
				
				followed by
				
				`TopicPublisher. publish`

Publish a message to a multiconsumer queue/topic and specify send options correlationID, delay, and exception queue	`DBMS_AQ.enqueue`
	
	Specify correlation, delay, `exception_queue` in
	
	`MESSAGE_PROPERTIES`
		`OCIAQEnq`
		
		Specify `OCI_ATTR_CORRELATI ON`, `OCI_ATTR_DELAY`, `OCI_ATTR_EXCEPTION _QUEUE` in
		
		`OCIAQMsgProperties`
			`<AQXmlPublish>`
			
			Specify `<correlation_id>`, `<delay>`, `<exception_queue>` in
			
			`<message_header>`
				`Message.setJMS CorrelationID`
				
				Delay and exception queue specified as provider-specific message properties
				
				`JMS_OracleDelay JMS_OracleExcpQ`
				
				followed by
				
				`TopicPublisher. publish`

Publish a message to a topic and specify user-defined message properties	Not supported
	
	Properties should be part of payload
		Not supported
		
		Properties should be part of payload
			`<AQXmlPublish>`
			
			Specify `<name>` and `<int_value>`, `<string_value>`, `<long_value>`, and so on in
			
			`<user_properties>`
				`Message.setInt Property`
				
				`Message.setString Property`
				
				`Message.setBoolean Property`
				
				and so forth, followed by
				
				`TopicPublisher. publish`

Table 3-5 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Publish Messages to a Multiconsumer Queue/Topic, Publish/Subscribe Model Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Publish a message to a topic and specify message transformation	`DBMS_AQ.enqueue`
	Specify transformation in
	`ENQUEUE_OPTIONS`
		`OCIAQEnq`
		Specify `OCI_ATTR_TRANSFORMATION` in
		`OCIAQEnqOptions`
			`<AQXmlPublish>`
			Specify `<transformation>` in
			`<producer_options>`
				`AQjmsTopicPublisher.setTransformation`
				followed by
				`TopicPublisher.publish`

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

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Add a subscriber	See administrative interfaces	Not supported	Not supported	`TopicSession.createDurableSubscriber` `AQjmsSession.createDurableSubscriber`

Alter a subscriber	See administrative interfaces	Not supported	Not supported	`TopicSession.createDurableSubscriber`
				`AQjmsSession.createDurableSubscriber`
				using the new selector

Remove a subscriber	See administrative interfaces	Not supported	Not supported	`AQjmsSession.unsubscribe`

Table 3-7 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Browse Messages in a Queue Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Browse messages in a queue/topic	`DBMS_AQ.dequeue`
	
	Set `dequeue_mode` to `BROWSE` in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Set `OCI_ATTR_DEQ_MODE` to `BROWSE` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<dequeue_mode>` `BROWSE` in
			
			`<consumer_options>`
				`QueueSession.createBrowser`
				
				`QueueBrowser.getEnumeration`
				
				Not supported on topics
				
				`oracle.jms.AQjmsSession.createBrowser`
				
				`oracle.jms.TopicBrowser.getEnumeration`

Browse messages in a queue/topic and lock messages while browsing	`DBMS_AQ.dequeue`
	
	Set `dequeue_mode` to `LOCKED` in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Set `OCI_ATTR_DEQ_MODE` to `LOCKED` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<dequeue_mode>` `LOCKED` in
			
			`<consumer_options>`
				`AQjmsSession.createBrowser`
				
				set `locked` to `TRUE`.
				
				`QueueBrowser.getEnumeration`
				
				Not supported on topics
				
				`oracle.jms.AQjmsSession.createBrowser`
				
				`oracle.jms.TopicBrowser.getEnumeration`

Table 3-8 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Receive Messages from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Start a connection for receiving messages	N/A	N/A	N/A	`Connection.start`

Create a message consumer	N/A	N/A	N/A	`QueueSession.createQueueReceiver`
				
				`TopicSession.createDurableSubscriber`
				
				`AQjmsSession.createTopicReceiver`

Dequeue a message from a queue/topic and specify visibility	`DBMS_AQ.dequeue`
	
	Specify visibility in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Specify `OCI_ATTR_VISIBILITY` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<visibility>` in
			
			`<consumer_options>`
				Not supported

Table 3-8 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Receive Messages from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Dequeue a message from a queue/topic and specify transformation	`DBMS_AQ.dequeue`
	
	Specify transformation in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Specify `OCI_ATTR_TRANSFORMATION` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<transformation>` in
			
			`<consumer_options>`
				`AQjmsQueueReceiver.setTransformation`
				
				`AQjmsTopicSubscriber.setTransformation`
				
				`AQjmsTopicReceiver.setTransformation`

Dequeue a message from a queue/topic and specify navigation mode	`DBMS_AQ.dequeue`
	
	Specify navigation in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Specify `OCI_ATTR_NAVIGATION` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<navigation>` in
			
			`<consumer_options>`
				`AQjmsQueueReceiver.setNavigationMode`
				
				`AQjmsTopicSubscriber.setNavigationMode`
				
				`AQjmsTopicReceiver.setNavigationMode`

Dequeue a message from a single-consumer queue	`DBMS_AQ.dequeue`
	
	Set `dequeue_mode` to `REMOVE` in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Set `OCI_ATTR_DEQ_MODE` to `REMOVE` in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`	`QueueReceiver.receive`
				
				or
				
				`QueueReceiver.receiveNoWait`
				
				or
				
				`AQjmsQueueReceiver.receiveNoData`

Dequeue a message from a multiconsumer queue/topic using subscription name	`DBMS_AQ.dequeue`
	
	Set `dequeue_mode` to `REMOVE` and set `consumer_name` to subscription name in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Set `OCI_ATTR_DEQ_MODE` to `REMOVE` and set `OCI_ATTR_CONSUMER_NAME` to subscription name in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<consumer_name>` in
			
			`<consumer_options>`
				Create a durable TopicSubscriber on the topic using the subscription name, then
				
				`TopicSubscriber.receive`
				
				or
				
				`TopicSubscriber.receiveNoWait`
				
				or
				
				`AQjmsTopicSubscriber.receiveNoData`

Table 3-8 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Receive Messages from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Dequeue a message from a multiconsumer queue/topic using recipient name	`DBMS_AQ.dequeue`
	
	Set `dequeue_mode` to `REMOVE` and set `consumer_name` to recipient name in
	
	`DEQUEUE_OPTIONS`
		`OCIAQDeq`
		
		Set `OCI_ATTR_DEQ_MODE` to `REMOVE` and set `OCI_ATTR_CONSUMER_NAME` to recipient name in
		
		`OCIAQDeqOptions`
			`<AQXmlReceive>`
			
			Specify `<consumer_name>` in
			
			`<consumer_options>`
				Create a `TopicReceiver` on the topic using the recipient name, then
				
				`AQjmsSession.createTopicReceiver`
				
				`AQjmsTopicReceiver.receive`
				
				or
				
				`AQjmsTopicReceiver.receiveNoWait`
				
				or
				
				`AQjmsTopicReceiver.receiveNoData`

Table 3-9 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Register to Receive Messages Asynchronously from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS

Receive messages asynchronously from a single-consumer queue	Define a PL/SQL callback procedure
	
	Register it using
	
	`DBMS_AQ.REGISTER`
		`OCISubscriptionRegister`
		
		Specify `queue_name` as subscription name
		
		`OCISubscriptionEnable`
			`<AQXmlRegister>`
			
			Specify queue name in `<destination>` and notification mechanism in
			
			`<notify_url>`
				Create a `QueueReceiver` on the queue, then
				
				`QueueReceiver.setMessageListener`

Receive messages asynchronously from a multiconsumer queue/ topic	Define a PL/SQL callback procedure
	
	Register it using
	
	`DBMS_AQ.REGISTER`
		`OCISubscriptionRegister`
		
		Specify `queue:OCI_ATTR_CONSUMER_NAME` as subscription name
		
		`OCISubscriptionEnable`
			`<AQXmlRegister>`
			
			Specify queue name in `<destination>`, consumer in `<consumer_name>` and notification mechanism in `<notify_url>`
				Create a `TopicSubscriber` or `TopicReceiver` on the topic, then
				
				`TopicSubscriber.setMessageListener`

Listen for messages on multiple queues/topics	-	-	-	-

Table 3-9 (Cont.) Comparison of Oracle Database Advanced Queuing Programmatic Interfaces: Operational Interface—Register to Receive Messages Asynchronously from a Queue/Topic Use Cases

Use Case	PL/SQL	OCI	AQ XML Servlet	JMS
Listen for messages on one (many) single-consumer queues	`DBMS_AQ.LISTEN` Use `agent_name` as `NULL` for all agents in `agent_list`	`OCIAQListen` Use `agent_name` as `NULL` for all agents in `agent_list`	Not supported	Create multiple `QueueReceivers` on a `QueueSession`, then `QueueSession.setMessageListener`
Listen for messages on one (many) multiconsumer queues/ Topics	`DBMS_AQ.LISTEN` Specify `agent_name` for all agents in `agent_list`	`OCIAQListen` Specify `agent_name` for all agents in `agent_list`	Not supported	Create multiple `TopicSubscribers` or `TopicReceivers` on a `TopicSession`, then `TopicSession.setMessageListener`

Using OCCI to Access Oracle Database Advanced Queuing

C++ applications can use OCCI, which has a set of Oracle Database Advanced Queuing interfaces that enable messaging clients to access Oracle Database Advanced Queuing.

OCCI AQ supports all the operational functions required to send/receive and publish/subscribe messages in a message-enabled database. Synchronous and asynchronous message consumption is available, based on a message selection rule. Transactional event queues (TxEventQ) do not support OCCI clients.

See Also:

"Oracle Database Advanced Queuing" in *Oracle C++ Call Interface Programmer's Guide*

Using Oracle Java Message Service (Oracle JMS) to Access Oracle Database Advanced Queuing

Java Message Service (JMS) is a messaging standard defined by Sun Microsystems, Oracle, IBM, and other vendors. JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product. Oracle Java Message Service (Oracle JMS) provides a Java API for Oracle Database Advanced Queuing based on the JMS standard.

Oracle Java Message Service (Oracle JMS) supports the standard JMS interfaces and has extensions to support administrative operations and other features that are not a part of the standard.

Standard Java Message Service(JMS) features include:

- Point-to-point model of communication using queues

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

Oracle JMS extensions include:

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

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

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

See Also:

– *Java Message Service Specification*, version 1.1, March 18, 2002, Sun Microsystems, Inc.

– *Oracle Database Advanced Queuing Java API Reference*

Accessing Standard and Oracle JMS Applications

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

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

Using JDBC OCI Driver or JDBC Thin Driver

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

Note that the Oracle Database does not support JDK 1.2, JDK 1.3, JDK 1.4, JDK5.n and all classes12*.* files. You need to use the `ojdbc6.jar` and `ojbc7.jar` files with JDK 6.n and JDK 7.n, respectively. The following jar and zip files should be in the `CLASSPATH` based on the release of JDK you are using.

For JDK 1.5.x, the `CLASSPATH` must contain:

```
ORACLE_HOME/jdbc/lib/ojdbc6.jar
```

For JDK 1.6.x, the `CLASSPATH` must contain:

```
ORACLE_HOME/jdbc/lib/ojdbc7.jar
```

The following files are used for either JDK version:

```
ORACLE_HOME/lib/jta.jar
ORACLE_HOME/xdk/lib/xmlparserv2.jar
ORACLE_HOME/rdbms/jlib/xdb.jar
ORACLE_HOME/rdbms/jlib/aqapi.jar
ORACLE_HOME/rdbms/jlib/jmscommon.jar
```

Using Oracle Server Driver in JServer

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

Using Oracle Database Advanced Queuing XML Servlet to Access Oracle Database Advanced Queuing

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

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

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

4

Managing Oracle Transactional Event Queues and Advanced Queuing

These topics discuss how to manage Oracle Transactional Event Queues and Advanced Queuing.

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

Oracle Database Advanced Queuing Compatibility Parameters

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

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

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

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

Queue Security and Access Control

These topics discuss Oracle Database Advanced Queuing queue security and access control.

- Oracle Database Advanced Queuing Security
- Queue Security
- Queue Privileges and Access Control

- OCI Applications and Queue Access
- Security Required for Propagation

Oracle Database Advanced Queuing Security

Configuration information can be managed through procedures in the `DBMS_AQADM` package.

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

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

Topics:

- Administrator Role
- User Role
- Access to Oracle Database Advanced Queuing Object Types

See Also:

"Granting Oracle Database Advanced Queuing System Privileges" for more information on AQ system privileges

Administrator Role

The `AQ_ADMINISTRATOR_ROLE` has all the required privileges to administer queues.

The privileges granted to the role let the grantee:

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

Note:

- A user does not need AQ_ADMINISTRATOR_ROLE to be able to create queues or do any other AQ DDL in his schema. User only needs execute privilege on DBMS_AQADM package for the same.
- AQ_ADMINISTRATOR_ROLE still will not be able to create AQ objects in SYS or SYSTEM schema. So the above description that it can create queue tables/queue on any schema excludes SYS and SYSTEM schema.
- AQ_ADMINISTRATOR_ROLE still will not be able to enqueue/dequeue on queues in SYS or SYSTEM schema.

User Role

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

Your database administrator has the option of granting the system privileges ENQUEUE_ANY and DEQUEUE_ANY, exercising DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE and DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE directly to a database user, if you want the user to have this level of control.

You as the application developer give rights to a queue by granting and revoking privileges at the object level by exercising DBMS_AQADM.GRANT_QUEUE_PRIVILEGE and DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE.

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

Note:

- A user only needs EXECUTE privilege on DBMS_AQ package to be able to enqueue or dequeue in his schema, if client is OCI or PL/SQL.

  If the client is JDBC or JMS, then added to this you need execute privilege on DBMS_AQIN package as well.
- A user does not need AQ_USER_ROLE for enqueue or dequeue queues in his schema.

Access to Oracle Database Advanced Queuing Object Types

All internal Oracle Database Advanced Queuing objects are accessible to PUBLIC.

Queue Security

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

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

Queue Privileges and Access Control

You can grant or revoke privileges at the object level on queues. You can also grant or revoke various system-level privileges.

Table 4-1 lists all common Oracle Database Advanced Queuing operations and the privileges needed to perform these operations.

Table 4-1 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.
ENQUEUE/ DEQUEUE to another's queues	Must be granted EXECUTE rights on DBMS_AQ and be granted privileges by the owner using DBMS_AQADM.GRANT_QUEUE_PRIVILEGE.
ENQUEUE/ DEQUEUE to any queues	Must be granted EXECUTE rights on DBMS_AQ and be granted ENQUEUE ANY QUEUE or DEQUEUE ANY QUEUE system privileges by an Oracle Database Advanced Queuing administrator using DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE.

OCI Applications and Queue Access

For an Oracle Call Interface (OCI) application to access a queue, the session user must be granted either the object privilege of the queue he intends to access or the ENQUEUE ANY QUEUE or DEQUEUE ANY QUEUE system privileges.

The EXECUTE right of DBMS_AQ is not checked against the session user's rights.

Security Required for Propagation

Oracle Database Advanced Queuing propagates messages through database links.

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

See Also:

"Propagation from Object Queues"

Security Required for AQ Buffered Messages on Oracle RAC

Internally, buffered queues on Oracle RAC may use `dblinks` between instances. Definer's rights packages that enqueue or dequeue into buffered queues on Oracle RAC must grant `INHERIT REMOTE PRIVILEGES` to users of the package.

Queue Table Export/Import

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

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

Note:

You cannot export or import buffered messages.

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

Topics:

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

Exporting Queue Table Data

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

Exporting Queue Tables with Multiple Recipients

For AQ queues, a queue table that supports multiple recipients is associated with the following tables:

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

- A history IOT

Transactional event queues are associated with the following objects:

- A queue table
- A dequeue table
- A time management table
- An optional exception queue map table
- Indexes for the above tables
- Sequences
- Rules sets and evaluation contexts

These tables are exported automatically during full database mode, user mode and table mode exports. See Export Modes .

Because the metadata tables contain ROWIDs of some rows in the queue table, the import process generates a note about the ROWIDs being made obsolete when importing the metadata tables. This message can be ignored, because the queuing system automatically corrects the obsolete ROWIDs as a part of the import operation. However, if another problem is encountered while doing the import (such as running out of rollback segment space), then you should correct the problem and repeat the import.

Export Modes

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

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

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

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

- `AQ$_`*`queue_table`*`_I` (the dequeue IOT)
- `AQ$_`*`queue_table`*`_T` (the time-management IOT)
- `AQ$_`*`queue_table`*`_S` (the subscriber table)
- `AQ$_`*`queue_table`*`_H` (the history IOT)

For transactional event queues, the following tables are automatically exported:

- *`queue_table`*
- `AQ$_`*`queue_name`*`_L` (dequeue table)
- `AQ$_`*`queue_name`*`_T` (time-management table)
- `AQ$_`*`queue_name`* (exception map table)
- `AQ$_`*`queue_name`*`_V` (evaluation context)
- *`queue_name`*`_R` (rule set)

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

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

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

Transactional event queues are associated with the following objects:

- A queue table
- A dequeue log table
- A time management table
- An optional exception queue map table
- Indexes for the above tables
- Sequences
- Rules sets and evaluation contexts

These objects must be imported along with the queue table itself.

Import IGNORE Parameter

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

Data Pump Export and Import

The Data Pump replace and skip modes are supported for queue tables.

In the replace mode an existing queue table is dropped and replaced by the new queue table from the export dump file. In the skip mode, a queue table that already exists is not imported.

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

See Also:

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

Oracle Enterprise Manager Support for AQ Queues

Oracle Enterprise Manager supports most of the administrative functions of Oracle Database Advanced Queuing. Oracle Database Advanced Queuing functions are found under the Distributed node in the navigation tree of the Enterprise Manager console.

Functions available through Oracle Enterprise Manager include:

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

Using Oracle Database Advanced Queuing with XA

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

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

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

See Also:

- "Working with Transaction Monitors with Oracle XA" in *Oracle Database Development Guide* for more information on XA
- "Large Objects (LOBs)" in *Pro*C/C++ Programmer's Guide*

Restrictions on Queue Management

These topics discuss restrictions on queue management.

- Subscribers

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

Note:

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

Subscribers

You cannot have more than 1,000 local subscribers for each queue.

Also, only 32 remote subscribers are allowed for each remote destination database.

DML Not Supported on Queue Tables or Associated IOTs

Oracle Database Advanced Queuing does not support data manipulation language (DML) operations on queue tables or associated index-organized tables (IOTs), if any.

The only supported means of modifying queue tables is through the supplied APIs. Queue tables and IOTs can become inconsistent and therefore effectively ruined, if data manipulation language (DML) operations are performed on them.

Propagation from Object Queues with REF Payload Attributes

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

Collection Types in Message Payloads

You cannot construct a message payload using a VARRAY that is not itself contained within an object.

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

Example 4-1   Creating Objects Containing VARRAYs

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

Synonyms on Queue Tables and Queues

No Oracle Database Advanced Queuing PL/SQL calls resolve synonyms on queues and queue tables.

Although you can create synonyms, you should not apply them to the Oracle Database Advanced Queuing interface.

Synonyms on Object Types

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

Tablespace Point-in-Time Recovery

Oracle Database Advanced Queuing currently does not support tablespace point-in-time recovery.

Creating a queue table in a tablespace disables that particular tablespace for point-in-time recovery. Oracle Database Advanced Queuing does support regular point-in-time recovery.

Virtual Private Database

You can use Oracle Database Advanced Queuing with Virtual Private Database by specifying a security policy with Oracle Database Advanced Queuing queue tables.

While dequeuing, use the dequeue condition (`deq_cond`) or the correlation identifier for the policy to be applied. You can use "1=1" as the dequeue condition. If you do not use a dequeue condition or correlation ID, then the dequeue results in an error.

Note:

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

Managing Propagation

These topics discuss managing Oracle Database Advanced Queuing propagation.

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

Note:

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

EXECUTE Privileges Required for Propagation

Propagation jobs are owned by `SYS`, but the propagation occurs in the security context of the queue table owner.

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

Propagation from Object Queues

Propagation from object queues with `BFILE` objects is supported.

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

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

See Also:

"CREATE DIRECTORY" in *Oracle Database SQL Language Reference* for more information on directory objects

Optimizing Propagation

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

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

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

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

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

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

The scheduling algorithm uses the job queue processes as follows (for this discussion, an active schedule is one that has a valid current window):

- If the number of active schedules is fewer than half the number of job queue processes, then the number of job queue processes acquired corresponds to the number of active schedules.
- If the number of active schedules is more than half the number of job queue processes, after acquiring half the number of job queue processes, then multiple active schedules are assigned to an acquired job queue process.
- If the system is overloaded (all schedules are busy propagating), depending on availability, then additional job queue processes are acquired up to one fewer than the total number of job queue processes.
- If none of the active schedules handled by a process has messages to be propagated, then that job queue process is released.
- The algorithm performs automatic load balancing by transferring schedules from a heavily loaded process to a lightly load process such that no process is excessively loaded.

Handling Failures in Propagation

The scheduling algorithm has robust support for handling failures. These are the common failures that prevent message propagation.

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

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

When an error occurs in a schedule, propagation of messages in that schedule is attempted again after a retry period that is a function of the number of failures. After the retries have exceeded a system defined maximum, the schedule is disabled.

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

See Also:

Troubleshooting Oracle Database Advanced Queuing

5

Kafka APIs for Oracle Transactional Event Queues

Oracle Transactional Event Queue (TxEventQ) makes it easy to implement event-based applications. It is also highly integrated with Apache Kafka, an open-source stream-processing software platform developed by LinkedIn and donated to the Apache Software Foundation, written in Scala and Java. Apart from enabling applications that use Kafka APIs to transparently operate on Oracle TxEventQ, Oracle TxEventQ also supports bi-directional information flow between TxEventQ and Kafka, so that changes are available in TxEventQ or Kafka as soon as possible in near-real-time.

Apache Kafka Connect is a framework included in Apache Kafka that integrates Kafka with other systems. Oracle TxEventQ will provide standard JMS package and related JDBC, Transaction packages to establish the connection and complete the transactional data flow. Oracle TxEventQ configures standard Kafka JMS connectors to establish interoperability and complete the data flow between the two messaging systems.

This chapter includes the following topics:

Apache Kafka Overview

Apache Kafka is a community distributed event streaming platform that is horizontally-scalable and fault-tolerant.

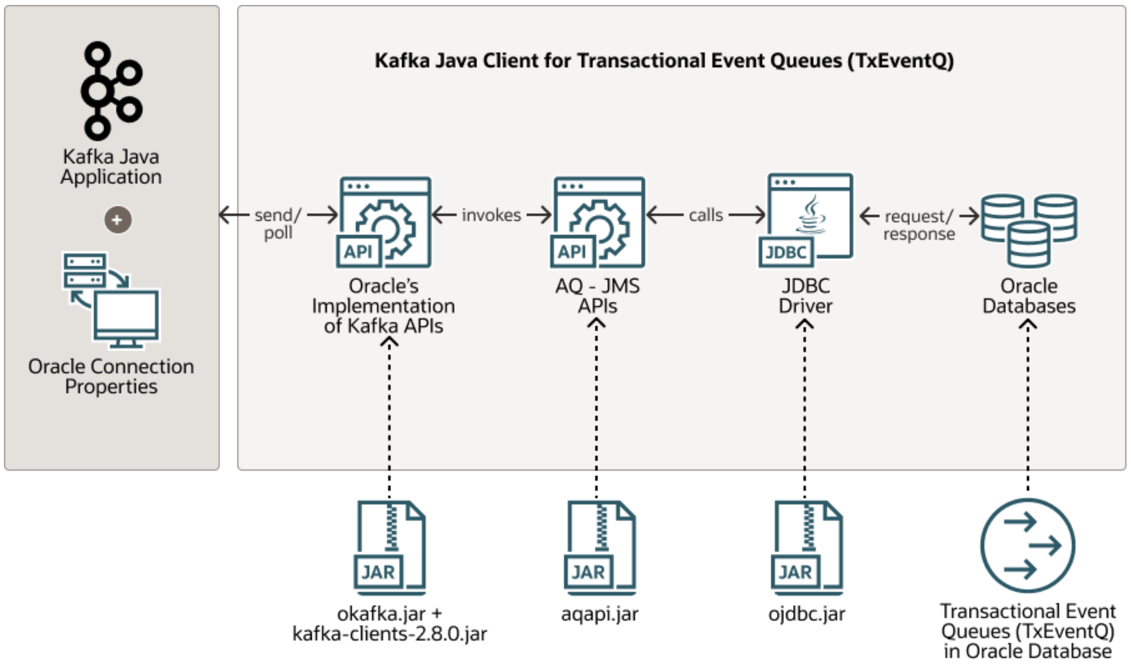
Kafka is deployed on a cluster of one or more servers. Each Kafka cluster stores streams of records in categories called topics. Each record consists of a key, a value, and a timestamp. The Kafka APIs allow an application to connect to a Kafka cluster and use the Kafka messaging platform.

Kafka Java Client for Transactional Event Queues

Oracle Database 21c introduced Kafka application compatibility with the Oracle Database. Oracle Database 23ai provides more refined compatibility for a Kafka application with the Oracle Database. This provides easy migration for Kafka Java applications to Transactional

Event Queues (TxEventQ). The Kafka Java APIs can now connect to an Oracle database server and use TxEventQ as a messaging platform.

Figure 5-1 Kafka Application Integration with Transactional Event Queue


Kafka Java Application
Oracle Connection Properties
Kafka Java Client for Transactional Event Queues (TxEventQ)
send/ poll
invokes
calls
request/ response
API
Oracle's Implementation of Kafka APIs
API
AQ - JMS APIs
JDBC
JDBC Driver
Oracle Databases
JAR
okafka.jar + kafka-clients-2.8.0.jar
JAR
aqapi.jar
JAR
ojdbc.jar
Transactional Event Queues (TxEventQ) in Oracle Database

This figure shows the Kafka API library, which contains Oracle specific implementation of Kafka's Java APIs which depends on the `kafka-clients-2.8.0.jar` file. This implementation internally invokes AQ-JMS APIs which in turn use the JDBC driver to communicate with the Oracle Database.

Developers can now migrate an existing Java application that uses Kafka to the Oracle database using `okafka.jar`. This client side library allows Kafka applications to connect to the Oracle Database instead of a Kafka cluster and use TxEventQ's messaging platform transparently.

Configuring Kafka Java Client for Transactional Event Queues

Prerequisites

The following are the prerequisites for configuring and running the Kafka Java client for TxEventQ in an Oracle Database.

1. Create a database user.
2. Grant the following privileges to the user.

> **Note:**
>
> It is preferred in general to assign or grant a specific quota on a tablespace to a database user instead of granting unlimited quota in default tablespace. One can create a table space and use the following command to grant quota on a specific tablespace to a database user.
>
> ```
> ALTER USER user QUOTA  UNLIMITED /* or size-clause */ on tablespace_name
> ```

- `GRANT EXECUTE on DBMS_AQ to` *`user`*.
- `GRANT EXECUTE on DBMS_AQADM to` *`user`*.
- `GRANT SELECT on GV_$SESSION to` *`user`*`;`
- `GRANT SELECT on V_$SESSION to` *`user`*`;`
- `GRANT SELECT on GV_$INSTANCE to` *`user`*`;`
- `GRANT SELECT on GV_$LISTENER_NETWORK to` *`user`*`;`
- `GRANT SELECT on GV_$PDBS to` *`user`*`;`
- `GRANT SELECT on USER_QUEUE_PARTITION_ASSIGNMENT_TABLE to` *`user`*`;`
- `exec DBMS_AQADM.GRANT_PRIV_FOR_RM_PLAN('`*`user`*`');`

3. Set the correct database configuration parameter to use TxEventQ.

   ```
   SET STREAMS_POOL_SIZE=400M
   ```

   > **Note:**
   >
   > Set the size appropriately based on your workload. `STREAMS_POOL_SIZE` cannot be set for Autonomous Database Shared. It is automatically configured. If set, then it is ignored.

4. Set `LOCAL_LISTENER` database parameter

   ```
   SET LOCAL_LISTENER= (ADDRESS=(PROTOCOL=TCP)(HOST=<HOST_NAME.DOMAIN_NAME/ IP> )
   (PORT=<PORT NUMBER>))
   ```

Connection Configuration

The Kafka API library connects to the Oracle Database using the JDBC Thin Driver. To set up this connection, the Kafka application can provide username and password in plain text, or applications can configure SSL. To run a Kafka application against Oracle Autonomous Database (ADB) on OCI, only SSL configuration is supported. For other deployments, you can connect to the Oracle Database using PLAINTEXT or SSL.

- PLAINTEXT: In this security protocol, JDBC connections to an Oracle Database are set up using the TCP protocol with username and password provided in plaintext in the `ojdbc.properties` file.

  To use the PLAINTEXT protocol, the application must set the following properties:

  – `oracle.service.name = <`*`name of the service running on the instance`*`>`
  – `bootstrap.servers = <`*`host`*`:`*`port`*`>`

  - `security.protocol`=PLAINTEXT
  - `oracle.net.tns_admin` = <*location of ojdbc.properties file*>

  The following properties should be present in the `ojdbc.properties` file:

  - `user = <`*nameofdatabaseuser*`>`
  - `password = <`*userpassword*`>`

- SSL: To use the SSL security protocol to connect to an ATP database, perform the following additional steps.

  1. JDBC Thin Driver Connection prerequisites for SSL security:
     - JDK8u162 or higher.
     - `oraclepki.jar`, `osdt_cert.jar`, and `osdt_core.jar`
     - 18.3 or higher JDBC Thin Driver
  2. To leverage the JDBC SSL security to connect to the Oracle Database instance, the user has to provide the following properties. JDBC supports SSL secured connections to Oracle Database in two ways.
     - Using wallets. To use wallets:
       a. Add the required dependant jars for using Oracle Wallets in classpath.

          Download `oraclepki.jar`, `osdt_cert.jar`, and `osdt_core.jar` files along with the JDBC Thin Driver, and add these jars to classpath.

       b. Enable Oracle PKI provider

          Add `OraclePKIProvider` at the end of the file `java.security` (located at `$JRE_HOME/jre/lib/security/java.security`) if SSO wallet, that is, `cwallet.sso` is used for providing SSL security. For example:

```
                  security.provider.1=sun.security.provider.Sun
                  security.provider.2=sun.security.rsa.SunRsaSign

security.provider.3=com.sun.net.ssl.internal.ssl.Provider
                  security.provider.4=com.sun.crypto.provider.SunJCE
                  security.provider.5=sun.security.jgss.SunProvider
                  security.provider.6=com.sun.security.sasl.Provider

security.provider.7=oracle.security.pki.OraclePKIProvider
```

          To use `ewallet.p12` for SSL security then place `OraclePKIProvider` before sun provider in file `java.security`. For example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=oracle.security.pki.OraclePKIProvider
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
security.provider.7=com.sun.security.sasl.Provider
```

       c. Set the following properties in the application.

```
security.protocol=SSL
oracle.net.tns_admin=<location of tnsnames.ora file>
```

```
tns.alias=<alias of connection string in tnsnames.ora>
```

Set the following properties in the `ojdbc.properties` file. This file must be available at the location specified by the `oracle.net.tns_admin` property.

```
                user(in smallletters)=nameofdatabaseuser
                password(in smallletters)=userpassword
                oracle.net.ssl_server_dn_match=true
                oracle.net.wallet_location="(SOURCE=(METHOD=FILE)
                                            (METHOD_DATA=(DIRECTORY=/
location../wallet_dbname)))"
```

– To use JDBC SSL security with a Java KeyStore, provide the following properties in the application:

```
security.protocol=SSL
oracle.net.tns_admin=<location of tnsnames.ora file>
tns.alias=<alias of connection string in tnsnames.ora>
```

Set the following properties in the `ojdbc.properties` file. This file must be available at the location specified by the `oracle.net.tns_admin` property.

```
user(in smallletters)=nameofdatabaseuser
password(in smallletters)=userpassword
oracle.net.ssl_server_dn_match=true
javax.net.ssl.trustStore==${TNS_ADMIN}/truststore.jks
javax.net.ssl.trustStorePassword=password
javax.net.ssl.keyStore=${TNS_ADMIN}/keystore.jks
javax.net.ssl.keyStorePassword=password
```

Note:

`tnsnames.ora` file in wallet downloaded from ATP contains a JDBC connection string which is used for establishing a JDBC connection.

See Also:

ALTER USER in *Oracle Database SQL Language Reference*

Kafka Client Interfaces

Kafka applications mainly use Producer, Consumer, and Admin APIs to communicate with a Kafka cluster. This version of Kafka client for TxEventQ supports only a subset of Apache Kafka 2.8.0's Producer, Consumer and Admin APIs and properties. With the `okafka.jar` client library, Kafka applications will be able to use the Oracle TxEventQ platform. The `okafka.jar` library requires JRE 9 or above.

We first illustrate the use of the Kafka Client APIs by way of simple examples, and later describe more details on the same.

- Kafka API Examples
- Overview of Kafka Producer Implementation for TxEventQ
- Overview of Kafka Consumer implementation for TxEventQ
- Overview of Kafka Admin Implementation for TxEventQ

Kafka API Examples

Example: Creating an Oracle Kafka Topic

```
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.admin.Admin;
import org.apache.kafka.clients.admin.CreateTopicsResult;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.common.KafkaFuture;

import org.oracle.okafka.clients.admin.AdminClient;

public class SimpleAdminOKafka {

    public static void main(String[] args) {
        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's 
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");

        // location for ojdbc.properties file where user and password properties are 
saved
        props.put("oracle.net.tns_admin",".");

        try (Admin admin = AdminClient.create(props)) {
            //Create Topic named TEQ with 10 Partitions.
            CreateTopicsResult result = admin.createTopics(
                    Arrays.asList(new NewTopic("TEQ", 10, (short)0)));
            try {
                KafkaFuture<Void> ftr =  result.all();
                ftr.get();
            } catch ( InterruptedException | ExecutionException e ) {

                throw new IllegalStateException(e);
            }
            System.out.println("Closing  OKafka admin now");
        }
        catch(Exception e)
        {
            System.out.println("Exception while creating topic " + e);
            e.printStackTrace();
        }
    }
}
```

Example: Kafka Consumer API

```
import java.util.Properties;
import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.common.header.Header;
import org.apache.kafka.common.TopicPartition;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.oracle.okafka.clients.consumer.KafkaConsumer;

public class SimpleConsumerOKafka {

    // Dummy implementation of ConsumerRebalanceListener interface
    // It only maintains the list of assigned partitions in assignedPartitions list
    static class ConsumerRebalance implements ConsumerRebalanceListener {

        public List<TopicPartition> assignedPartitions = new ArrayList<>();

        @Override
        public synchronized void onPartitionsAssigned(Collection<TopicPartition> 
partitions) {
            System.out.println("Newly Assigned Partitions:");
            for (TopicPartition tp :partitions ) {
                System.out.println(tp);
                assignedPartitions.add(tp);
            }
        }

        @Override
        public synchronized void onPartitionsRevoked(Collection<TopicPartition> 
partitions) {
            System.out.println("Revoked previously assigned partitions. ");
            for (TopicPartition tp :assignedPartitions ) {
                System.out.println(tp);
            }
            assignedPartitions.clear();
        }
    }

    public static void main(String[] args) {
        //System.setProperty("org.slf4j.simpleLogger.defaultLogLevel", "TRACE");
        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's 
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");
        // location for ojdbc.properties file where user and password properties are
saved
        props.put("oracle.net.tns_admin",".");

        //Consumer Group Name
        props.put("group.id" , "CG1");
        props.put("enable.auto.commit","false");

        // Maximum number of records fetched in single poll call
        props.put("max.poll.records", 2000);

        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

        Consumer<String , String> consumer = new KafkaConsumer<String,
String>(props);
        ConsumerRebalanceListener rebalanceListener = new ConsumerRebalance();

        //Subscribe to a single topic named 'TEQ'.
        consumer.subscribe(Arrays.asList("TEQ"), rebalanceListener);

        int expectedMsgCnt = 40000;
        int msgCnt = 0;
        Instant startTime = Instant.now();
        try {
            while(true) {
                try {
                    //Consumes records from the assigned partitions of 'TEQ' topic
                    ConsumerRecords <String, String> records =
consumer.poll(Duration.ofMillis(10000));
                    //Print consumed records
                    for (ConsumerRecord<String, String> record : records)
                    {
                        System.out.printf("partition = %d, offset = %d, key = %s, value
=%s\n ", record.partition(), record.offset(), record.key(), record.value());
                        for(Header h: record.headers())
                        {
                            System.out.println("Header: " +h.toString());
                        }
                    }
                    //Commit all the consumed records
                    if(records != null && records.count() > 0) {
                        msgCnt += records.count();
                        System.out.println("Committing records " + records.count());
                        try {
                            consumer.commitSync();
                        }catch(Exception e)
                        {
                            System.out.println("Exception in commit " + e.getMessage());
                            continue;
                        }
                        if(msgCnt >= expectedMsgCnt )
                        {
                            System.out.println("Received " + msgCnt + " Expected " +
expectedMsgCnt +". Exiting Now.");
                            break;
                        }
                    }
                    else {
                        System.out.println("No Record Fetched. Retrying in 1 second");
                        Thread.sleep(1000);
                    }

                }catch(Exception e)
                {
                    System.out.println("Inner Exception " + e.getMessage());
                    throw e;
                }
            }
        }catch(Exception e)
        {
            System.out.println("Exception from OKafka consumer " + e);
            e.printStackTrace();
        }finally {
            long runDuration = Duration.between(startTime, Instant.now()).toMillis();
            System.out.println("Closing OKafka Consumer. Received "+ msgCnt +" records.
Run Duration " + runDuration);
            consumer.close();
        }
    }
}
```

Example: Kafka Producer API

```
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.header.internals.RecordHeader;

import org.oracle.okafka.clients.producer.KafkaProducer;

import java.time.Duration;
import java.time.Instant;
import java.util.Properties;
import java.util.concurrent.Future;

public class SimpleProducerOKafka {
    public static void main(String[] args) {
        try {
            Properties props = new Properties();
            //IP or Host name where Oracle Database 23ai is running and Database
Listener's Port
            props.put("bootstrap.servers", "localhost:1521");

            //name of the service running on the database instance
            props.put("oracle.service.name", "freepdb1");
            props.put("security.protocol","PLAINTEXT");

            // location for ojdbc.properties file where user and password properties are
saved
            props.put("oracle.net.tns_admin",".");


            props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
            props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

            String baseMsg = "This is a test message ";
            // Creates OKafka Producer
            Producer<String, String> producer = new KafkaProducer<String, String>(props);

            Future<RecordMetadata> lastFuture = null;
            int msgCnt = 40000;
            Instant startTime = Instant.now();

            //Headers, common for all records
            RecordHeader rH1 = new RecordHeader("CLIENT_ID", "FIRST_CLIENT".getBytes());
            RecordHeader rH2 = new RecordHeader("REPLY_TO", 
"REPLY_TOPIC_NAME".getBytes());

            //Produce 40000 messages into topic named "TEQ".
            for(int i=0;i<msgCnt;i++) {
                ProducerRecord<String, String> producerRecord = new 
ProducerRecord<String, String>("TEQ", ""+i, baseMsg + i);
                producerRecord.headers().add(rH1).add(rH2);
                lastFuture =producer.send(producerRecord);
            }
            //Waits until the last message is acknowledged
            lastFuture.get();
            long runTime = Duration.between( startTime, Instant.now()).toMillis();
            System.out.println("Produced "+ msgCnt +" messages. Run Duration " + 
runTime);
            //Closes the OKafka producer
            producer.close();
        }
        catch(Exception e)
        {
            System.out.println("Exception in Main " + e );
            e.printStackTrace();
        }
    }
}
```

Example: Deleting an Oracle Kafka Topic

```
import java.util.Collections;
import java.util.Properties;

import org.apache.kafka.clients.admin.Admin;

import org.oracle.okafka.clients.admin.AdminClient;

public class SimpleAdminDeleteTopic {

    public static void main(String[] args) {

        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's 
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");

        // location for ojdbc.properties file where user and password properties are 
saved
        props.put("oracle.net.tns_admin",".");
        try (Admin admin = AdminClient.create(props)) {
            //Throws Exception if failed to delete the topic. Returns null on successful
deletion.
            org.apache.kafka.clients.admin.DeleteTopicsResult delResult =
                    admin.deleteTopics(Collections.singletonList("TEQ"));
            Thread.sleep(1000);
            System.out.println("Closing admin now");
        }
        catch(Exception e)
        {
            System.out.println("Exception while creating topic " + e);
            e.printStackTrace();
        }
    }

}
```

Note:

- Topics created using the KafkaAdmin interface can be accessed only by KafkaProducer or KafkaConsumer interfaces.
- Similarly, the KafkaProducer and KafkaConsumer interfaces can produce or consume records only from topics which are created using the KafkaAdmin interface.

Kafka REST APIs for TxEventQ

The TxEventQ REST APIs allow common operations to produce and consume from topics and partitions, and are implemented using Oracle REST Data Services (ORDS) in the Oracle Database. Common operations include creating and deleting topics, producing and consuming messages, and operational APIs for getting consumer lag on a topic, seeking to an offset, among many others.

The following three APIs for Kafka allow TxEventQ to co-exist with Kafka deployments, and provide the advantages of transactional outbox, JMS messaging and pub/sub in the database and high throughput streaming of events to the event queue in the Oracle Database.

See Also:

Oracle Transactional Event Queues REST Endpoints for the Oracle REST Data Services API documentation

Overview of Kafka Producer Implementation for TxEventQ

Producer APIs allow a Kafka application to publish messages into Oracle Transactional Event Queues (TxEventQ). A Kafka application needs to provide Oracle specific properties: `bootstrap.servers`, `oracle.servicename`, and `oracle.net.tns_admin`. More details about these properties are mentioned in the configuration section. These properties are used to set up the database connection and produce the message into TxEventQ. In the current release, Oracle's implementation of `KafkaProducer` supports only a subset of the Producer APIs.

Internally, an Oracle Kafka Producer object encapsulates an AQ JMS producer object which is used to publish messages into Oracle TxEventQ. Similar to Apache Kafka Producer, each Producer `send()` call will append a Kafka Record into a batch based on its topic and partition. Based on Apache Kafka's internal algorithm, a background thread will publish the entire batch to an Oracle TxEventQ.

The following `KafkaProducer` APIs are supported in Oracle Database 23ai.

- Constructor:

  `KafkaProducer`: Creates a producer object and internal AQ JMS objects. The `KafkaProducer` class has four types of constructors defined, which all take configuration parameters as input.

- Methods:
  - `send(ProducerRecord) , send(ProducerRecord, Callback):`

    The `send` method asynchronously publishes a message into TxEventQ. This method returns immediately once a Kafka Record has been stored in the buffer of records waiting to be sent. If the buffer is full, then the send call blocks for a maximum time of `max.block.ms`. Records will be published into the topic using AQ JMS.

    The method returns a `Future<RecordMetadata>`, which contains the partition, offset, and publish timestamp of the record. Both the `send(ProducerRecord)` and `send(ProducerRecord, Callback)` versions are supported.

  - `close`: Closes the producer and frees the memory. It will also close the internal connection to the Oracle Database.

- Classes
  - `ProducerRecord`: A class that represents a message in the Kafka platform. The Kafka API library translates a `ProducerRecord` into a JMS BytesMessage for the TxEventQ platform.
  - `RecordMetadata`: This contains metadata of the record like topic, partition, offset, timestamp etc. of the Record in the Kafka platform. This is assigned values relevant for TxEventQs. A message id of TxEventQ is converted into an offset of `RecordMetadata`.
  - Callback Interface: A callback function which is executed once a Record is successfully published into a Kafka topic.
  - Partitioner Interface: Defines methods which map a `Key` of the message to a partition number of the topic. A partition number is analogous to a stream id of TxEventQs.

- Properties
  - `key.serializer` and `value.serializer`: Converts Key and payload into byte array respectively.
  - `acks`: For Kafka APIs, the only value relevant for the `acks` property is `all`. Any other field set by the user is ignored.
  - `linger.ms`: Time in miliseconds for which the sender thread waits before publishing the records in TxEventQ.
  - `batch.size`: Total size of records to be batched in bytes for which the sender thread waits before publishing records in TxEventQ.
  - `buffer.memory`: Total memory in bytes the accumulator can hold.

  - `max.block.ms`: If `buffer.memory` size is full in the accumulator, then wait for `max.block.ms` amount of time before the `send()` method can receive an out of memory error.
  - `retries`: This property enables a producer to resend a record in case of transient errors. This value limits the number of retries per batch.
  - `retry.backoff.ms` : The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.
  - `bootstrap.servers`: IP address and port of a machine where an instance of the database is running.

Overview of Kafka Consumer implementation for TxEventQ

The Consumer API allows applications to read streams of data from a Transactional Event Queue (TxEventQ). Kafka consumer for TxEventQ uses AQ JMS APIs and a JDBC driver to consume messages in batches from Oracle TxEventQ. For Oracle Kafka, consuming messages from a topic is achieved by dequeuing messages from a Transactional Event Queue.

Similar to Apache Kafka, in TxEventQ's implementation, a consumer group (subscriber) may contains many consumer instances (unique database sessions that are consuming for the subscriber). Each consumer group has a unique group-id (subscriber name). Each consumer instance internally maintains a single connection/session to an Oracle Database instance provided by the `bootstrap.servers` property. Oracle Database 23ai introduces Kafka API support for Consumer Group Rebalancing. Partitions of a topic will be distributed among the active consumers of a consumer group such that no two consumers from the same consumer group are assigned the same partition of the topic simultaneously. Whenever new consumers join a consumer group or an existing consumer leaves the group, the partitions will be redistributed among the active consumers.

For the 23ai release of Kafka APIs, a consumer can subscribe to only one topic.

The following `KafkaConsumer` APIs are supported in Oracle Database 23ai.

- Constructor: `KafkaConsumer`: Creates a consumer that allows the application to consume messages from a key based TxEventQ. Internal client side TxEventQ objects created are not visible to a client application. All variations of the `KafkaConsumer` constructor are supported in Oracle Database 23ai.
- Methods:
  - `Subscribe`: This method takes a list of topics to subscribe to. In Oracle Database 23ai, only the first topic of the list will be subscribed to. An exception is thrown if the size of the list is greater than 1. This method creates a durable subscriber on TxEventQ server side with Group-Id as subscriber name. An application can also implement the `ConsumerRebalanceListener` interface and pass an object of the implemented class to the subscribe method. This allows a consumer to execute callbacks when a partition is revoked or assigned.
  - `Poll`: The `poll` method returns a batch of messages from assigned partitions from TxEventQ. It attempts to dequeue a message from the key based TxEventQ for the subscriber. TxEventQ uses the array dequeue API of AQ JMS to receive a batch of messages from the queue. The size of the batch depends on the parameter `max.poll.records` set by the Kafka client application. `Poll` takes time in milliseconds as an argument. The AQ JMS API for array dequeue can pass this timeout as a

  dequeue option to the TxEventQ server and make the dequeue call, which will wait for messages till the timeout if the full array batch is not complete.

  When poll is invoked for the first time by a consumer, it triggers consumer rebalancing for all the alive consumers of the consumer group. At the end of the consumer rebalancing, all alive consumers are assigned topic partitions, and subsequent poll requests will fetch messages from the assigned partitions only.

  An application can participate and influence rebalancing using the `ConsumerRebalanceListener` interface and `partition.assignment.strategy` configuration.

  The `partition.assignment.strategy` configuration allows an application to select a strategy for assigning partitions to consumer streams. OKafka supports all values for this configuration parameter which are documented in the Apache Kafka 2.8.0 documentation.

  The default value for this configuration is `org.oracle.okafka.clients.consumer.TXEQAssignor` which is aware of Oracle RAC and implements a strategy that is best for achieving higher throughput from Oracle TxEventQ.

  This strategy prioritizes fair distribution of partitions and local consumption of messages while distributing partitions among alive sessions.

  `ConsumerRebalanceListener` allows an application to invoke callbacks when partitions are revoked or assigned to a consumer.

  The database view `USER_QUEUE_PARTITION_ASSIGNMENT_TABLE` allows a developer to view the current distribution of partitions among the alive consumers.

  – `commitSync`: Commits all consumed messages. Commit to an offset is not supported in Oracle Database 23ai. This call directly calls commit on the database which commits all consumed messages from TxEventQ.
  – `commitAsync`: This call is translated into `commitSync`. A callback function passed as an argument gets executed once the commit is successful.
  – `Unsubscribe`: Unsubscribes the topic that a consumer has subscribed to. A consumer can no longer consume messages from unsubscribed topics. This call does not remove a subscriber group from the TxEventQ metadata. Other consumer applications can still continue to consume for the same consumer group.
  – `close`: Closes the consumer and unsubscribes the topic it has subscribed to.

- Class: `ConsumerRecord`: A class that represents a consumed record in the Kafka platform. Kafka APIs receive AQ JMS messages from TxEventQ and convert each of them into a `ConsumerRecord` and deliver it to the application.
- Properties:
  – `auto.offset.reset`: When there is no initial offset found for this consumer group, then the value of this property controls whether to consume the messages from the beginning of the topic partition or to only consume new messages. Values for this property and its usage are as follows:
    * `earliest`: Consume from the beginning of the topic partition.
    * `latest`: Consume from the end of the topic partition (default).
    * `none`: Throw an exception if no offset present for the consumer group.
  – `key.deserializer` and `value.deserializer`: For Oracle Kafka messaging platform, key and value are stored as byte arrays in a JMS Message in Oracle's TxEventQ. On

consuming, these byte arrays are deserialized into a key and a value using `key.deserializer` and `value.deserializer` respectively. These properties allow an application to convert `Key` and `Value`, which are stored in byte array format, into application specified data types.

  - `group.id`: This is a consumer group name for which messages are consumed from the Kafka topic. This property is used as a durable subscriber name for key based TxEventQs.
  - `max.poll.records`: Maximum number of records to fetch in a single array dequeue call from an Oracle TxEventQ server.
  - `enable.auto.commit`: Enables auto commit of consumed messages for every specified interval.
  - `auto.commit.interval.ms`: Interval in milliseconds for auto commit of messages.
  - `bootstrap.servers`: IP address and port of a machine where a database instance is running.

Overview of Kafka Admin Implementation for TxEventQ

The Kafka admininistrative API allows applications to perform administrative tasks like creating a topic, deleting a topic, adding a partition to a topic and so on. Oracle Database 23ai release supports only the following administrative APIs.

- Methods
  - `create(props)` and `create(config)`: Creates an object of `KafkaAdmin` class that uses passed parameters. The method creates a database session which is used for further operations. An application has to provide connection configuration parameters as explained in the Connection Configuration section.
  - `createTopics()`: Allows an application to create a Kafka Topic. This creates a TxEventQ in the user's schema.
  - `close()`: Closes a database session and Admin client.
  - `deleteTopic`: Deletes a Kafka Topic. This returns `null` when a topic is deleted successfully. Otherwise, the method throws an exception. The method does not return until the topic is successfully deleted or any error is encountered.
- Classes: `NewTopic`: Class used for creating a new topic. This class contains parameters with which a transactional event queue is created.

Kafka REST APIs for TxEventQ

The TxEventQ REST APIs allow common operations to produce and consume from topics and partitions, and are implemented using Oracle REST Data Services (ORDS) in the Oracle Database. Common operations include creating and deleting topics, producing and consuming messages, and operational APIs for getting consumer lag on a topic, seeking to an offset, among many others.

The following three APIs for Kafka allow TxEventQ to co-exist with Kafka deployments, and provide the advantages of transactional outbox, JMS messaging and pub/sub in the database and high throughput streaming of events to the event queue in the Oracle Database.

See Also:

Oracle Transactional Event Queues REST Endpoints for the Oracle REST Data Services API documentation

Kafka Connectors for TxEventQ

The Kafka Sink and Source Connector requires a minimum Oracle Database version of 21c in order to create a Transactional Event Queue. To use the application, Kafka with a minimum version number of 3.1.0 will need to be downloaded and installed on a server.

See Also:

https://github.com/oracle/okafka/tree/master/connectors for more information.

Monitoring Message Transfer

The Sink/Source connector messages transfer can be monitored from Oracle TxEventQ.

See Also:

Monitoring Transactional Event Queues to startup TxEventQ Monitor System to check enqueue/dequeue rate, TxEventQ depth, and more DB/System Level statistics.

6

Java Message Service for Transactional Event Queues and Advanced Queuing

This chapter contains the following topics:

Java Messaging Service Interface for Oracle Transactional Event Queues and Advanced Queuing

The following topics describe the Oracle Java Message Service (JMS) interface to Oracle Database Advanced Queuing (AQ).

General Features of JMS and Oracle JMS

This section contains these topics:

- Retention and Message History in JMS
- Supporting Oracle Real Application Clusters in JMS
- Supporting Statistics Views in JMS

JMS Connection and Session

This section contains these topics:

- ConnectionFactory Objects
- Using AQjmsFactory to Obtain ConnectionFactory Objects
- Using JNDI to Look Up ConnectionFactory Objects
- JMS Connection
- JMS Session

ConnectionFactory Objects

A `ConnectionFactory` encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider. In this case Oracle JMS, part of Oracle Database, is the JMS provider.

The three types of `ConnectionFactory` objects are:

- `ConnectionFactory`
- `QueueConnectionFactory`
- `TopicConnectionFactory`

Using AQjmsFactory to Obtain ConnectionFactory Objects

You can use the `AQjmsFactory` class to obtain a handle to a `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory` object.

To obtain a `ConnectionFactory`, which supports both point-to-point and publish/subscribe operations, use `AQjmsFactory.getConnectionFactory()`. To obtain a `QueueConnectionFactory`, use `AQjmsFactory.getQueueConnectionFactory()`. To obtain a `TopicConnectionFactory`, use `AQjmsFactory.getTopicConnectionFactory()`.

The `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory` can be created using hostname, port number, and SID driver or by using JDBC URL and properties.

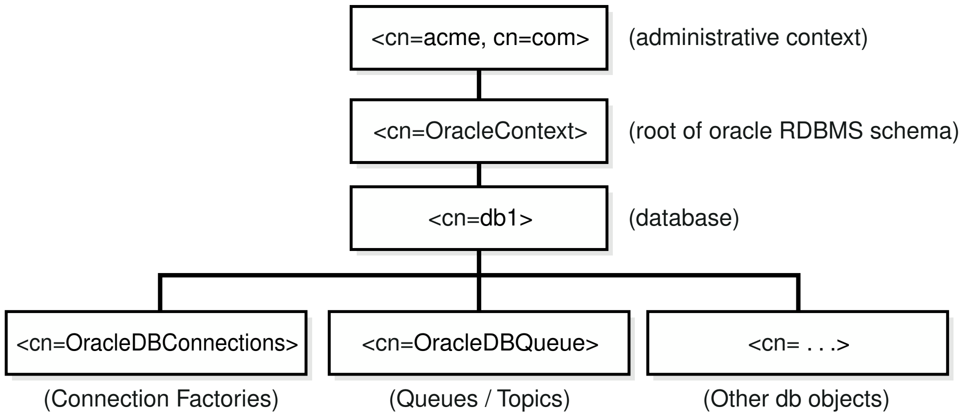
Using JNDI to Look Up ConnectionFactory Objects

A JMS administrator can register `ConnectionFactory` objects in a Lightweight Directory Access Protocol (LDAP) server. The following setup is required to enable Java Naming and Directory Interface (JNDI) lookup in JMS:

1. Register Database

   When the Oracle Database server is installed, the database must be registered with the LDAP server. This can be accomplished using the Database Configuration Assistant (DBCA). Figure 6-1 shows the structure of Oracle Database Advanced Queuing entries in the LDAP server. `ConnectionFactory` information is stored under `<cn=OracleDBConnections>`, while topics and queues are stored under `<cn=OracleDBQueues>`.

Figure 6-1 Structure of Oracle Database Advanced Queuing Entries in LDAP Server



2. Set Parameter `GLOBAL_TOPIC_ENABLED`.

   The `GLOBAL_TOPIC_ENABLED` system parameter for the database must be set to `TRUE`. This ensures that all queues and topics created in Oracle Database Advanced Queuing are automatically registered with the LDAP server. This parameter can be set by using `ALTER SYSTEM SET GLOBAL_TOPIC_ENABLED = TRUE`.

3. Register `ConnectionFactory` Objects

   After the database has been set up to use an LDAP server, the JMS administrator can register `ConnectionFactory`, `QueueConnectionFactory`, and `TopicConnectionFactory` objects in LDAP by using `AQjmsFactory.registerConnectionFactory()`.

   The registration can be accomplished in one of the following ways:

   - Connect directly to the LDAP server

     The user must have the `GLOBAL_AQ_USER_ROLE` to register connection factories in LDAP.

     To connect directly to LDAP, the parameters for the `registerConnectionFactory` method include the LDAP context, the name of the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory`, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

   - Connect to LDAP through the database server

     The user can log on to Oracle Database first and then have the database update the LDAP entry. The user that logs on to the database must have the `AQ_ADMINISTRATOR_ROLE` to perform this operation.

     To connect to LDAP through the database server, the parameters for the `registerConnectionFactory` method include a JDBC connection (to a user having `AQ_ADMINISTRATOR_ROLE`), the name of the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory`, hostname, database SID, port number, JDBC driver (thin or oci8) and factory type (queue or topic).

JMS Connection

A JMS `Connection` is an active connection between a client and its JMS provider. A JMS `Connection` performs several critical services:

- Encapsulates either an open connection or a pool of connections with a JMS provider
- Typically represents an open TCP/IP socket (or a set of open sockets) between a client and a provider's service daemon
- Provides a structure for authenticating clients at the time of its creation
- Creates `Sessions`
- Provides connection metadata
- Supports an optional `ExceptionListener`

A JMS `Connection` to the database can be created by invoking `createConnection()`, `createQueueConnection()`, or `createTopicConnection()` and passing the parameters `username` and `password` on the `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory` object respectively.

Some of the methods that are supported on the `Connection` object are

- `start()`

  This method starts or restart delivery of incoming messages.
- `stop()`

  This method temporarily stops delivery of incoming messages. When a `Connection` object is stopped, delivery to all of its message consumers is inhibited. Also, synchronous receive's block and messages are not delivered to message listener.
- `close()`

  This method closes the JMS session and releases all associated resources.
- `createSession(true, 0)`

  This method creates a JMS `Session` using a JMS `Connection` instance.
- `createQueueSession(true, 0)`

  This method creates a `QueueSession`.
- `createTopicSession(true, 0)`

  This method creates a `TopicSession`.
- `setExceptionListener(ExceptionListener)`

  This method sets an exception listener for the `Connection`. This allows a client to be notified of a problem asynchronously. If a `Connection` only consumes messages, then it has no other way to learn it has failed.
- `getExceptionListener()`

  This method gets the `ExceptionListener` for this `Connection`.

A JMS client typically creates a `Connection`, a `Session` and several `MessageProducer` and `MessageConsumer` objects. In the current version only one open `Session` for each `Connection` is allowed, except in the following cases:

- If the JDBC oci8 driver is used to create the JMS connection
- If the user provides an `OracleOCIConnectionPool` instance during JMS connection creation

When a `Connection` is created it is in stopped mode. In this state no messages can be delivered to it. It is typical to leave the `Connection` in stopped mode until setup is complete. At that point the `Connection start()` method is called and messages begin arriving at the `Connection` consumers. This setup convention minimizes any client confusion that can result from asynchronous message delivery while the client is still in the process of setup.

It is possible to start a `Connection` and to perform setup subsequently. Clients that do this must be prepared to handle asynchronous message delivery while they are still in the process of setting up. A `MessageProducer` can send messages while a `Connection` is stopped.

JMS Session

A JMS `Session` is a single threaded context for producing and consuming messages. Although it can allocate provider resources outside the Java Virtual Machine (JVM), it is considered a lightweight JMS object.

A `Session` serves several purposes:

- Constitutes a factory for `MessageProducer` and `MessageConsumer` objects
- Provides a way to get a handle to destination objects (queues/topics)
- Supplies provider-optimized message factories
- Supports a single series of transactions that combines work spanning session `MessageProducer` and `MessageConsumer` objects, organizing these into units
- Defines a serial order for the messages it consumes and the messages it produces
- Serializes execution of `MessageListener` objects registered with it

In Oracle Database 20c, you can create as many JMS `Sessions` as resources allow using a single JMS `Connection`, when using either JDBC thin or JDBC thick (OCI) drivers.

Because a provider can allocate some resources on behalf of a `Session` outside the JVM, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough. The same is true for `MessageProducer` and `MessageConsumer` objects created by a `Session`.

Methods on the `Session` object include:

- `commit()`

  This method commits all messages performed in the transaction and releases locks currently held.

- `rollback()`

  This method rolls back any messages accomplished in the transaction and release locks currently held.

- `close()`

  This method closes the `Session`.

- `getDBConnection()`

  This method gets a handle to the underlying JDBC connection. This handle can be used to perform other SQL DML operations as part of the same `Session`. The method is specific to Oracle JMS.

- `acknowledge()`

  This method acknowledges message receipt in a nontransactional session.

- `recover()`

This method restarts message delivery in a nontransactional session. In effect, the series of delivered messages in the session is reset to the point after the last acknowledged message.

The following are some Oracle JMS extensions:

- `createQueueTable()`

  This method creates a queue table.
- `getQueueTable()`

  This method gets a handle to an existing queue table.
- `createQueue()`

  This method creates a queue.
- `getQueue()`

  This method gets a handle to an existing queue.
- `createTopic()`

  This method creates a topic.
- `getTopic()`

  This method gets a handle to an existing topic.

The `Session` object must be cast to `AQjmsSession` to use any of the extensions.

Note:

The JMS specification expects providers to return null messages when receives are accomplished on a JMS `Connection` instance that has not been started.

After you create a `javax.jms.Connection` instance, you must call the `start()` method on it before you can receive messages. If you add a line like `t_conn.start();` any time after the connection has been created, but before the actual receive, then you can receive your messages.

JMS Destination

A `Destination` is an object a client uses to specify the destination where it sends messages, and the source from which it receives messages. A `Destination` object can be a `Queue` or a `Topic`. In Oracle Database Advanced Queuing, these map to a *`schema.queue`* at a specific database. `Queue` maps to a single-consumer queue, and `Topic` maps to a multiconsumer queue.

Using a JMS Session to Obtain Destination Objects

`Destination` objects are created from a `Session` object using the following domain-specific `Session` methods:

- `AQjmsSession.getQueue(queue_owner, queue_name)`

  This method gets a handle to a JMS queue.
- `AQjmsSession.getTopic(topic_owner, topic_name)`

This method gets a handle to a JMS topic.

Using JNDI to Look Up Destination Objects

The database can be configured to register schema objects with an LDAP server. If a database has been configured to use LDAP and the GLOBAL_TOPIC_ENABLED parameter has been set to TRUE, then all JMS queues and topics are automatically registered with the LDAP server when they are created. The administrator can also create aliases to the queues and topics registered in LDAP. Queues and topics that are registered in LDAP can be looked up through JNDI using the name or alias of the queue or topic.

See Also:

"Adding an Alias to the LDAP Server"

JMS Destination Methods

Methods on the `Destination` object include:

- `alter()`

  This method alters a `Queue` or a `Topic`.
- `schedulePropagation()`

  This method schedules propagation from a source to a destination.
- `unschedulePropagation()`

  This method unschedules a previously scheduled propagation.
- `enablePropagationSchedule()`

  This method enables a propagation schedule.
- `disablePropagationSchedule()`

  This method disables a propagation schedule.
- `start()`

  This method starts a `Queue` or a `Topic`. The queue can be started for enqueue or dequeue. The topic can be started for publish or subscribe.
- `stop()`

  This method stops a `Queue` or a `Topic`. The queue is stopped for enqueue or dequeue. The topic is stopped for publish or subscribe.
- `drop()`

  This method drops a `Queue` or a `Topic`.

System-Level Access Control in JMS

Oracle8*i* or higher supports system-level access control for all queuing operations. This feature allows an application designer or DBA to create users as queue administrators. A queue administrator can invoke administrative and operational JMS interfaces on any queue in the database. This simplifies administrative work, because all administrative scripts for the queues in a database can be managed under one schema.

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

To propagate to a remote destination queue, the login user (specified in the database link in the address field of the agent structure) should either be granted the `ENQUEUE_ANY` privilege, or be granted the rights to enqueue to the destination queue. However, you are not required to grant any explicit privileges if the login user in the database link also owns the queue tables at the destination.

See Also:

"Oracle Enterprise Manager Support"

Destination-Level Access Control in JMS

Oracle8*i* or higher supports access control for enqueue and dequeue operations at the queue or topic level. This feature allows the application designer to protect queues and topics created in one schema from applications running in other schemas. You can grant only minimal access privileges to the applications that run outside the schema of the queue or topic. The supported access privileges on a queue or topic are `ENQUEUE`, `DEQUEUE` and `ALL`.

See Also:

"Oracle Enterprise Manager Support"

Retention and Message History in JMS

Messages are often related to each other. For example, if a message is produced as a result of the consumption of another message, then the two are related. As the application designer, you may want to keep track of such relationships. Oracle Database Advanced Queuing allows users to retain messages in the queue table, which can then be queried in SQL for analysis.

Along with retention and message identifiers, Oracle Database Advanced Queuing lets you automatically create message journals, also called tracking journals or event journals. Taken together, retention, message identifiers and SQL queries make it possible to build powerful message warehouses.

Supporting Oracle Real Application Clusters in JMS

A transactional event queue (TxEventQ) is a single logical queue that is divided into multiple, independent, physical queues through system-maintained partitioning. TxEventQs are the preferred JMS queues for queues used across Oracle RAC instances, for queues with high enqueue or dequeue rates, or for queues with many subscribers. See "Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC)" for more information.

For AQ queues, Oracle Real Application Clusters (Oracle RAC) can be used to improve Oracle Database Advanced Queuing performance by allowing different queues to be managed by different instances. You do this by specifying different instance affinities (preferences) for the queue tables that store the queues. This allows queue operations (enqueue/dequeue) or topic operations (publish/subscribe) on different queues or topics to occur in parallel.

The Oracle Database Advanced Queuing queue monitor process continuously monitors the instance affinities of the queue tables. The queue monitor assigns ownership of a queue table to the specified primary instance if it is available, failing which it assigns it to the specified secondary instance.

If the owner instance of a queue table terminates, then the queue monitor changes ownership to a suitable instance such as the secondary instance.

Oracle Database Advanced Queuing propagation can 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 is handling the propagation from queues stored in that queue table, thereby minimizing pinging.

See Also:

- "Transactional Event Queues"
- "Scheduling a Queue Propagation"
- *Oracle Real Application Clusters Administration and Deployment Guide*

Supporting Statistics Views in JMS

Each instance keeps its own Oracle Database Advanced Queuing statistics information in its own System Global Area (SGA), and does not have knowledge of the statistics gathered by other instances. Then, when a `GV$AQ` view is queried by an instance, all other instances funnel their statistics information to the instance issuing the query.

The `GV$AQ` view can be queried at any time to see the number of messages in waiting, ready or expired state. The view also displays the average number of seconds messages have been waiting to be processed.

See Also:

"V$AQ: Number of Messages in Different States in Database"

Structured Payload/Message Types in JMS

JMS messages are composed of a header, properties, and a body.

The header consists of header fields, which contain values used by both clients and providers to identify and route messages. All messages support the same set of header fields.

Properties are optional header fields. In addition to standard properties defined by JMS, there can be provider-specific and application-specific properties.

The body is the message payload. JMS defines various types of message payloads, and a type that can store JMS messages of any or all JMS-specified message types.

This section contains these topics:

JMS Message Headers

A JMS message header contains the following fields:

- `JMSDestination`

  This field contains the destination to which the message is sent. In Oracle Database Advanced Queuing this corresponds to the destination queue/topic. It is a `Destination` type set by JMS after the `Send` method has completed.

- `JMSDeliveryMode`

  This field determines whether the message is logged or not. JMS supports `PERSISTENT` delivery (where messages are logged to stable storage) and `NONPERSISTENT` delivery (messages not logged). It is a `INTEGER` set by JMS after the `Send` method has completed. JMS permits an administrator to configure JMS to override the client-specified value for `JMSDeliveryMode`.

- `JMSMessageID`

  This field uniquely identifies a message in a provider. All message IDs must begin with the string `ID:`. It is a `String` type set by JMS after the `Send` method has completed.

- `JMSTimeStamp`

  This field contains the time the message was handed over to the provider to be sent. This maps to Oracle Database Advanced Queuing message enqueue time. It is a `Long` type set by JMS after the `Send` method has completed.

- `JMSCorrelationID`

  This field can be used by a client to link one message with another. It is a `String` type set by the JMS client.

- `JMSReplyTo`

  This field contains a `Destination` type supplied by a client when a message is sent. Clients can use `oracle.jms.AQjmsAgent`; `javax.jms.Queue`; or `javax.jms.Topic`.

- `JMSType`

  This field contains a message type identifier supplied by a client at send time. It is a `String` type. For portability Oracle recommends that the `JMSType` be symbolic values.

- `JMSExpiration`

  This field is the sum of the enqueue time and the `TimeToLive` in non-Java EE compliance mode. In compliant mode, the `JMSExpiration` header value in a dequeued message is the sum of `JMSTimeStamp` when the message was enqueued (Greenwich Mean Time, in milliseconds) and the `TimeToLive` (in milliseconds). It is a `Long` type set by JMS after the `Send` method has completed. JMS permits an administrator to configure JMS to override the client-specified value for `JMSExpiration`.

- `JMSPriority`

  This field contains the priority of the message. It is a `INTEGER` set by JMS after the `Send` method has completed. In Java EE-compliance mode, the permitted values for priority are `0`–`9`, with `9` the highest priority and `4` the default, in conformance with the Sun Microsystem JMS 1.1 standard. Noncompliant mode is the default. JMS permits an administrator to configure JMS to override the client-specified value for `JMSPriority`.

- `JMSRedelivered`

  This field is a Boolean set by the JMS provider.

See Also:

"Java EE Compliance"

JMS Message Properties

JMS properties are set either explicitly by the client or automatically by the JMS provider (these are generally read-only). Some JMS properties are set using the parameters specified in `Send` and `Receive` operations.

Properties add optional header fields to a message. Properties allow a client, using a `messageSelector`, to have a JMS provider select messages on its behalf using application-specific criteria. Property names are strings and values can be: `Boolean`, `byte`, `short`, `int`, `long`, `float`, `double`, and `string`.

JMS-defined properties, which all begin with `"JMSX"`, include the following:

- `JMSXUserID`

  This field is the identity of the user sending the message. It is a `String` type set by JMS after the `Send` method has completed.

- `JMSXAppID`

  This field is the identity of the application sending the message. It is a `String` type set by JMS after the `Send` method has completed.

- `JMSXDeliveryCount`

  This field is the number of message delivery attempts. It is an `Integer` set by JMS after the `Send` method has completed.

- `JMSXGroupid`

  This field is the identity of the message group that this message belongs to. It is a `String` type set by the JMS client.

- `JMSXGroupSeq`

  This field is the sequence number of a message within a group. It is an `Integer` set by the JMS client.

- `JMSXRcvTimeStamp`

  This field is the time the message was delivered to the consumer (dequeue time). It is a `String` type set by JMS after the `Receive` method has completed.

- `JMSXState`

This field is the message state, set by the provider. The message state can be `WAITING`, `READY`, `EXPIRED`, or `RETAINED`.

Oracle-specific JMS properties, which all begin with `JMS_Oracle`, include the following:

- `JMS_OracleExcpQ`

  This field is the queue name to send the message to if it cannot be delivered to the original destination. It is a `String` type set by the JMS client. Only destinations of type `EXCEPTION` can be specified in the `JMS_OracleExcpQ` property.
- `JMS_OracleDelay`

  This field is the time in seconds to delay the delivery of the message. It is an `Integer` set by the JMS client. This can affect the order of message delivery.
- `JMS_OracleOriginalMessageId`

  This field is set to the message identifier of the message in the source if the message is propagated from one destination to another. It is a `String` type set by the JMS provider. If the message is not propagated, then this property has the same value as `JMSMessageId`.

A client can add additional header fields to a message by defining properties. These properties can then be used in a `messageSelector` to select specific messages.

JMS Message Bodies

JMS provides five forms of message body:

- StreamMessage
- BytesMessage
- MapMessage
- TextMessage
- ObjectMessage
- AdtMessage

StreamMessage

A `StreamMessage` object is used to send a stream of Java primitives. It is filled and read sequentially. It inherits from `Message` and adds a `StreamMessage` body. Its methods are based largely on those found in `java.io.DataInputStream` and `java.io.DataOutputStream`.

The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects. To use `StreamMessage` objects, create the queue table with the `SYS.AQ$_JMS_STREAM_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

`StreamMessage` objects support the conversions shown in Table 6-1. A value written as the row type can be read as the column type.

Table 6-1 StreamMessage Conversion

Input	Boolean	byte	short	char	int	long	float	double	String	byte[]
`Boolean`	X	-	-	-	-	-	-	-	X	-
`byte`	-	X	X	-	X	X	-	-	X	-
`short`	-	-	X	-	X	X	-	-	X	-

Table 6-1 (Cont.) StreamMessage Conversion

Input	Boolean	byte	short	char	int	long	float	double	String	byte[]
`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

BytesMessage

A `BytesMessage` object is used to send a message containing a stream of uninterpreted bytes. It inherits `Message` and adds a `BytesMessage` body. The receiver of the message interprets the bytes. Its methods are based largely on those found in `java.io.DataInputStream` and `java.io.DataOutputStream`.

This message type is for client encoding of existing message formats. If possible, one of the other self-defining message types should be used instead.

The primitive types can be written explicitly using methods for each type. They can also be written generically as objects. To use `BytesMessage` objects, create the queue table with `SYS.AQ$_JMS_BYTES_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

MapMessage

A `MapMessage` object is used to send a set of name-value pairs where the names are `String` types, and the values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined. It inherits from `Message` and adds a `MapMessage` body. The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects.

To use `MapMessage` objects, create the queue table with the `SYS.AQ$_JMS_MAP_MESSAGE` or `AQ$_JMS_MESSAGE` payload types. `MapMessage` objects support the conversions shown in Table 6-2. An "X" in the table means that a value written as the row type can be read as the column type.

Table 6-2 MapMessage Conversion

Input	Boolean	byte	short	char	int	long	float	double	String	byte[]
`Boolean`	X	-	-	-	-	-	-	-	X	-
`byte`	-	X	X	-	X	X	-	-	X	-
`short`	-	-	X	-	X	X	-	-	X	-
`char`	-	-	-	X	-	-	-	-	X	-
`int`	-	-	-	-	X	X	-	-	X	-
`long`	-	-	-	-	-	X	-	-	X	-
`float`	-	-	-	-	-	-	X	X	X	-

Table 6-2 (Cont.) MapMessage Conversion

Input	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

TextMessage

A `TextMessage` object is used to send a message containing a `java.lang.StringBuffer`. It inherits from `Message` and adds a `TextMessage` body. The text information can be read or written using methods `getText()` and `setText(...)`. To use `TextMessage` objects, create the queue table with the `SYS.AQ$_JMS_TEXT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

ObjectMessage

An `ObjectMessage` object is used to send a message that contains a serializable Java object. It inherits from Message and adds a body containing a single Java reference. Only serializable Java objects can be used. If a collection of Java objects must be sent, then one of the collection classes provided in JDK 1.4 can be used. The objects can be read or written using the methods `getObject()` and `setObject(...)`.To use `ObjectMessage` objects, create the queue table with the `SYS.AQ$_JMS_OBJECT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

AdtMessage

An `AdtMessage` object is used to send a message that contains a Java object that maps to an Oracle object type. These objects inherit from `Message` and add a body containing a Java object that implements the `CustomDatum` or `ORAData` interface.

To use `AdtMessage` objects, create the queue table with payload type as the Oracle object type. The `AdtMessage` payload can be read and written using the `getAdtPayload` and `setAdtPayload` methods.

You can also use an `AdtMessage` object to send messages to queues of type `SYS.XMLType`. You must use the `oracle.xdb.XMLType` class to create the message.

For `AdtMessage` objects, the client can get:

- `JMSXDeliveryCount`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

See Also:

Oracle Database Java Developer's Guide for information about the `CustomDatum` and `ORAData` interfaces

Using Message Properties with Different Message Types

The following message properties can be set by the client using the `setProperty` call. For `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, and `MapMessage` objects, the client can set:

- `JMSXAppID`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

For `AdtMessage` objects, the client can set:

- `JMS_OracleExcpQ`
- `JMS_OracleDelay`

The following message properties can be obtained by the client using the `getProperty` call. For `StreamMessage`, `BytesMessage`, `ObjectMessage`, `TextMessage`, and `MapMessage` objects, the client can get:

- `JMSXuserID`
- `JMSXAppID`
- `JMSXDeliveryCount`
- `JMSXGroupID`
- `JMSXGroupSeq`
- `JMSXRecvTimeStamp`
- `JMSXState`
- `JMS_OracleExcpQ`
- `JMS_OracleDelay`
- `JMS_OracleOriginalMessageID`

Buffered Messaging with Oracle JMS

Users can send a nonpersistent JMS message by specifying the `deliveryMode` to be `NON_PERSISTENT` when sending a message. JMS nonpersistent messages are not required to be logged to stable storage, so they can be lost after a JMS system failure. JMS nonpersistent messages are similar to the buffered messages available in Oracle Database Advanced Queuing, but there are also important differences between the two.

Note:

Do not confuse Oracle JMS nonpersistent messages with Oracle Database Advanced Queuing nonpersistent queues, which are deprecated in Oracle Database 10*g* Release 2 (10.2).

See Also:

- "Buffered Messaging"
- Nonpersistent Queues

Transaction Commits and Client Acknowledgments

The JMS `deliveryMode` is orthogonal to the transaction attribute of a message. JMS nonpersistent messages can be sent and received by either a transacted session or a nontransacted session. If a JMS nonpersistent message is sent and received by a transacted session, then the effect of the JMS operation is only visible after the transacted session commits. If it is received by a nontransacted session with `CLIENT_ACKNOWLEDGE` acknowledgment mode, then the effect of receiving this message is only visible after the client acknowledges the message. Without the acknowledgment, the message is not removed and will be redelivered if the client calls `Session.recover`.

Oracle Database Advanced Queuing buffered messages, however, do not support these transaction or acknowledgment concepts. Both sending and receiving a buffered message must be in the `IMMEDIATE` visibility mode. The effects of the sending and receiving operations are therefore visible to the user immediately, no matter whether the session is committed or the messages are acknowledged.

Different APIs

Messages sent with the regular JMS send and publish methods are treated by Oracle Database Advanced Queuing as persistent messages. The regular JMS receive methods receive only AQ persistent messages. To send and receive buffered messages, you must use the Oracle extension APIs `bufferSend`, `bufferPublish`, and `bufferReceive`.

See Also:

Oracle Database Advanced Queuing Java API Reference for more information on `bufferSend`, `bufferPublish`, and `bufferReceive`

Payload Limits

The Oracle Database Advanced Queuing implementation of buffered messages does not support `LOB` attributes. This places limits on the payloads for the five types of standard JMS messages:

- JMS `TextMessage` payloads cannot exceed 4000 bytes.

  This limit might be even lower with some database character sets, because during the Oracle JMS character set conversion, Oracle JMS sometimes must make a conservative choice of using `CLOB` instead of `VARCHAR` to store the text payload in the database.
- JMS `BytesMessage` payloads cannot exceed 2000 bytes.
- JMS `ObjectMessage`, `StreamMessage`, and `MapMessage` data serialized by JAVA cannot exceed 2000 bytes.
- For all other Oracle JMS ADT messages, the corresponding Oracle database ADT cannot contain `LOB` attributes.

Different Constants

The Oracle Database Advanced Queuing and Oracle JMS APIs use different numerical values to designate buffered and persistent messages, as shown in Table 6-3.

Table 6-3 Oracle Database AQ and Oracle JMS Buffered Messaging Constants

API	Persistent Message	Buffered Message
Oracle Database Advanced Queuing	`PERSISTENT := 1`	`BUFFERED :=2`
Oracle JMS	`PERSISTENT := 2`	`NON_PERSISTENT := 1`

Buffered Messaging in JMS

Buffered messaging fully supports JMS messaging standards. Oracle JMS extends those standards in several ways.

See Also:

"Buffered Messaging"

Enqueuing JMS Buffered Messages

Oracle JMS allows applications to send buffered messages by setting `JMSDeliveryMode` for individual messages, so persistent and buffered messages can be enqueued to the same JMS queue/topic.

Oracle JMS buffered messages can be ordered by enqueue time, priority, or both. The ordering does not extend across message types. So a persistent message sent later, for example, can be delivered before an buffered message sent earlier. Expiration is also supported for buffered messages in Oracle JMS.

See Also:

"JMS Message Headers"

Dequeuing JMS Buffered Messages

JMS does not require subscribers to declare interest in just persistent messages or just buffered messages, so JMS subscribers can be interested in both message types.

Oracle JMS supports fast and efficient dequeue of messages by `JMSMessageID`, selectors on message headers, and selectors on message properties. The Oracle JMS dequeue call checks for both persistent and buffered messages.

Note:

Oracle JMS persistent messages have unique message identifiers. Oracle JMS buffered message identifiers are unique only within a queue/topic.

If concurrent dequeue processes are dequeuing from the same queue as the same subscriber, then they will skip messages that are locked by the other process.

See Also:

- "MessageSelector"
- "Receiving Messages "

Transactions Support

If buffered messages are enqueued in a transacted session, then JMS requires transaction support for them. Oracle JMS guarantees that transacted sessions involving buffered messages meet the following standards:

- Atomicity

  Both persistent and buffered messages within an Oracle JMS transaction are committed or rolled back atomically. Even if buffered messages were written to disk, as in the case of messages involving LOBs, rollback nevertheless removes them.
- Consistency

  If persistent and buffered messaging operations interleave in a transaction, then all Oracle JMS users share a consistent view of the affected queues/topics. All persistent and buffered messages enqueued by a transaction become visible at commit time. If a process ends in the middle of a transaction, then both persistent and buffered messages are undone. Oracle JMS users see either all persistent and buffered messages in a transaction or none of them.
- Isolation

  An buffered enqueue operation in a transaction is visible only to the owner transaction before the transaction is committed. It is visible to all consumers after the transaction is committed.

Messages locked by dequeue transaction may be browsed.

Acknowledging Message Receipt

Three values are defined for the `ack_mode` parameter for acknowledging message receipt in nontransacted sessions:

- `DUPS_OK_ACKNOWLEDGE`

  In this mode, duplicate messages are allowed.
- `AUTO_ACKNOWLEDGE`

  In this mode, the session automatically acknowledges messages.
- `CLIENT_ACKNOWLEDGE`

In this mode, the client explicitly acknowledges messages by calling the message producer acknowledge method. Acknowledging a message acknowledges all previously consumed messages.

> **See Also:**
>
> "Creating a Session"

Buffered Messaging Quality of Service

JMS requires providers to support at-most-once delivery of unpropagated buffered messages. If recovery of buffered messages is disabled, then Oracle JMS meets this standard.

Duplicate delivery of messages is possible with the current implementation of message propagation. But this does not violate the JMS standard, because message propagation is an extension offered by Oracle JMS.

> **See Also:**
>
> "Propagating Buffered Messages" for the causes of duplicate delivery of buffered messages

JMS Types Support for Buffered Messages

Oracle JMS maps the JMS-defined types to Oracle user-defined types and creates queues of these user-defined types for storing JMS messages. Some of these types have LOB attributes, which Oracle JMS writes to disk whether the message is persistent or buffered.

The user-defined type `SYS.AQ$_JMS_TEXT_MESSAGE` for JMS type `JMSTextMessage`, for example, stores text strings smaller than 4k in a `VARCHAR2` column. But it has a CLOB attribute for storing text strings larger than 4k.

Because JMS messages are often larger than 4k, Oracle JMS offers a new ADT that allows larger messages to be stored in memory. The disk representation of the ADT remains unchanged, but several `VARCHAR2`/`RAW` attributes allow for JMS messages of sizes up to 100k to be stored in memory. Messages larger than 100k can still be published as buffered messages, but they are written to disk.

> **See Also:**
>
> "Enqueuing Buffered Messages"

JMS Point-to-Point Model Features

In the point-to-point model, clients exchange messages from one point to another. Message producers and consumers send and receive messages using single-consumer queues. An administrator creates the single-consumer queues with the `createQueue` method in `AQjmsSession`. Before they can be used, the queues must be enabled for enqueue/dequeue using the `start` call in `AQjmsDestination`. Clients obtain a handle to a previously created queue using the `getQueue` method on `AQjmsSession`.

In a single-consumer queue, a message can be consumed exactly once by a single consumer. If there are multiple processes or operating system threads concurrently dequeuing from the same queue, then each process dequeues the first unlocked message at the head of the queue. A locked message cannot be dequeued by a process other than the one that has created the lock.

After processing, the message is removed if the retention time of the queue is 0, or it is retained for a specified retention time. As long as the message is retained, it can be either queried using SQL on the queue table view or dequeued by specifying the message identifier of the processed message in a `QueueBrowser`.

QueueSender

A client uses a `QueueSender` to send messages to a queue. It is created by passing a queue to the `createSender` method in a client `Session`. A client also has the option of creating a `QueueSender` without supplying a queue. In that case a queue must be specified on every send operation.

A client can specify a default delivery mode, priority and `TimeToLive` for all messages sent by the `QueueSender`. Alternatively, the client can define these options for each message.

QueueReceiver

A client uses a `QueueReceiver` to receive messages from a queue. It is created using the `createQueueReceiver` method in a client `Session`. It can be created with or without a `messageSelector`.

QueueBrowser

A client uses a `QueueBrowser` to view messages on a queue without removing them. The browser method returns a `java.util.Enumeration` that is used to scan messages in the queue. The first call to `nextElement` gets a snapshot of the queue. A `QueueBrowser` can be created with or without a `messageSelector`.

A `QueueBrowser` can also optionally lock messages as it is scanning them. This is similar to a "`SELECT`... `for UPDATE`" command on the message. This prevents other consumers from removing the message while they are being scanned.

MessageSelector

A `messageSelector` allows the client to restrict messages delivered to the consumer to those that match the `messageSelector` expression. A `messageSelector` for queues containing payloads of type `TextMessage`, `StreamMessage`, `BytesMessage`, `ObjectMessage`, or `MapMessage` can contain any expression that has one or more of the following:

- JMS message identifier prefixed with "ID:"

  ```
  JMSMessageID ='ID:23452345'
  ```

- JMS message header fields or properties

  ```
  JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
  ```

  ```
  JMSCorrelationID LIKE 'RE%'
  ```

- User-defined message properties

  ```
  color IN ('RED', BLUE', 'GREEN') AND price < 30000
  ```

The `messageSelector` for queues containing payloads of type `AdtMessage` can contain any expression that has one or more of the following:

- Message identifier without the "ID:" prefix

  ```
  msgid = '23434556566767676'
  ```

- Priority, correlation identifier, or both

  ```
  priority < 3 AND corrid = 'Fiction'
  ```

- Message payload

  ```
  tab.user_data.color = 'GREEN' AND tab.user_data.price < 30000
  ```

JMS Publish/Subscribe Model Features

This section contains these topics:

- JMS Publish/Subscribe Overview
- DurableSubscriber
- RemoteSubscriber
- TopicPublisher
- Recipient Lists
- TopicReceiver
- TopicBrowser
- Setting Up JMS Publish/Subscribe Operations

JMS Publish/Subscribe Overview

JMS enables flexible and dynamic communication between applications functioning as publishers and applications playing the role of subscribers. The applications are not coupled together; they interact based on messages and message content.

In distributing messages, publisher applications are not required to handle or manage message recipients explicitly. This allows new subscriber applications to be added dynamically without changing any publisher application logic.

Similarly, subscriber applications receive messages based on message content without regard to which publisher applications are sending messages. This allows new publisher applications to be added dynamically without changing any subscriber application logic.

Subscriber applications specify interest by defining a rule-based subscription on message properties or the message content of a topic. The system automatically routes messages by computing recipients for published messages using the rule-based subscriptions.

In the publish/subscribe model, messages are published to and received from topics. A topic is created using the `CreateTopic()` method in an `AQjmsSession`. A client can obtain a handle to a previously-created topic using the `getTopic()` method in `AQjmsSession`.

DurableSubscriber

A client creates a `DurableSubscriber` with the `createDurableSubscriber()` method in a client `Session`. It can be created with or without a `messageSelector`.

A `messageSelector` allows the client to restrict messages delivered to the subscriber to those that match the selector. The syntax for the selector is described in detail in `createDurableSubscriber` in *Oracle Database Advanced Queuing Java API Reference*.

When subscribers use the same name, durable subscriber action depends on the Java EE compliance mode set for an Oracle Java Message Service (Oracle JMS) client at runtime.

In noncompliant mode, two durable `TopicSubscriber` objects with the same name can be active against two different topics. In compliant mode, durable subscribers with the same name are not allowed. If two subscribers use the same name and are created against the same topic, but the selector used for each subscriber is different, then the underlying Oracle Database Advanced Queuing subscription is altered using the internal `DBMS_AQJMS.ALTER_SUBSCRIBER()` call.

If two subscribers use the same name and are created against two different topics, and if the client that uses the same subscription name also originally created the subscription name, then the existing subscription is dropped and the new subscription is created.

If two subscribers use the same name and are created against two different topics, and if a different client (a client that did not originate the subscription name) uses an existing subscription name, then the subscription is not dropped and an error is thrown. Because it is not known if the subscription was created by JMS or PL/SQL, the subscription on the other topic should not be dropped.

See Also:

- "MessageSelector"
- "Java EE Compliance"

RemoteSubscriber

Remote subscribers are defined using the `createRemoteSubscriber` call. The remote subscriber can be a specific consumer at the remote topic or all subscribers at the remote topic

A remote subscriber is defined using the `AQjmsAgent` structure. An `AQjmsAgent` consists of a name and address. The name refers to the `consumer_name` at the remote topic. The address refers to the remote topic:

```
schema.topic_name[@dblink]
```

To publish messages to a particular consumer at the remote topic, the `subscription_name` of the recipient at the remote topic must be specified in the name field of `AQjmsAgent`. The remote topic must be specified in the address field of `AQjmsAgent`.

To publish messages to all subscribers of the remote topic, the name field of `AQjmsAgent` must be set to null. The remote topic must be specified in the address field of `AQjmsAgent`.

TopicPublisher

Messages are published using `TopicPublisher`, which is created by passing a `Topic` to a `createPublisher` method. A client also has the option of creating a `TopicPublisher` without supplying a `Topic`. In this case, a `Topic` must be specified on every publish operation. A client can specify a default delivery mode, priority and `TimeToLive` for all messages sent by the `TopicPublisher`. It can also specify these options for each message.

Recipient Lists

In the JMS publish/subscribe model, clients can specify explicit recipient lists instead of having messages sent to all the subscribers of the topic. These recipients may or may not be existing subscribers of the topic. The recipient list overrides the subscription list on the topic for this message. Recipient lists functionality is an Oracle extension to JMS.

TopicReceiver

If the recipient name is explicitly specified in the recipient list, but that recipient is not a subscriber to the queue, then messages sent to it can be received by creating a `TopicReceiver`. If the subscriber name is not specified, then clients must use durable subscribers at the remote site to receive messages. `TopicReceiver` is an Oracle extension to JMS.

A `TopicReceiver` can be created with a `messageSelector`. This allows the client to restrict messages delivered to the recipient to those that match the selector.

See Also:

"MessageSelector"

TopicBrowser

A client uses a `TopicBrowser` to view messages on a topic without removing them. The browser method returns a `java.util.Enumeration` that is used to scan topic messages. Only durable subscribers are allowed to create a `TopicBrowser`. The first call to `nextElement` gets a snapshot of the topic.

A `TopicBrowser` can optionally lock messages as it is scanning them. This is similar to a `SELECT`... `for UPDATE` command on the message. This prevents other consumers from removing the message while it is being scanned.

A `TopicBrowser` can be created with a `messageSelector`. This allows the client to restrict messages delivered to the browser to those that match the selector.

`TopicBrowser` supports a purge feature. This allows a client using a `TopicBrowser` to discard all messages that have been seen during the current browse operation on the topic. A purge is equivalent to a destructive receive of all of the seen messages (as if performed using a `TopicSubscriber`).

For a purge, a message is considered seen if it has been returned to the client using a call to the `nextElement()` operation on the `java.lang.Enumeration` for the `TopicBrowser`. Messages that have not yet been seen by the client are not discarded during a purge. A purge operation can be performed multiple times on the same `TopicBrowser`.

The effect of a purge becomes stable when the JMS `Session` used to create the `TopicBrowser` is committed. If the operations on the session are rolled back, then the effects of the purge operation are also undone.

See Also:

- "Creating a TopicBrowser for Standard JMS Messages"
- "Creating a TopicBrowser for Standard JMS Messages_ Locking Messages"
- "MessageSelector"
- "Browsing Messages Using a TopicBrowser"

Setting Up JMS Publish/Subscribe Operations

Follow these steps to use the publish/subscribe model of communication in JMS:

1. Set up one or more topics to hold messages. These topics represent an area or subject of interest. For example, a topic can represent billed orders.
2. Enable enqueue/dequeue on the topic using the `start` call in `AQjmsDestination`.
3. Create a set of durable subscribers. Each subscriber can specify a `messageSelector` that selects the messages that the subscriber wishes to receive. A null `messageSelector` indicates that the subscriber wishes to receive all messages published on the topic.

   Subscribers can be local or remote. Local subscribers are durable subscribers defined on the same topic on which the message is published. Remote subscribers are other topics, or recipients on other topics that are defined as subscribers to a particular queue. In order to use remote subscribers, you must set up propagation between the source and destination topics. Remote subscribers and propagation are Oracle extensions to JMS.

   See Also:

   "Managing Propagations"

4. Create `TopicPublisher` objects using the `createPublisher()` method in the publisher `Session`. Messages are published using the `publish` call. Messages can be published to all subscribers to the topic or to a specified subset of recipients on the topic.
5. Subscribers receive messages on the topic by using the `receive` method.
6. Subscribers can also receive messages asynchronously by using message listeners.

   See Also:

   "Listening to One or More Queues"

JMS Message Producer Features

- Priority and Ordering of Messages
- Specifying a Message Delay
- Specifying a Message Expiration
- Message Grouping

Priority and Ordering of Messages

Message ordering dictates the order in which messages are received from a queue or topic. The ordering method is specified when the queue table for the queue or topic is created. Currently, Oracle Database Advanced Queuing supports ordering on message priority and enqueue time, producing four possible ways of ordering:

- First-In, First-Out (FIFO)

  If enqueue time was chosen as the ordering criteria, then messages are received in the order of the enqueue time. The enqueue time is assigned to the message by Oracle Database Advanced Queuing at message publish/send time. This is also the default ordering.
- Priority Ordering

  If priority ordering was chosen, then each message is assigned a priority. Priority can be specified as a message property at publish/send time by the `MessageProducer`. The messages are received in the order of the priorities assigned.
- FIFO Priority

  If FIFO priority ordering was chosen, then the topic/queue acts like a priority queue. If two messages are assigned the same priority, then they are received in the order of their enqueue time.
- Enqueue Time Followed by Priority

  Messages with the same enqueue time are received according to their priorities. If the ordering criteria of two message is the same, then the order they are received is indeterminate. However, Oracle Database Advanced Queuing does ensure that messages produced in one session with a particular ordering criteria are received in the order they were sent.

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

Specifying a Message Delay

Messages can be sent/published to a queue/topic with delay. The delay represents a time interval after which the message becomes available to the message consumer. A message specified with a delay is in a waiting state until the delay expires. Receiving by message identifier overrides the delay specification.

Delay is an Oracle Database Advanced Queuing extension to JMS message properties. It requires the Oracle Database Advanced Queuing background process queue monitor to be started.

Specifying a Message Expiration

Producers of messages can specify expiration limits, or `TimeToLive` for messages. This defines the period of time the message is available for a Message Consumer.

`TimeToLive` can be specified at send/publish time or using the set `TimeToLive` method of a `MessageProducer`, with the former overriding the latter. The Oracle Database Advanced Queuing background process queue monitor must be running to implement `TimeToLive`.

Message Grouping

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

Message grouping is an Oracle Database Advanced Queuing extension to the JMS specification.

You can use this feature to divide a complex message into a linked series of simple messages. For example, an invoice directed to an invoices queue could be divided into a header message, followed by several messages representing details, followed by the trailer message.

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

The priority, delay, and expiration properties for the messages in a group are determined solely by the message properties specified for the first message (head) of the group. Properties specified for subsequent messages in the group are ignored.

Message grouping is preserved during propagation. The destination topic must be enabled for transactional grouping.

See Also:

"Dequeue Features" for a discussion of restrictions you must keep in mind if message grouping is to be preserved while dequeuing messages from a queue enabled for transactional grouping

JMS Message Consumer Features

This section contains these topics:

- Receiving Messages
- Message Navigation in Receive
- Browsing Messages
- Remove No Data
- Retry with Delay Interval
- Asynchronously Receiving Messages Using MessageListener
- Exception Queues

Receiving Messages

A JMS application can receive messages by creating a message consumer. Messages can be received synchronously using the `receive` call or asynchronously using a message listener.

There are three modes of receive:

- Block until a message arrives for a consumer

- Block for a maximum of the specified time
- Nonblocking

Message Navigation in Receive

If a consumer does not specify a navigation mode, then its first `receive` in a session retrieves the first message in the queue or topic, its second `receive` gets the next message, and so on. If a high priority message arrives for the consumer, then the consumer does not receive the message until it has cleared the messages that were already there before it.

To provide the consumer better control in navigating the queue for its messages, Oracle Database Advanced Queuing offers several navigation modes as JMS extensions. These modes can be set at the `TopicSubscriber`, `QueueReceiver` or the `TopicReceiver`.

Two modes are available for ungrouped messages:

- `FIRST_MESSAGE`

  This mode resets the position to the beginning of the queue. It is useful for priority ordered queues, because it allows the consumer to remove the message on the top of the queue.
- `NEXT_MESSAGE`

  This mode gets whatever message follows the established position of the consumer. For example, a `NEXT_MESSAGE` applied when the position is at the fourth message will get the fifth message in the queue. This is the default action.

Three modes are available for grouped messages:

- `FIRST_MESSAGE`

  This mode resets the position to the beginning of the queue.
- `NEXT_MESSAGE`

  This mode sets the position to the next message in the same transaction.
- `NEXT_TRANSACTION`

  This mode sets the position to the first message in the next transaction.

Note:

Transactional event queues do not support the three preceding modes.

The transaction grouping property can be negated if messages are received in the following ways:

- Receive by specifying a correlation identifier in the selector
- Receive by specifying a message identifier in the selector
- Committing before all the messages of a transaction group have been received

If the consumer reaches the end of the queue while using the `NEXT_MESSAGE` or `NEXT_TRANSACTION` option, and you have specified a blocking `receive()`, then the navigating position is automatically changed to the beginning of the queue.

By default, a `QueueReceiver`, `TopicReceiver`, or `TopicSubscriber` uses `FIRST_MESSAGE` for the first receive call, and `NEXT_MESSAGE` for subsequent `receive()` calls.

Browsing Messages

Aside from the usual `receive`, which allows the dequeuing client to delete the message from the queue, JMS provides an interface that allows the JMS client to browse its messages in the queue. A `QueueBrowser` can be created using the `createBrowser` method from `QueueSession`.

If a message is browsed, then it remains available for further processing. That does not necessarily mean that the message will remain available to the JMS session after it is browsed, because a `receive` call from a concurrent session might remove it.

To prevent a viewed message from being removed by a concurrent JMS client, you can view the message in the locked mode. To do this, you must create a `QueueBrowser` with the locked mode using the Oracle Database Advanced Queuing extension to the JMS interface. The lock on the message is released when the session performs a commit or a rollback.

To remove a message viewed by a `QueueBrowser`, the session must create a `QueueReceiver` and use the `JMSmesssageID` as the selector.

Remove No Data

The consumer can remove a message from a queue or topic without retrieving it using the `receiveNoData` call. This is useful when the application has already examined the message, perhaps using a `QueueBrowser`. This mode allows the JMS client to avoid the overhead of retrieving a payload from the database, which can be substantial for a large message.

Retry with Delay Interval

If a transaction receiving a message from a queue/topic fails, then it is regarded as an unsuccessful attempt to remove the message. Oracle Database Advanced Queuing records the number of failed attempts to remove the message in the message history.

An application can specify the maximum number of retries supported on messages at the queue/topic level. If the number of failed attempts to remove a message exceeds this maximum, then the message is moved to an exception queue.

Oracle Database Advanced Queuing allows users to specify a `retry_delay` along with `max_retries`. This means that a message that has undergone a failed attempt at retrieving remains visible in the queue for dequeue after `retry_delay` interval. Until then it is in the `WAITING` state. The Oracle Database Advanced Queuing background process time manager enforces the retry delay property.

The maximum retries and retry delay are properties of the queue/topic. They can be set when the queue/topic is created or by using the alter method on the queue/topic. The default value for `MAX_RETRIES` is `5`.

Note:

Transactional event queues do not support retry delay.

Asynchronously Receiving Messages Using MessageListener

The JMS client can receive messages asynchronously by setting the `MessageListener` using the `setMessageListener` method.

When a message arrives for the consumer, the `onMessage` method of the message listener is invoked with the message. The message listener can commit or terminate the receipt of the message. The message listener does not receive messages if the JMS `Connection` has been stopped. The `receive` call must not be used to receive messages once the message listener has been set for the consumer.

The JMS client can receive messages asynchronously for all consumers in the session by setting the `MessageListener` at the session. No other mode for receiving messages must be used in the session once the message listener has been set.

Exception Queues

An exception queue is a repository for all expired or unserviceable messages. Applications cannot directly enqueue into exception queues. However, an application that intends to handle these expired or unserviceable messages can receive/remove them from the exception queue.

To retrieve messages from exception queues, the JMS client must use the point-to-point interface. The exception queue for messages intended for a topic must be created in a queue table with multiple consumers enabled. Like any other queue, the exception queue must be enabled for receiving messages using the `start` method in the `AQOracleQueue` class. You get an exception if you try to enable it for enqueue.

Transactional event queues (TxEventQ) support exception queues through the `DBMS_AQADM.CREATE_EQ_EXCEPTION_QUEUE` API.

```
PROCEDURE CREATE_EQ_EXCEPTION_QUEUE(
  queue_name     IN VARCHAR2,
  exception_queue_name   IN VARCHAR2 DEFAULT NULL,
  multiple_consumers     IN BOOLEAN DEFAULT FALSE,
  storage_clause         IN VARCHAR2 DEFAULT NULL,
  sort_list              IN VARCHAR DEFAULT NULL,
  comment                IN VARCHAR2 DEFAULT NULL
  );
```

The exception queue is an Oracle-specific message property called "`JMS_OracleExcpQ`" that can be set with the message before sending/publishing it. If an exception queue is not specified, then the default exception queue is used. For AQ queues, the default exception queue is automatically created when the queue table is created and is named `AQ$_`*`queue_table_name`*`_E`. By default, no exception queue is created for TxEventQs.

Messages are moved to the exception queue under the following conditions:

- The message was not dequeued within the specified `timeToLive`.

  For messages intended for more than one subscriber, the message is moved to the exception queue if one or more of the intended recipients is not able to dequeue the message within the specified `timeToLive`.

- The message was received successfully, but the application terminated the transaction that performed the `receive` because of an error while processing the message. The message is returned to the queue/topic and is available for any applications that are waiting to receive messages.

A `receive` is considered rolled back or undone if the application terminates the entire transaction, or if it rolls back to a savepoint that was taken before the `receive`.

Because this was a failed attempt to receive the message, its retry count is updated. If the retry count of the message exceeds the maximum value specified for the queue/topic where it resides, then it is moved to the exception queue.

If a message has multiple subscribers, then the message is moved to the exception queue only when all the recipients of the message have exceeded the retry limit.

Note:

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

JMS Propagation

This section contains these topics:

- RemoteSubscriber
- Scheduling Propagation
- Enhanced Propagation Scheduling Capabilities
- Exception Handling During Propagation

Note:

TxEventQ queues do not support RemoteSubscriber, Scheduling Propagation, Enhanced Propagation Scheduling Capabilities, and Exception Handling During Propagation.

RemoteSubscriber

Oracle Database Advanced Queuing allows a subscriber at another database to subscribe to a topic. If a message published to the topic meets the criterion of the remote subscriber, then it is automatically propagated to the queue/topic at the remote database specified for the remote subscriber. Propagation is performed using database links and Oracle Net Services. This enables applications to communicate with each other without having to be connected to the same database.

There are two ways to implement remote subscribers:

- The `createRemoteSubscriber` method can be used to create a remote subscriber to/on the topic. The remote subscriber is specified as an instance of the class `AQjmsAgent`.
- The `AQjmsAgent` has a name and an address. The address consists of a queue/topic and the database link to the database of the subscriber.

There are two kinds of remote subscribers:

- The remote subscriber is a topic.

  This occurs when no name is specified for the remote subscriber in the `AQjmsAgent` object and the address is a topic. The message satisfying the subscriber's subscription is propagated to the remote topic. The propagated message is now available to all the subscriptions of the remote topic that it satisfies.
- A specific remote recipient is specified for the message.

  The remote subscription can be for a particular consumer at the remote database. If the name of the remote recipient is specified (in the `AQjmsAgent` object), then the message satisfying the subscription is propagated to the remote database for that recipient only. The recipient at the remote database uses the `TopicReceiver` interface to retrieve its messages. The remote subscription can also be for a point-to-point queue.

Scheduling Propagation

Propagation must be scheduled using the `schedule_propagation` method for every topic from which messages are propagated to target destination databases.

A schedule indicates the time frame during which messages can be propagated from the source topic. This time frame can depend on several factors such as network traffic, the load at the source database, the load at the destination database, and so on. The schedule therefore must be tailored for the specific source and destination. When a schedule is created, a job is automatically submitted to the `job_queue` facility to handle propagation.

The administrative calls for propagation scheduling provide great flexibility for managing the schedules. The duration or propagation window parameter of a schedule specifies the time frame during which propagation must take place. If the duration is unspecified, then the time frame is an infinite single window. If a window must be repeated periodically, then a finite duration is specified along with a `next_time` function that defines the periodic interval between successive windows.

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue. In addition there are calls for temporarily disabling a schedule (instead of dropping the schedule) and enabling a disabled schedule. A schedule is active when messages are being propagated in that schedule. All the administrative calls can be made irrespective of whether the schedule is active or not. If a schedule is active, then it takes a few seconds for the calls to be executed.

Job queue processes must be started for propagation to take place. At least 2 job queue processes must be started. The database links to the destination database must also be valid. The source and destination topics of the propagation must be of the same message type. The remote topic must be enabled for enqueue. The user of the database link must also have enqueue privileges to the remote topic.

See Also:

"Scheduling a Propagation"

Enhanced Propagation Scheduling Capabilities

Catalog views defined for propagation provide the following information about active schedules:

- Name of the background process handling the schedule

- SID (session and serial number) for the session handling the propagation
- Instance handling a schedule (if using Oracle RAC)
- Previous successful execution of a schedule
- Next planned execution of a schedule

The following propagation statistics are maintained for each schedule, providing useful information to queue administrators for tuning:

- The total number of messages propagated in a schedule
- Total number of bytes propagated in a schedule
- Maximum number of messages propagated in a window
- Maximum number of bytes propagated in a window
- Average number of messages propagated in a window
- Average size of propagated messages
- Average time to propagated a message

Propagation has built-in support for handling failures and reporting errors. For example, if the database link specified is invalid, or if the remote database is unavailable, or if the remote topic/queue is not enabled for enqueuing, then the appropriate error message is reported. Propagation uses an exponential backoff scheme for retrying propagation from a schedule that encountered a failure. If a schedule continuously encounters failures, then the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, then the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts are made after which the schedule is automatically disabled.

Note:

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

When a schedule is disabled automatically due to failures, the relevant information is written into the alert log. It is possible to check at any time if there were failures encountered by a schedule and if so how many successive failures were encountered, the error message indicating the cause for the failure and the time at which the last failure was encountered. By examining this information, an administrator can fix the failure and enable the schedule.

If propagation is successful during a retry, then the number of failures is reset to 0.

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

The number of `job_queue_processes` is limited to a maximum of 1000 and some of these can be used to handle jobs unrelated to propagation. Hence, propagation has built in support for multitasking and load balancing. The propagation algorithms are designed such that multiple schedules can be handled by a single snapshot (`job_queue`) process. The propagation load on a `job_queue` processes can be skewed based on the arrival rate of messages in the different source topics. If one process is overburdened with several active schedules while another is less loaded with many passive schedules, then propagation automatically redistributes the schedules among the processes such that they are loaded uniformly.

Exception Handling During Propagation

When a system error such as a network failure occurs, Oracle Database Advanced Queuing continues to attempt to propagate messages using an exponential back-off algorithm. In some situations that indicate application errors in queue-to-dblink propagations, Oracle Database Advanced Queuing marks messages as `UNDELIVERABLE` and logs a message in `alert.log`. Examples of such errors are when the remote queue does not exist or when there is a type mismatch between the source queue and the remote queue. The trace files in the `background_dump_dest` directory can provide additional information about the error.

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

Note:

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

Message Transformation with JMS AQ

A transformation can be defined to map messages of one format to another. Transformations are useful when applications that use different formats to represent the same information must be integrated. Transformations can be SQL expressions and PL/SQL functions. Message transformation is an Oracle Database Advanced Queuing extension to the standard JMS interface.

The transformations can be created using the `DBMS_TRANSFORM.create_transformation` procedure. Transformation can be specified for the following operations:

- Sending a message to a queue or topic
- Receiving a message from a queue or topic
- Creating a `TopicSubscriber`
- Creating a `RemoteSubscriber`. This enables propagation of messages between topics of different formats.

> **Note:**
>
> TxEventQ does not support message transformation.

JMS Streaming

AQ JMS supports streaming with enqueue and dequeue for TxEventQ through `AQjmsBytesMessage` and `AQjmsStreamMessage` for applications to send and receive large message data or payload.

JMS streaming reduces the memory requirement when dealing with large messages, by dividing the message payload into small chunks rather than sending or receiving a large contiguous array of bytes. As JMS standard does not have any streaming mechanism, AQ JMS will provide proprietary interfaces to expose AQ streaming enqueue and dequeue features. This allows users to easily use an existing java input output stream to send and receive message data or payload.

In order to allow the existing applications to work without any changes on upgrading database to RDBMS 12.2, the streaming APIs will be disabled by default.

The client application can enable JMS Streaming by using the system property `oracle.jms.useJmsStreaming` set to `true`.

> **Note:**
>
> JMS Streaming is supported only for thin drivers.

JMS Streaming with Enqueue

AQ JMS provides the new API `setInputStream(java.io.InputStream)` in `AQjmsBytesMessage` and `AQjmsStreamMessage`, to set an input stream for message data.

```
/**
 * @param inputStream - InputStream to read the message payload
 * @throws JMSException - if the JMS provided fails to read the payload due to
 *                         some internal error
 */
public void setInputStream(InputStream inputStream) throws JMSException
```

The following code snippet creates a message of type `AQjmsBytesMessage` and sets a `FileInputStream` for the message data.

```
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
Destination destination = session.createQueue("queueName");
MessageProducer producer = session.createProducer(destination);

 AQjmsBytesMessage bytesMessage = (AQjmsBytesMessage)session.createBytesMessage();
 InputStream input = new FileInputStream("somefile.data");
 bytesMessage.setInputStream(input);
 producer.send(bytesMessage);
```

Note:

- The methods in `BytesMessage` and `StreamMessage` are based on the methods found in `java.io.DataInputStream` and `java.io.DataOutputStream`, and hence, meaningful conversion of various `read*()` and `write*()` methods is not possible with streaming. The following scenarios will result in an exception:
  - `bytesMessage.setInputStream(input);`

    `bytesMessage.writeInt(99);`
  - `bytesMessage.writeInt(99);`

    `bytesMessage.setInputStream(input);`
- As with normal enqueue operation, the enqueue with streaming is going to be a synchronous one and we will return the control to the client only after the enqueue is complete.
- Streaming will be used with enqueue only when these APIs are explicitly used by the client. AQ JMS will not use streaming with enqueue with the normal enqueue, irrespective of the size of the message data.

JMS Streaming with Dequeue

The dequeue operation with streaming is achieved in two steps. The server decides whether to stream the message body or not based on the size of the message body. The default threshold limit is 10 MB. So when the message body is greater than 10MB and streaming is enabled by the client using the system property oracle.jms.useJmsStreaming, server will use streaming with dequeue.

- This is the normal dequeue process where a client calls the `receive()` method.

```
Destination destination = session.createQueue ("queueName");
AQjmsConsumer consumer = (AQjmsConsumer)
session.createConsumer(destination);
Message message = consumer.receive(10000);
```

- When the client receives the message without the payload, client can figure out whether the streaming is used for dequeue by calling `isLargeBody()` on the received message.

```
/**
 * This method can be used by the client applications to check whether the message
 * contains large messaege body and hence requires streaming with dequeue.
 *
 * @return true when the message body is large and server decides to stream
 *         the payload with dequeue
 */
public boolean isLargeBody()
```

  A value of true returned by `isLargeBody()` indicates streaming with dequeue. When the dequeue uses streaming, AQ JMS will populate the length of the message body properly for `AQjmsStreamMessage` along with `AQjmsBytesMessage`. So the client application can call the `getBodyLength()` on the message to determine the size of the payload.

```
public long getBodyLength()
```

Once client has the understanding about the streaming with dequeue, the message data can be fetched by using one of the following APIs on the received message.

The client application can use on the following APIs available in `AQjmsBytesMessage` and `AQjmsStreamMessage` to receive the message data.

```
 /**
     * Writes the message body to the OutputStream specified.
     *
     * @param outputStream - the OutputStream to which message body can be written
     * @return the OutputStream containing the message body.
     * @throws JMSException - if the JMS provided fails to receive the message body
     *                         due to some internal error
     */
    public OutputStream getBody(OutputStream outputStream) throws JMSException

    /**
     * Writes the message body to the OutputStream specified, with chunkSize bytes
     * written at a time.
     *
     * @param outputStream - the OutputStream to which message body can be written
     * @param chunkSize - the number of bytes to be written at a time, default value
     *                    8192 (ie. 8KB)
     * @return the OutputStream containing the message body.
     * @throws JMSException - if the JMS provided fails to receive the message body
     *                         due to some internal error
     */
    public OutputStream getBody(OutputStream outputStream, int chunkSize)throws
JMSException


    /**
     * Writes the message body to the OutputStream specified. This method waits until
     * the message body is written completely to the OutputStream or the timeout expires.
     *
     * A timeout of zero never expires, and a timeout of negative value is ignored.
     *
     * @param outputStream - the OutputStream to which message body can be written
     * @param timeout - the timeout value (in milliseconds)
     * @return the OutputStream containing the message body.
     * @throws JMSException - if the JMS provided fails to receive the message body
     *                         due to some internal error
     */
    public OutputStream getBody(OutputStream outputStream, long timeout) throws
JMSException


    /**
     * Writes the message body to the OutputStream specified, chunkSize bytes at a time.
     * This method waits until the message body is written completely to the OutputStream
     * or the timeout expires.
     *
     * A timeout of zero never expires, and a timeout of negative value is ignored.
     *
     * @param outputStream - the OutputStream to which message body can be written
     * @param chunkSize - the number of bytes to be written at a time,
     *                    default value 8192 (ie. 8KB)
     * @param timeout - the timeout value (in milliseconds)
     * @return the OutputStream containing the message body.
     * @throws JMSException - if the JMS provided fails to receive the message body
     *                         due to some internal error
     */
    public OutputStream getBody(OutputStream outputStream, int chunkSize, long timeout)
throws JMSException
```

The following code snippet checks whether streaming is used with dequeue and the payload received will be written to a `FileOutputStream`.

```
if (message instanceof BytesMessage && (AQjmsBytesMessage)message.isLargeBody()){
    // optional : check the size of the payload and take appropriate action before
    // receiving the payload.
     (AQjmsBytesMessage) message.getBody(new FileOutputStream(new File("…")));
} else {
     // normal dequeue
}
```

In general, when both the steps are complete, the message is considered as consumed completely. The AQ server keeps a lock on the message after Step 1 which will be released only after Step 2.

Considering the possible issues with partially consumed messages by the message consumers, we have restricted the Streaming APIs for the session with acknowledgement modes `CLIENT_ACKNOWLEDGE` and `SESSION_TRANSACTED`.

So all the messages including partially consumed messages are considered fully consumed when:

- `message.acknowledge()` is called with `CLIENT_ACKNOWLEDGE` session.
- Session's `commit()` is called in a transacted session.

As in normal case, session `rollback()`, rolls back the messages received in that session.

The JMS Streaming is available with the following restrictions:

- Streaming is disabled by default, and can be enabled by the client application using the system property `oracle.jms.useJmsStreaming`
- Dequeue uses streaming when the size of the message data is more than the threshold value. The default threshold value is 10 MB.
- Streaming support is available with `AQjmsBytesMessage` and `AQjmsStreamMessage`
- Streaming support is available only for TxEventQ queues
- Streaming support is available only with thin drivers
- Streaming support is not available when the message producer uses the message delivery mode as `NON_PERSISTENT`
- Streaming is not supported with message listener. So when a MessageConsumer has a message listener set and if the message data crosses threshold limit, internally we will use the normal dequeue.
- Streaming support is available with Sessions using acknowledgement modes `CLIENT_ACKNOWLEDGE` and `SESSION_TRANSACTED`.

Java EE Compliance

Oracle JMS conforms to the Oracle Sun Microsystems JMS 1.1 standard. You can define the Java EE compliance mode for an Oracle Java Message Service (Oracle JMS) client at runtime. For compliance, set the Java property `oracle.jms.j2eeCompliant` to `TRUE` as a command line option. For noncompliance, do nothing. `FALSE` is the default value.

Features in Oracle Database Advanced Queuing that support Java EE compliance (and are also available in the noncompliant mode) include:

- Nontransactional sessions

- Durable subscribers
- Temporary queues and topics
- Nonpersistent delivery mode
- Multiple JMS messages types on a single JMS queue or topic (using Oracle Database Advanced Queuing queues of the `AQ$_JMS_MESSAGE` type)
- The `noLocal` option for durable subscribers
- TxEventQ has native JMS support and conform to Java EE compliance

See Also:

- *Java Message Service Specification*, version 1.1, March 18, 2002, Sun Microsystems, Inc.
- "JMS Message Headers" for information on how the Java property `oracle.jms.j2eeCompliant` affects JMSPriority and JMSExpiration
- "DurableSubscriber" for information on how the Java property `oracle.jms.j2eeCompliant` affects durable subscribers

Oracle Java Message Service Basic Operations

The following topics describe the basic operational Java Message Service (JMS) administrative interface to Oracle Database Advanced Queuing (AQ).

- EXECUTE Privilege on DBMS_AQIN
- Registering a ConnectionFactory
- Unregistering a Queue/Topic ConnectionFactory
- Getting a QueueConnectionFactory or TopicConnectionFactory
- Getting a Queue or Topic in LDAP
- Creating an AQ Queue Table
- Creating a Queue
- Getting an AQ Queue Table
- Granting and Revoking Privileges
- Managing Destinations
- Propagation Schedules

EXECUTE Privilege on DBMS_AQIN

Users should never directly call methods in the `DBMS_AQIN` package, but they do need the `EXECUTE` privilege on `DBMS_AQIN`. Use the following syntax to accomplish this:

```
GRANT EXECUTE ON DBMS_AQIN to user;
```

Registering a ConnectionFactory

You can register a ConnectionFactory four ways:

- Registering Through the Database Using JDBC Connection Parameters
- Registering Through the Database Using a JDBC URL
- Registering Through LDAP Using JDBC Connection Parameters
- Registering Through LDAP Using a JDBC URL

Registering Through the Database Using JDBC Connection Parameters

```
public static int registerConnectionFactory(java.sql.Connection connection,
                                            java.lang.String conn_name,
                                            java.lang.String hostname,
                                            java.lang.String oracle_sid,
                                            int portno,
                                            java.lang.String driver,
                                            java.lang.String type)
                                     throws JMSException
```

This method registers a `QueueConnectionFactory` or TopicConnectionFactory through the database to a Lightweight Directory Access Protocol (LDAP) server with JDBC connection parameters. This method is static and has the following parameters:

Parameter	Description
`connection`	JDBC connection used in registration
`conn_name`	Name of the connection to be registered
`hostname`	Name of the host running Oracle Database Advanced Queuing
`oracle_sid`	Oracle system identifier
`portno`	Port number
`driver`	JDBC driver type
`type`	Connection factory type (`QUEUE` or `TOPIC`)

The database connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, you can look up the connection factory using Java Naming and Directory Interface (JNDI).

Example 6-1   Registering Through the Database Using JDBC Connection Parameters

```
String               url;
java.sql.connection  db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.registerConnectionFactory(
   db_conn, "queue_conn1", "sun-123", "db1", 1521, "thin", "queue");
```

Registering Through the Database Using a JDBC URL

```
public static int registerConnectionFactory(java.sql.Connection connection,
                                            java.lang.String conn_name,
                                            java.lang.String jdbc_url,
                                            java.util.Properties info,
                                            java.lang.String type)
                                     throws JMSException
```

This method registers a `QueueConnectionFactory` or TopicConnectionFactory through the database with a JDBC URL to LDAP. It is static and has the following parameters:

Parameter	Description
`connection`	JDBC connection used in registration
`conn_name`	Name of the connection to be registered
`jdbc_url`	URL to connect to
`info`	Properties information
`portno`	Port number
`type`	Connection factory type (`QUEUE` or `TOPIC`)

The database connection passed to `registerConnectionFactory` must be granted `AQ_ADMINISTRATOR_ROLE`. After registration, you can look up the connection factory using JNDI.

Example 6-2   Registering Through the Database Using a JDBC URL

```
String                        url;
java.sql.connection           db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.registerConnectionFactory(
   db_conn, "topic_conn1", url, null, "topic");
```

Registering Through LDAP Using JDBC Connection Parameters

```
public static int registerConnectionFactory(java.util.Hashtable env,
                                            java.lang.String conn_name,
                                            java.lang.String hostname,
                                            java.lang.String oracle_sid,
                                            int portno,
                                            java.lang.String driver,
                                            java.lang.String type)
                                     throws JMSException
```

This method registers a `QueueConnectionFactory` or TopicConnectionFactory through LDAP with JDBC connection parameters to LDAP. It is static and has the following parameters:

Parameter	Description
`env`	Environment of LDAP connection
`conn_name`	Name of the connection to be registered
`hostname`	Name of the host running Oracle Database Advanced Queuing
`oracle_sid`	Oracle system identifier
`portno`	Port number
`driver`	JDBC driver type
`type`	Connection factory type (`QUEUE` or `TOPIC`)

The hash table passed to `registerConnectionFactory()` must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted `GLOBAL_AQ_USER_ROLE`). After registration, look up the connection factory using JNDI.

Example 6-3 Registering Through LDAP Using JDBC Connection Parameters

```
Hashtable            env = new Hashtable(5, 0.75f);
/* the following statements set in hashtable env:
   * service provider package
   * the URL of the ldap server
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put("searchbase", "cn=db1,cn=Oraclecontext,cn=acme,cn=com");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aqadmin,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");

AQjmsFactory.registerConnectionFactory(env,
                                       "queue_conn1",
                                       "sun-123",
                                       "db1",
                                       1521,
                                       "thin",
                                       "queue");
```

Registering Through LDAP Using a JDBC URL

```
public static int registerConnectionFactory(java.util.Hashtable env,
                                            java.lang.String conn_name,
                                            java.lang.String jdbc_url,
                                            java.util.Properties info,
                                            java.lang.String type)
                                     throws JMSException
```

This method registers a `QueueConnectionFactory` or TopicConnectionFactory through LDAP with JDBC connection parameters to LDAP. It is static and has the following parameters:

Parameter	Description
`env`	Environment of LDAP connection
`conn_name`	Name of the connection to be registered
`jdbc_url`	URL to connect to
info	Properties information
`type`	Connection factory type (`QUEUE` or `TOPIC`)

The hash table passed to `registerConnectionFactory()` must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted `GLOBAL_AQ_USER_ROLE)`. After registration, look up the connection factory using JNDI.

Example 6-4 Registering Through LDAP Using a JDBC URL

```
String               url;
Hashtable            env = new Hashtable(5, 0.75f);

/* the following statements set in hashtable env:
   * service provider package
   * the URL of the ldap server
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put("searchbase", "cn=db1,cn=Oraclecontext,cn=acme,cn=com");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aqadmin,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
url = "jdbc:oracle:thin:@sun-123:1521:db1";
AQjmsFactory.registerConnectionFactory(env, "topic_conn1", url, null, "topic");
```

Unregistering a Queue/Topic ConnectionFactory

You can unregister a queue/topic `ConnectionFactory` in LDAP two ways:

- Unregistering Through the Database
- Unregistering Through LDAP

Unregistering Through the Database

```
public static int unregisterConnectionFactory(java.sql.Connection connection,
                                              java.lang.String conn_name)
                                       throws JMSException
```

This method unregisters a `QueueConnectionFactory` or `TopicConnectionFactory` in LDAP. It is static and has the following parameters:

Parameter	Description
`connection`	JDBC connection used in registration
`conn_name`	Name of the connection to be registered

The database connection passed to `unregisterConnectionFactory()` must be granted `AQ_ADMINISTRATOR_ROLE`.

Example 6-5 Unregistering Through the Database

```
String               url;
java.sql.connection  db_conn;

url = "jdbc:oracle:thin:@sun-123:1521:db1";
db_conn = DriverManager.getConnection(url, "scott", "tiger");
AQjmsFactory.unregisterConnectionFactory(db_conn, "topic_conn1");
```

Unregistering Through LDAP

```
public static int unregisterConnectionFactory(java.util.Hashtable env,
                                              java.lang.String conn_name)
                                       throws JMSException
```

This method unregisters a `QueueConnectionFactory` or TopicConnectionFactory in LDAP. It is static and has the following parameters:

Parameter	Description
`env`	Environment of LDAP connection
`conn_name`	Name of the connection to be registered

The hash table passed to `unregisterConnectionFactory()` must contain all the information to establish a valid connection to the LDAP server. Furthermore, the connection must have write access to the connection factory entries in the LDAP server (which requires the LDAP user to be either the database itself or be granted `GLOBAL_AQ_USER_ROLE`).

Example 6-6 Unregistering Through LDAP

```
Hashtable            env = new Hashtable(5, 0.75f);

/* the following statements set in hashtable env:
   * service provider package
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put("searchbase", "cn=db1,cn=Oraclecontext,cn=acme,cn=com");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aqadmin,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
url = "jdbc:oracle:thin:@sun-123:1521:db1";
AQjmsFactory.unregisterConnectionFactory(env, "queue_conn1");
```

Getting a QueueConnectionFactory or TopicConnectionFactory

This section contains these topics:

- Getting a QueueConnectionFactory with JDBC URL
- Getting a QueueConnectionFactory with JDBC Connection Parameters
- Getting a TopicConnectionFactory with JDBC URL
- Getting a TopicConnectionFactory with JDBC Connection Parameters
- Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

Getting a QueueConnectionFactory with JDBC URL

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String jdbc_url,
              java.util.Properties info)
       throws JMSException
```

This method gets a `QueueConnectionFactory` with JDBC URL. It is static and has the following parameters:

Parameter	Description
`jdbc_url`	URL to connect to
info	Properties information

Example 6-7 Getting a QueueConnectionFactory with JDBC URL

```
String      url          = "jdbc:oracle:oci10:internal/oracle"
Properties  info         = new Properties();
QueueConnectionFactory   qc_fact;

info.put("internal_logon", "sysdba");
qc_fact = AQjmsFactory.getQueueConnectionFactory(url, info);
```

Getting a QueueConnectionFactory with JDBC Connection Parameters

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String hostname,
              java.lang.String oracle_sid,
              int portno,
              java.lang.String driver)
       throws JMSException
```

This method gets a `QueueConnectionFactory` with JDBC connection parameters. It is static and has the following parameters:

Parameter	Description
`hostname`	Name of the host running Oracle Database Advanced Queuing
`oracle_sid`	Oracle system identifier
`portno`	Port number
`driver`	JDBC driver type

Example 6-8 Getting a QueueConnectionFactory with JDBC Connection Parameters

```
String      host         = "dlsun";
String      ora_sid      = "rdbms10i"
String      driver       = "thin";
int         port         = 5521;
QueueConnectionFactory   qc_fact;

qc_fact = AQjmsFactory.getQueueConnectionFactory(host, ora_sid, port, driver);
```

Getting a TopicConnectionFactory with JDBC URL

```
public static javax.jms.QueueConnectionFactory getQueueConnectionFactory(
              java.lang.String jdbc_url,
              java.util.Properties info)
       throws JMSException
```

This method gets a `TopicConnectionFactory` with a JDBC URL. It is static and has the following parameters:

Parameter	Description
`jdbc_url`	URL to connect to
info	Properties information

Example 6-9   Getting a TopicConnectionFactory with JDBC URL

```
String      url          = "jdbc:oracle:oci10:internal/oracle"
Properties  info         = new Properties();
TopicConnectionFactory   tc_fact;

info.put("internal_logon", "sysdba");
tc_fact = AQjmsFactory.getTopicConnectionFactory(url, info);
```

Getting a TopicConnectionFactory with JDBC Connection Parameters

```
public static javax.jms.TopicConnectionFactory getTopicConnectionFactory(
              java.lang.String hostname,
              java.lang.String oracle_sid,
              int portno,
              java.lang.String driver)
       throws JMSException
```

This method gets a `TopicConnectionFactory` with JDBC connection parameters. It is static and has the following parameters:

Parameter	Description
`hostname`	Name of the host running Oracle Database Advanced Queuing
`oracle_sid`	Oracle system identifier
`portno`	Port number
`driver`	JDBC driver type

Example 6-10   Getting a TopicConnectionFactory with JDBC Connection Parameters

```
String      host         = "dlsun";
String      ora_sid      = "rdbms10i"
String      driver       = "thin";
int         port         = 5521;
TopicConnectionFactory   tc_fact;

tc_fact = AQjmsFactory.getTopicConnectionFactory(host, ora_sid, port, driver);
```

Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

This method gets a `QueueConnectionFactory` or `TopicConnectionFactory` from LDAP.

Example 6-11   Getting a QueueConnectionFactory or TopicConnectionFactory in LDAP

```
Hashtable              env = new Hashtable(5, 0.75f);
DirContext             ctx;
queueConnectionFactory qc_fact;

/* the following statements set in hashtable env:
   * service provider package
   * the URL of the ldap server
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aquser1,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");

ctx = new InitialDirContext(env);
ctx =
(DirContext)ctx.lookup("cn=OracleDBConnections,cn=db1,cn=Oraclecontext,cn=acme,cn=com");
qc_fact = (queueConnectionFactory)ctx.lookup("cn=queue_conn1");
```

Getting a Queue or Topic in LDAP

This method gets a queue or topic from LDAP.

Example 6-12 Getting a Queue or Topic in LDAP

```
Hashtable              env = new Hashtable(5, 0.75f);
DirContext             ctx;
topic                  topic_1;

/* the following statements set in hashtable env:
   * service provider package
   * the URL of the ldap server
   * the distinguished name of the database server
   * the authentication method (simple)
   * the LDAP username
   * the LDAP user password
*/
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://sun-456:389");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=db1aquser1,cn=acme,cn=com");
env.put(Context.SECURITY_CREDENTIALS, "welcome");

ctx = new InitialDirContext(env);
ctx = (DirContext)ctx.lookup("cn=OracleDBQueues,cn=db1,cn=Oraclecontext,cn=acme,cn=com");
topic_1 = (topic)ctx.lookup("cn=topic_1");
```

Creating an AQ Queue Table

```
public oracle.AQ.AQQueueTable createQueueTable(
                 java.lang.String owner,
                 java.lang.String name,
                 oracle.AQ.AQQueueTableProperty property)
          throws JMSException
```

This method creates a queue table. It has the following parameters:

Parameter	Description
`owner`	Queue table owner (schema)
name	Queue table name
property	Queue table properties

If the queue table is used to hold queues, then the queue table must not be multiconsumer enabled (default). If the queue table is used to hold topics, then the queue table must be multiconsumer enabled.

CLOB, BLOB, and BFILE objects are valid attributes for an Oracle Database Advanced Queuing object type load. However, only CLOB and BLOB can be propagated using Oracle Database Advanced Queuing propagation in Oracle8*i* and after.

Note:

Currently TxEventQ queues can be created and dropped only through the `DBMS_AQADM` PL/SQL APIs.

Example 6-13 Creating a Queue Table

```
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

This section contains these topics:

- Creating a Point-to-Point Queue
- Creating a Publish/Subscribe Topic
- Creating a TxEventQ Queue for Point-to-Point Queue and Publish/Subscribe Topic

Creating a Point-to-Point Queue

```
public javax.jms.Queue createQueue(
             oracle.AQ.AQQueueTable q_table,
             java.lang.String queue_name,
             oracle.jms.AQjmsDestinationProperty dest_property)
      throws JMSException
```

This method creates a queue in a specified queue table. It has the following parameters:

Parameter	Description
`q_table`	Queue table in which the queue is to be created. The queue table must be single-consumer.
`queue_name`	Name of the queue to be created
`dest_property`	Queue properties

This method is specific to Oracle JMS. You cannot use standard Java `javax.jms.Session` objects with it. Instead, you must cast the standard type to the Oracle JMS concrete class `oracle.jms.AQjmsSession`.

Example 6-14 Creating a Point-to-Point Queue

```
QueueSession                q_sess;
AQQueueTable                q_table;
AqjmsDestinationProperty dest_prop;
Queue                       queue;

queue = ((AQjmsSession)q_sess).createQueue(q_table, "jms_q1", dest_prop);
```

Creating a Publish/Subscribe Topic

```
public javax.jms.Topic createTopic(
             oracle.AQ.AQQueueTable q_table,
             java.lang.String topic_name,
             oracle.jms.AQjmsDestinationProperty dest_property)
      throws JMSException
```

This method creates a topic in the publish/subscribe model. It has the following parameters:

Parameter	Description
`q_table`	Queue table in which the queue is to be created. The queue table must be multiconsumer.
`queue_name`	Name of the queue to be created
`dest_property`	Queue properties

This method is specific to Oracle JMS. You cannot use standard Java `javax.jms.Session` objects with it. Instead, you must cast the standard type to the Oracle JMS concrete class `oracle.jms.AQjmsSession`.

In Example 6-16, if an order cannot be filled because of insufficient inventory, then the transaction processing the order is terminated. The `bookedorders` topic is set up with `max_retries` = 4 and `retry_delay` = 12 hours.Thus, if an order is not filled up in two days, then it is moved to an exception queue.

Example 6-15 Creating a Publish/Subscribe Topic

```
TopicSession                t_sess;
AQQueueTable                q_table;
AqjmsDestinationProperty dest_prop;
Topic                       topic;

topic = ((AQjmsSessa)t_sess).createTopic(q_table, "jms_t1", dest_prop);
```

Example 6-16 Specifying Max Retries and Max Delays in Messages

```
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;
   }
```

Creating a TxEventQ Queue for Point-to-Point Queue and Publish/Subscribe Topic

AQ JMS has defined a new APIs to create and drop TxEventQ queues. There is no alter queue API in JMS. The signatures are as follows:

```
    /**
     * Create a TxEventQ queue. It also internally creates the related queue
     * objects (table, indexes) based on this name.
     *
     * @param queueName name of the queue to be created, format is schema.queueName
     *        (where the schema. is optional
     * @param isMultipleConsumer flag to indicate whether the queue is a
     *        multi-consumer or single-consumer queue
     * @return javax.jms.Destination
     * @throws JMSException if the queue could not be created
     */
    public synchronized javax.jms.Destination createJMSTransactionalEventQueue(String
queueName,
            boolean isMultipleConsumer) throws JMSException {
        return createJMSTransactionalEventQueue(queueName, isMultipleConsumer, null, 0,
null);
    }


    /**
     * Create a TxEventQ queue. It also internally creates the related queue
     * objects (table, indexes) based on this name.
     *
     * @param queueName name of the queue to be created, format is schema.queueName
     *        (where the schema. is optional
     * @param isMultipleConsumer flag to indicate whether the queue is a
     *        multi-consumer or single-consumer queue
     * @param storageClause additional storage clause
     * @param maxRetries retry count before skip the message while dequeue
     * @param comment comment for the queue
     * @return javax.jms.Destination
     * @throws JMSException if the queue could not be created
*/
public Destination createJMSTransactionalEventQueue(java.lang.String queueName,
                                         boolean isMultipleConsumer,
                                         java.lang.String storageClause,
                                         int maxRetries,
                                         java.lang.String comment)
                                  throws JMSException
```

Getting an AQ Queue Table

```
public oracle.AQ.AQQueueTable getQueueTable(java.lang.String owner,
                                            java.lang.String name)
                                     throws JMSException
```

This method gets a queue table for an AQ queue. It has the following parameters:

Parameter	Description
`owner`	Queue table owner (schema)
name	Queue table name

If the caller that opened the connection is not the owner of the queue table, then the caller must have Oracle Database Advanced Queuing enqueue/dequeue privileges on queues/topics in the queue table. Otherwise the queue table is not returned.

Example 6-17 Getting a Queue Table

```
QueueSession                q_sess;
AQQueueTable                q_table;

q_table = ((AQjmsSession)q_sess).getQueueTable(
   "boluser", "bol_ship_queue_table");
```

Granting and Revoking Privileges

This section contains these topics:

- Granting Oracle Database Advanced Queuing System Privileges
- Revoking Oracle Database Advanced Queuing System Privileges
- Granting Publish/Subscribe Topic Privileges
- Revoking Publish/Subscribe Topic Privileges
- Granting Point-to-Point Queue Privileges
- Revoking Point-to-Point Queue Privileges

Granting Oracle Database Advanced Queuing System Privileges

```
public void grantSystemPrivilege(java.lang.String privilege,
                                 java.lang.String grantee,
                                 boolean admin_option)
                          throws JMSException
```

This method grants Oracle Database Advanced Queuing system privileges to a user or role.

Parameter	Description
`privilege`	`ENQUEUE_ANY`, `DEQUEUE_ANY` or `MANAGE_ANY`
`grantee`	Grantee (user, role, or `PUBLIC`)
`admin_option`	If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles

Initially only `SYS` and `SYSTEM` can use this procedure successfully. Users granted the `ENQUEUE_ANY` privilege are allowed to enqueue messages to any queues in the database. Users granted the `DEQUEUE_ANY` privilege are allowed to dequeue messages from any queues in the database. Users granted the `MANAGE_ANY` privilege are allowed to run `DBMS_AQADM` calls on any schemas in the database.

Example 6-18 Granting Oracle Database Advanced Queuing System Privileges

```
TopicSession             t_sess;

((AQjmsSession)t_sess).grantSystemPrivilege("ENQUEUE_ANY", "scott", false);
```

Revoking Oracle Database Advanced Queuing System Privileges

```
public void revokeSystemPrivilege(java.lang.String privilege,
                                  java.lang.String grantee)
                           throws JMSException
```

This method revokes Oracle Database Advanced Queuing system privileges from a user or role. It has the following parameters:

Parameter	Description
`privilege`	`ENQUEUE_ANY`, `DEQUEUE_ANY` or `MANAGE_ANY`
`grantee`	Grantee (user, role, or `PUBLIC`)

Users granted the `ENQUEUE_ANY` privilege are allowed to enqueue messages to any queues in the database. Users granted the `DEQUEUE_ANY` privilege are allowed to dequeue messages from any queues in the database. Users granted the `MANAGE_ANY` privilege are allowed to run `DBMS_AQADM` calls on any schemas in the database.

Example 6-19 Revoking Oracle Database Advanced Queuing System Privileges

```
TopicSession             t_sess;

((AQjmsSession)t_sess).revokeSystemPrivilege("ENQUEUE_ANY", "scott");
```

Granting Publish/Subscribe Topic Privileges

```
public void grantTopicPrivilege(javax.jms.Session session,
                                java.lang.String privilege,
                                java.lang.String grantee,
                                boolean grant_option)
                         throws JMSException
```

This method grants a topic privilege in the publish/subscribe model. Initially only the queue table owner can use this procedure to grant privileges on the topic. It has the following parameters:

Parameter	Description
`session`	JMS session
`privilege`	`ENQUEUE`, `DEQUEUE`, or `ALL` (`ALL` means both.)
`grantee`	Grantee (user, role, or `PUBLIC`)
`grant_option`	If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles

Example 6-20 Granting Publish/Subscribe Topic Privileges

```
TopicSession             t_sess;
Topic                    topic;

((AQjmsDestination)topic).grantTopicPrivilege(
   t_sess, "ENQUEUE", "scott", false);
```

Revoking Publish/Subscribe Topic Privileges

```
public void revokeTopicPrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee)
                          throws JMSException
```

This method revokes a topic privilege in the publish/subscribe model. It has the following parameters:

Parameter	Description
`session`	JMS session
`privilege`	`ENQUEUE`, `DEQUEUE`, or `ALL` (`ALL` means both.)
`grantee`	Revoked grantee (user, role, or `PUBLIC`)

Example 6-21 Revoking Publish/Subscribe Topic Privileges

```
TopicSession             t_sess;
Topic                    topic;

((AQjmsDestination)topic).revokeTopicPrivilege(t_sess, "ENQUEUE", "scott");
```

Granting Point-to-Point Queue Privileges

```
public void grantQueuePrivilege(javax.jms.Session session,
                                java.lang.String privilege,
                                java.lang.String grantee,
                                boolean grant_option)
                         throws JMSException
```

This method grants a queue privilege in the point-to-point model. Initially only the queue table owner can use this procedure to grant privileges on the queue. It has the following parameters:

Parameter	Description
`session`	JMS session
`privilege`	`ENQUEUE`, `DEQUEUE`, or `ALL` (`ALL` means both.)

Parameter	Description
`grantee`	Grantee (user, role, or `PUBLIC`)
`grant_option`	If this is set to true, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles

Example 6-22 Granting Point-to-Point Queue Privileges

```
QueueSession                 q_sess;
Queue                        queue;

((AQjmsDestination)queue).grantQueuePrivilege(
   q_sess, "ENQUEUE", "scott", false);
```

Revoking Point-to-Point Queue Privileges

```
public void revokeQueuePrivilege(javax.jms.Session session,
                                 java.lang.String privilege,
                                 java.lang.String grantee)
                          throws JMSException
```

This method revokes queue privileges in the point-to-point model. Initially only the queue table owner can use this procedure to grant privileges on the queue. It has the following parameters:

Parameter	Description
`session`	JMS session
`privilege`	`ENQUEUE`, `DEQUEUE`, or `ALL` (`ALL` means both.)
`grantee`	Revoked grantee (user, role, or `PUBLIC`)

To revoke a privilege, the revoker must be the original grantor of the privilege. Privileges propagated through the `GRANT` option are revoked if the grantor privilege is also revoked.

Example 6-23 Revoking Point-to-Point Queue Privileges

```
QueueSession                 q_sess;
Queue                        queue;

((AQjmsDestination)queue).revokeQueuePrivilege(q_sess, "ENQUEUE", "scott");
```

Managing Destinations

This section contains these topics:

- Starting a Destination
- Stopping a Destination
- Altering a Destination
- Dropping a Destination

Note:

Currently TEQs can be managed only through the `DBMS_AQADM` PL/SQL APIs.

Starting a Destination

```
public void start(javax.jms.Session session,
                  boolean enqueue,
                  boolean dequeue)
           throws JMSException
```

This method starts a destination. It has the following parameters:

Parameter	Description
`session`	JMS session
`enqueue`	If set to `TRUE`, then enqueue is enabled
dequeue	If set to `TRUE`, then dequeue is enabled

Example 6-24 Starting a Destination

```
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

```
public void stop(javax.jms.Session session,
                 boolean enqueue,
                 boolean dequeue,
                 boolean wait)
          throws JMSException
```

This method stops a destination. It has the following parameters:

Parameter	Description
`session`	JMS session
`enqueue`	If set to `TRUE`, then enqueue is disabled
dequeue	If set to `TRUE`, then dequeue is disabled
`wait`	If set to true, then pending transactions on the queue/topic are allowed to complete before the destination is stopped

Example 6-25 Stopping a Destination

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).stop(t_sess, true, false);
```

Altering a Destination

```
public void alter(javax.jms.Session session,
                  oracle.jms.AQjmsDestinationProperty dest_property)
           throws JMSException
```

This method alters a destination. It has the following properties:

Parameter	Description
`session`	JMS session
dest_property	New properties of the queue or topic

Example 6-26 Altering a Destination

```
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

```
public void drop(javax.jms.Session session)
          throws JMSException
```

This method drops a destination. It has the following parameter:

Parameter	Description
`session`	JMS session

Example 6-27 Dropping a Destination

```
QueueSession q_sess;
Queue        queue;
TopicSession t_sess;
Topic        topic;

((AQjmsDestionation)queue).drop(q_sess);
((AQjmsDestionation)topic).drop(t_sess);
```

Propagation Schedules

This section contains these topics:

- Scheduling a Propagation
- Enabling a Propagation Schedule
- Altering a Propagation Schedule
- Disabling a Propagation Schedule
- Unscheduling a Propagation

Note:

TxEventQs are currently managed only through the `DBMS_AQADM` PL/SQL APIs and do not support propagation.

Scheduling a Propagation

```
public void schedulePropagation(javax.jms.Session session,
                                java.lang.String destination,
                                java.util.Date start_time,
                                java.lang.Double duration,
                                java.lang.String next_time,
                                java.lang.Double latency)
                         throws JMSException
```

This method schedules a propagation. It has the following parameters:

Parameter	Description
`session`	JMS session
`destination`	Database link of the remote database for which propagation is being scheduled. A null string means that propagation is scheduled for all subscribers in the database of the topic.
`start_time`	Time propagation starts
`duration`	Duration of propagation
`next_time`	Next time propagation starts
`latency`	Latency in seconds that can be tolerated. Latency is the difference between the time a message was enqueued and the time it was propagated.

If a message has multiple recipients at the same destination in either the same or different queues, then it is propagated to all of them at the same time.

Example 6-28   Scheduling a Propagation

```
TopicSession t_sess;
Topic        topic;

((AQjmsDestination)topic).schedulePropagation(
   t_sess, null, null, null, null, new Double(0));
```

Enabling a Propagation Schedule

```
public void enablePropagationSchedule(javax.jms.Session session,
                                      java.lang.String destination)
                               throws JMSException
```

This method enables a propagation schedule. It has the following parameters:

Parameter	Description
`session`	JMS session
`destination`	Database link of the destination database. A null string means that propagation is to the local database.

Example 6-29   Enabling a Propagation Schedule

```
TopicSession            t_sess;
Topic                   topic;

((AQjmsDestination)topic).enablePropagationSchedule(t_sess, "dbs1");
```

Altering a Propagation Schedule

```
public void alterPropagationSchedule(javax.jms.Session session,
                                     java.lang.String destination,
                                     java.lang.Double duration,
                                     java.lang.String next_time,
                                     java.lang.Double latency)
                              throws JMSException
```

This method alters a propagation schedule. It has the following parameters:

Parameter	Description
`session`	JMS session
`destination`	Database link of the remote database for which propagation is being scheduled. A null string means that propagation is scheduled for all subscribers in the database of the topic.
`duration`	Duration of propagation
`next_time`	Next time propagation starts
`latency`	Latency in seconds that can be tolerated. Latency is the difference between the time a message was enqueued and the time it was propagated.

Example 6-30 Altering a Propagation Schedule

```
TopicSession              t_sess;
Topic                     topic;

((AQjmsDestination)topic).alterPropagationSchedule(
   t_sess, null, 30, null, new Double(30));
```

Disabling a Propagation Schedule

```
public void disablePropagationSchedule(javax.jms.Session session,
                                       java.lang.String destination)
                                throws JMSException
```

This method disables a propagation schedule. It has the following parameters:

Parameter	Description
`session`	JMS session
`destination`	Database link of the destination database. A null string means that propagation is to the local database.

Example 6-31 Disabling a Propagation Schedule

```
TopicSession              t_sess;
Topic                     topic;

((AQjmsDestination)topic).disablePropagationSchedule(t_sess, "dbs1");
```

Unscheduling a Propagation

```
public void unschedulePropagation(javax.jms.Session session,
                                  java.lang.String destination)
                           throws JMSException
```

This method unschedules a previously scheduled propagation. It has the following parameters:

Parameter	Description
`session`	JMS session
`destination`	Database link of the destination database. A null string means that propagation is to the local database.

Example 6-32 Unscheduling a Propagation

```
TopicSession   t_sess;
Topic          topic;

((AQjmsDestination)topic).unschedulePropagation(t_sess, "dbs1");
```

Oracle Java Message Service Point-to-Point

The following topics describe the components of the Oracle Database Advanced Queuing (AQ) Java Message Service (JMS) operational interface that are specific to point-to-point operations. Components that are shared by point-to-point and publish/subscribe are described in Oracle Java Message Service Shared Interfaces.

- Creating a Connection with User Name/Password
- Creating a Connection with Default ConnectionFactory Parameters
- Creating a QueueConnection with User Name/Password
- Creating a QueueConnection with an Open JDBC Connection
- Creating a QueueConnection with Default ConnectionFactory Parameters
- Creating a QueueConnection with an Open OracleOCIConnectionPool
- Creating a Session
- Creating a QueueSession
- Creating a QueueSender
- Sending Messages Using a QueueSender with Default Send Options
- Sending Messages Using a QueueSender by Specifying Send Options
- Creating a QueueBrowser for Standard JMS Type Messages
- Creating a QueueBrowser for Standard JMS Type Messages_ Locking Messages
- Creating a QueueBrowser for Oracle Object Type Messages
- Creating a QueueBrowser for Oracle Object Type Messages_ Locking Messages
- Creating a QueueReceiver for Standard JMS Type Messages
- Creating a QueueReceiver for Oracle Object Type Messages

Creating a Connection with User Name/Password

```
public javax.jms.Connection createConnection(
              java.lang.String username,
              java.lang.String password)
      throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with the specified user name and password. This method is new and supports JMS version 1.1 specifications. It has the following parameters:

Parameter	Description
`username`	Name of the user connecting to the database for queuing
`password`	Password for creating the connection to the server

Creating a Connection with Default ConnectionFactory Parameters

```
public javax.jms.Connection createConnection()
      throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with default ConnectionFactory parameters. This method is new and supports JMS version 1.1 specifications. If the `ConnectionFactory` properties do not contain a default user name and password, then it throws a JMSException.

Creating a QueueConnection with User Name/Password

```
public javax.jms.QueueConnection createQueueConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

This method creates a queue connection with the specified user name and password. It has the following parameters:

Parameter	Description
`username`	Name of the user connecting to the database for queuing
`password`	Password for creating the connection to the server

Example 6-33 Creating a QueueConnection with User Name/Password

```
QueueConnectionFactory qc_fact = AQjmsFactory.getQueueConnectionFactory(
   "sun123", "oratest", 5521, "thin");
QueueConnection qc_conn = qc_fact.createQueueConnection("jmsuser", "jmsuser");
```

Creating a QueueConnection with an Open JDBC Connection

```
public static javax.jms.QueueConnection createQueueConnection(
   java.sql.Connection jdbc_connection)
   throws JMSException
```

This method creates a queue connection with an open JDBC connection. It is static and has the following parameter:

Parameter	Description
`jdbc_connection`	Valid open connection to the database

The method in Example 6-34 can be used if the user wants to use an existing JDBC connection (say from a connection pool) for JMS operations. In this case JMS does not open a new connection, but instead uses the supplied JDBC connection to create the JMS `QueueConnection` object.

The method in Example 6-35 is the only way to create a JMS `QueueConnection` when using JMS from a Java stored procedures inside the database (JDBC Server driver)

Example 6-34 Creating a QueueConnection with an Open JDBC Connection

```
Connection db_conn;     /* previously opened JDBC connection */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(
           db_conn);
```

Example 6-35 Creating a QueueConnection from a Java Procedure Inside Database

```
OracleDriver ora = new OracleDriver();
QueueConnection qc_conn =
AQjmsQueueConnectionFactory.createQueueConnection(ora.defaultConnection());
```

Creating a QueueConnection with Default ConnectionFactory Parameters

```
public javax.jms.QueueConnection createQueueConnection()
      throws JMSException
```

This method creates a queue connection with default ConnectionFactory parameters. If the queue connection factory properties do not contain a default user name and password, then it throws a JMSException.

Creating a QueueConnection with an Open OracleOCIConnectionPool

```
public static javax.jms.QueueConnection createQueueConnection(
          oracle.jdbc.pool.OracleOCIConnectionPool cpool)
   throws JMSException
```

This method creates a queue connection with an open `OracleOCIConnectionPool`. It is static and has the following parameter:

Parameter	Description
`cpool`	Valid open OCI connection pool to the database

The method in Example 6-36 can be used if the user wants to use an existing `OracleOCIConnectionPool` instance for JMS operations. In this case JMS does not open an new `OracleOCIConnectionPool` instance, but instead uses the supplied `OracleOCIConnectionPool` instance to create the JMS QueueConnection object.

Example 6-36 Creating a QueueConnection with an Open OracleOCIConnectionPool

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
QueueConnection qc_conn = AQjmsQueueConnectionFactory.createQueueConnection(cpool);
```

Creating a Session

```
public javax.jms.Session createSession(boolean transacted,
                                       int ack_mode)
                                throws JMSException
```

This method creates a `Session`, which supports both point-to-point and publish/subscribe operations. This method is new and supports JMS version 1.1 specifications. Transactional and nontransactional sessions are supported. It has the following parameters:

Parameter	Description
`transacted`	If set to true, then the session is transactional
`ack_mode`	Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Creating a QueueSession

```
public javax.jms.QueueSession createQueueSession(
   boolean transacted, int ack_mode)
        throws JMSException
```

This method creates a `QueueSession`. Transactional and nontransactional sessions are supported. It has the following parameters:

Parameter	Description
`transacted`	If set to true, then the session is transactional
`ack_mode`	Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Example 6-37 Creating a Transactional QueueSession

```
QueueConnection qc_conn;
QueueSession  q_sess = qc_conn.createQueueSession(true, 0);
```

Creating a QueueSender

```
public javax.jms.QueueSender createSender(javax.jms.Queue queue)
                                   throws JMSException
```

This method creates a `QueueSender`. If a sender is created without a default queue, then the destination queue must be specified on every send operation. It has the following parameter:

Parameter	Description
`queue`	Name of destination queue

Sending Messages Using a QueueSender with Default Send Options

```
public void send(javax.jms.Queue queue,
                 javax.jms.Message message)
          throws JMSException
```

This method sends a message using a `QueueSender` with default send options. This operation uses default values for message `priority` (`1`) and `timeToLive` (`infinite`). It has the following parameters:

Parameter	Description
`queue`	Queue to send this message to
`message`	Message to send

If the `QueueSender` has been created with a default queue, then the queue parameter may not necessarily be supplied in the `send()` call. If a queue is specified in the `send()` operation, then this value overrides the default queue of the `QueueSender`.

If the `QueueSender` has been created without a default queue, then the queue parameter must be specified in every `send()` call.

Example 6-38 Creating a Sender to Send Messages to Any Queue

```
/* Create a sender to send messages to any queue */
QueueSession  jms_sess;
QueueSender  sender1;
TextMessage  message;
sender1 = jms_sess.createSender(null);
sender1.send(queue, message);
```

Example 6-39 Creating a Sender to Send Messages to a Specific Queue

```
/* Create a sender to send messages to a specific queue */
QueueSession jms_sess;
QueueSender sender2;
Queue   billed_orders_que;
TextMessage  message;
sender2 = jms_sess.createSender(billed_orders_que);
sender2.send(queue, message);
```

Sending Messages Using a QueueSender by Specifying Send Options

```
public void send(javax.jms.Queue queue,
                 javax.jms.Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive)
          throws JMSException
```

This method sends messages using a `QueueSender` by specifying send options. It has the following parameters:

Parameter	Description
`queue`	Queue to send this message to
`message`	Message to send
`deliveryMode`	Delivery mode to use
`priority`	Priority for this message
`timeToLive`	Message lifetime in milliseconds (zero is unlimited)

If the `QueueSender` has been created with a default queue, then the queue parameter may not necessarily be supplied in the `send()` call. If a queue is specified in the `send()` operation, then this value overrides the default queue of the `QueueSender`.

If the `QueueSender` has been created without a default queue, then the queue parameter must be specified in every `send()` call.

Example 6-40 Sending Messages Using a QueueSender by Specifying Send Options 1

```
/* Create a sender to send messages to any queue */
/* Send a message to new_orders_que with priority 2 and timetoLive 100000
   milliseconds */
QueueSession  jms_sess;
QueueSender  sender1;
TextMessage mesg;
Queue   new_orders_que
sender1 = jms_sess.createSender(null);
sender1.send(new_orders_que, mesg, DeliveryMode.PERSISTENT, 2, 100000);
```

Example 6-41 Sending Messages Using a QueueSender by Specifying Send Options 2

```
/* Create a sender to send messages to a specific queue */
/* Send a message with priority 1 and timetoLive 400000 milliseconds */
QueueSession jms_sess;
QueueSender sender2;
Queue   billed_orders_que;
TextMessage mesg;
sender2 = jms_sess.createSender(billed_orders_que);
sender2.send(mesg, DeliveryMode.PERSISTENT, 1, 400000);
```

Creating a QueueBrowser for Standard JMS Type Messages

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector)
                                     throws JMSException
```

This method creates a `QueueBrowser` for queues with text, stream, objects, bytes or MapMessage message bodies. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered

Use methods in `java.util.Enumeration` to go through list of messages.

See Also:

"MessageSelector"

Example 6-42 Creating a QueueBrowser Without a Selector

```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
browser = jms_session.createBrowser(queue);
```

Example 6-43 Creating a QueueBrowser With a Specified Selector

```
/* Create a browser for queues with a specified selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
/* create a Browser to look at messages with correlationID = RUSH  */
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'");
```

Creating a QueueBrowser for Standard JMS Type Messages, Locking Messages

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            boolean locked)
                                     throws JMSException
```

This method creates a `QueueBrowser` for queues with TextMessage, StreamMessage, ObjectMessage, BytesMessage, or MapMessage message bodies, locking messages while browsing. Locked messages cannot be removed by other consumers until the browsing session ends the transaction. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered
`locked`	If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)

Example 6-44 Creating a QueueBrowser Without a Selector, Locking Messages

```
/* Create a browser without a selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
browser = jms_session.createBrowser(queue, null, true);
```

Example 6-45 Creating a QueueBrowser With a Specified Selector, Locking Messages

```
/* Create a browser for queues with a specified selector */
QueueSession    jms_session;
QueueBrowser    browser;
Queue           queue;
/* create a Browser to look at messages with
correlationID = RUSH in lock mode */
browser = jms_session.createBrowser(queue, "JMSCorrelationID = 'RUSH'", true);
```

Creating a QueueBrowser for Oracle Object Type Messages

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            java.lang.Object payload_factory)
                                     throws JMSException
```

This method creates a `QueueBrowser` for queues of Oracle object type messages. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered
payload_factory	`CustomDatumFactory` or `ORADataFactory` for the java class that maps to the Oracle ADT

The `CustomDatumFactory` for a particular java class that maps to the SQL object payload can be obtained using the `getFactory` static method.

Note:

`CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.

Assume the queue `test_queue` has payload of type `SCOTT.EMPLOYEE` and the java class that is generated by Jpublisher for this Oracle object type is called Employee. The Employee class implements the `CustomDatum` interface. The `CustomDatumFactory` for this class can be obtained by using the `Employee.getFactory()` method.

Note:

TEQs do not support Object Type messages

See Also:

"MessageSelector"

Example 6-46 Creating a QueueBrowser for ADTMessages

```
/* Create a browser for a Queue with AdtMessage messages of type EMPLOYEE*/
QueueSession jms_session
QueueBrowser browser;
Queue        test_queue;
browser = ((AQjmsSession)jms_session).createBrowser(test_queue,
                                                   "corrid='EXPRESS'",
                                                    Employee.getFactory());
```

Creating a QueueBrowser for Oracle Object Type Messages, Locking Messages

```
public javax.jms.QueueBrowser createBrowser(javax.jms.Queue queue,
                                            java.lang.String messageSelector,
                                            java.lang.Object payload_factory,
                                            boolean locked)
                                     throws JMSException
```

This method creates a `QueueBrowser` for queues of Oracle object type messages, locking messages while browsing. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered
payload_factory	`CustomDatumFactory` or `ORADataFactory` for the java class that maps to the Oracle ADT
`locked`	If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)

Note:

`CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.

Note:

TxEventQ queues do not support Object Type messages

Example 6-47 Creating a QueueBrowser for AdtMessages, Locking Messages

```
/* Create a browser for a Queue with AdtMessage messagess of type EMPLOYEE* in lock mode/
QueueSession jms_session
QueueBrowser browser;
Queue        test_queue;
browser = ((AQjmsSession)jms_session).createBrowser(test_queue,
                                                    null,
                                                    Employee.getFactory(),
                                                    true);
```

Creating a QueueReceiver for Standard JMS Type Messages

```
public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
                                              java.lang.String messageSelector)
                                       throws JMSException
```

This method creates a `QueueReceiver` for queues of standard JMS type messages. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered

See Also:

"MessageSelector"

Example 6-48 Creating a QueueReceiver Without a Selector

```
/* Create a receiver without a selector */
QueueSession    jms_session
QueueReceiver   receiver;
Queue           queue;
receiver = jms_session.createReceiver(queue);
```

Example 6-49 Creating a QueueReceiver With a Specified Selector

```
/* Create a receiver for queues with a specified selector */
QueueSession    jms_session;
QueueReceiver   receiver;
Queue           queue;
/* create Receiver to receive messages with correlationID starting with EXP  */
browser = jms_session.createReceiver(queue, "JMSCorrelationID LIKE 'EXP%'");
```

Creating a QueueReceiver for Oracle Object Type Messages

```
public javax.jms.QueueReceiver createReceiver(javax.jms.Queue queue,
                                              java.lang.String messageSelector,
                                              java.lang.Object payload_factory)
                                       throws JMSException
```

This method creates a `QueueReceiver` for queues of Oracle object type messages. It has the following parameters:

Parameter	Description
`queue`	Queue to access
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered
payload_factory	`CustomDatumFactory` or `ORADataFactory` for the java class that maps to the Oracle ADT

The `CustomDatumFactory` for a particular java class that maps to the SQL object type payload can be obtained using the `getFactory` static method.

Note:

`CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.

Assume the queue `test_queue` has payload of type `SCOTT.EMPLOYEE` and the java class that is generated by Jpublisher for this Oracle object type is called Employee. The Employee class implements the `CustomDatum` interface. The `ORADataFactory` for this class can be obtained by using the Employee.getFactory() method.

Note:

TxEventQ queues do not support Object Type messages

See Also:

"MessageSelector"

Example 6-50 Creating a QueueReceiver for AdtMessage Messages

```
/* Create a receiver for a Queue with AdtMessage messages of type EMPLOYEE*/
QueueSession jms_session
QueueReceiver receiver;
Queue         test_queue;
browser = ((AQjmsSession)jms_session).createReceiver(
                 test_queue,
                "JMSCorrelationID = 'MANAGER',
                 Employee.getFactory());
```

Oracle Java Message Service Publish/Subscribe

The following topics describe the components of the Oracle Database Advanced Queuing (AQ) Java Message Service (JMS) operational interface that are specific to publish/subscribe operations. Components that are shared by point-to-point and publish/subscribe are described in Oracle Java Message Service Shared Interfaces.

- Creating a Connection with User Name/Password
- Creating a Connection with Default ConnectionFactory Parameters
- Creating a TopicConnection with User Name/Password
- Creating a TopicConnection with Open JDBC Connection
- Creating a TopicConnection with an Open OracleOCIConnectionPool
- Creating a Session
- Creating a TopicSession
- Creating a TopicPublisher
- Publishing Messages with Minimal Specification
- Publishing Messages Specifying Topic
- Publishing Messages Specifying Delivery Mode_ Priority_ and TimeToLive
- Publishing Messages Specifying a Recipient List
- Creating a DurableSubscriber for a JMS Topic Without Selector
- Creating a DurableSubscriber for a JMS Topic with Selector
- Creating a DurableSubscriber for an Oracle Object Type Topic Without Selector
- Creating a DurableSubscriber for an Oracle Object Type Topic with Selector
- Specifying Transformations for Topic Subscribers

Creating a Connection with User Name/Password

```
public javax.jms.Connection createConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with the specified user name and password. This method is new and supports JMS version 1.1 specifications. It has the following parameters:

Parameter	Description
`username`	Name of the user connecting to the database for queuing
`password`	Password for creating the connection to the server

Creating a Connection with Default ConnectionFactory Parameters

```
public javax.jms.Connection createConnection()
      throws JMSException
```

This method creates a connection supporting both point-to-point and publish/subscribe operations with default ConnectionFactory parameters. This method is new and supports JMS version 1.1 specifications. If the `ConnectionFactory` properties do not contain a default user name and password, then it throws a JMSException.

Creating a TopicConnection with User Name/Password

```
public javax.jms.TopicConnection createTopicConnection(
             java.lang.String username,
             java.lang.String password)
      throws JMSException
```

This method creates a `TopicConnection` with the specified user name and password. It has the following parameters:

Parameter	Description
`username`	Name of the user connecting to the database for queuing
`password`	Password for creating the connection to the server

Example 6-51 Creating a TopicConnection with User Name/Password

```
TopicConnectionFactory tc_fact = AQjmsFactory.getTopicConnectionFactory("sun123",
"oratest", 5521, "thin");
/* Create a TopicConnection using a username/password */
TopicConnection tc_conn = tc_fact.createTopicConnection("jmsuser", "jmsuser");
```

Creating a TopicConnection with Open JDBC Connection

```
public static javax.jms.TopicConnection createTopicConnection(
              java.sql.Connection jdbc_connection)
       throws JMSException
```

This method creates a `TopicConnection` with open JDBC connection. It has the following parameter:

Parameter	Description
`jdbc_connection`	Valid open connection to database

Example 6-52 Creating a TopicConnection with Open JDBC Connection

```
Connection db_conn;   /*previously opened JDBC connection */
TopicConnection tc_conn =
AQjmsTopicConnectionFactory createTopicConnection(db_conn);
```

Example 6-53 Creating a TopicConnection with New JDBC Connection

```
OracleDriver ora = new OracleDriver();
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(ora.defaultConnection());
```

Creating a TopicConnection with an Open OracleOCIConnectionPool

```
public static javax.jms.TopicConnection createTopicConnection(
              oracle.jdbc.pool.OracleOCIConnectionPool cpool)
       throws JMSException
```

This method creates a `TopicConnection` with an open `OracleOCIConnectionPool`. It is static and has the following parameter:

Parameter	Description
`cpool`	Valid open OCI connection pool to the database

Example 6-54 Creating a TopicConnection with Open OracleOCIConnectionPool

```
OracleOCIConnectionPool cpool; /* previously created OracleOCIConnectionPool */
TopicConnection tc_conn =
AQjmsTopicConnectionFactory.createTopicConnection(cpool);
```

Creating a Session

```
public javax.jms.Session createSession(boolean transacted,
                                                  int ack_mode)
                                          throws JMSException
```

This method creates a `Session` supporting both point-to-point and publish/subscribe operations. It is new and supports JMS version 1.1 specifications. It has the following parameters:

Parameter	Description
`transacted`	If set to true, then the session is transactional
`ack_mode`	Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Creating a TopicSession

```
public javax.jms.TopicSession createTopicSession(boolean transacted,
                                                  int ack_mode)
                                          throws JMSException
```

This method creates a `TopicSession`. It has the following parameters:

Parameter	Description
`transacted`	If set to true, then the session is transactional
`ack_mode`	Indicates whether the consumer or the client will acknowledge any messages it receives. It is ignored if the session is transactional. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Example 6-55   Creating a TopicSession

```
TopicConnection tc_conn;
TopicSession t_sess = tc_conn.createTopicSession(true,0);
```

Creating a TopicPublisher

```
public javax.jms.TopicPublisher createPublisher(javax.jms.Topic topic)
                                         throws JMSException
```

This method creates a `TopicPublisher`. It has the following parameter:

Parameter	Description
`topic`	Topic to publish to, or null if this is an unidentified producer

Publishing Messages with Minimal Specification

```
public void publish(javax.jms.Message message)
             throws JMSException
```

This method publishes a message with minimal specification. It has the following parameter:

Parameter	Description
`message`	Message to send

The `TopicPublisher` uses the default values for message `priority` (`1`) and `timeToLive` (`infinite`).

Example 6-56 Publishing Without Specifying Topic

```
/* Publish without specifying topic */
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                       myport = 5521;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME",
          "MYSID",
           myport,
          "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
/* create TopicSession */
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* get shipped orders topic */
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE",
          "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* create TextMessage */
TextMessage     jms_sess.createTextMessage();
/* publish without specifying the topic */
publisher1.publish(text_message);
```

Example 6-57 Publishing Specifying Correlation and Delay

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                       myport = 5521;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME",
          "MYSID",
           myport,
          "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE",
          "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* Create TextMessage */
TextMessage     jms_sess.createTextMessage();
/* 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 Messages Specifying Topic

```
public void publish(javax.jms.Topic topic, javax.jms.Message message)
             throws JMSException
```

This method publishes a message specifying the topic. It has the following parameters:

Parameter	Description
`topic`	Topic to publish to
`message`	Message to send

If the `TopicPublisher` has been created with a default topic, then the `topic` parameter may not be specified in the `publish()` call. If a topic is specified, then that value overrides the default in the `TopicPublisher`. If the `TopicPublisher` has been created without a default topic, then the topic must be specified with the `publish()` call.

Example 6-58 Publishing Specifying Topic

```
/* Publish specifying topic */
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                       myport = 5521;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          'MYHOSTNAME', 'MYSID', myport, 'oci8');
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* create TopicPublisher */
publisher1 = jms_sess.createPublisher(null);
/* get topic object */
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          'WS', 'Shipped_Orders_Topic');
/* create text message */
TextMessage     jms_sess.createTextMessage();
/* publish specifying the topic */
publisher1.publish(shipped_orders, text_message);
```

Publishing Messages Specifying Delivery Mode, Priority, and TimeToLive

```
public void publish(javax.jms.Topic topic,
                    javax.jms.Message message,
                    oracle.jms.AQjmsAgent[] recipient_list,
                    int deliveryMode,
                    int priority,
                    long timeToLive)
             throws JMSException
```

This method publishes a message specifying delivery mode, priority and `TimeToLive`. It has the following parameters:

Parameter	Description
`topic`	Topic to which to publish the message (overrides the default topic of the `MessageProducer`)
`message`	Message to publish
`recipient_list`	List of recipients to which the message is published. Recipients are of type `AQjmsAgent`.
`deliveryMode`	`PERSISTENT` or `NON_PERSISTENT` (only `PERSISTENT` is supported in this release)
`priority`	Priority for this message
`timeToLive`	Message lifetime in milliseconds (zero is unlimited)

Example 6-59 Publishing Specifying Priority and 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 TextMessage */
TextMessage     jms_sess.createTextMessage();
/* Publish  message with priority 1 and time to live 200 seconds */
publisher1.publish(text_message, DeliveryMode.PERSISTENT, 1, 200000);
```

Publishing Messages Specifying a Recipient List

```
public void publish(javax.jms.Message message,
                    oracle.jms.AQjmsAgent[] recipient_list)
             throws JMSException
```

This method publishes a message specifying a recipient list overriding topic subscribers. It has the following parameters:

Parameter	Description
`message`	Message to publish
`recipient_list`	List of recipients to which the message is published. Recipients are of type `AQjmsAgent`.

Example 6-60 Publishing Specifying a Recipient List Overriding Topic Subscribers

```
/* Publish specifying priority and timeToLive */
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
```

```
int                         myport = 5521;
AQjmsAgent[]                recipList;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);
/* create TextMessage */
TextMessage     jms_sess.createTextMessage();
/* create two receivers */
recipList = new AQjmsAgent[2];
recipList[0] = new AQjmsAgent(
          "ES", "ES.shipped_orders_topic", AQAgent.DEFAULT_AGENT_PROTOCOL);
recipList[1] = new AQjmsAgent(
          "WS", "WS.shipped_orders_topic", AQAgent.DEFAULT_AGENT_PROTOCOL);
/* publish  message specifying a recipient list */
publisher1.publish(text_message, recipList);
```

Creating a DurableSubscriber for a JMS Topic Without Selector

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name)
      throws JMSException
```

This method creates a DurableSubscriber for a JMS topic without selector. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to subscribe to
`subs_name`	Name used to identify this subscription

Exclusive Access to Topics

`CreateDurableSubscriber()` and `Unsubscribe()` both require exclusive access to their target topics. If there are pending JMS `send()`, `publish()`, or `receive()` operations on the same topic when these calls are applied, then exception ORA - 4020 is raised. There are two solutions to the problem:

- Limit calls to `createDurableSubscriber()` and `Unsubscribe()` to the setup or cleanup phase when there are no other JMS operations pending on the topic. That makes sure that the required resources are not held by other JMS operational calls.
- Call `TopicSession.commit` before calling `createDurableSubscriber()` or `Unsubscribe()`.

Example 6-61 Creating a Durable Subscriber for a JMS Topic Without 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 durable subscriber on the shipped_orders topic*/
subscriber1 = jms_sess.createDurableSubscriber(
          shipped_orders,
         'WesternShipping');
```

Creating a DurableSubscriber for a JMS Topic with Selector

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name,
             java.lang.String messageSelector,
             boolean noLocal)
      throws JMSException
```

This method creates a durable subscriber for a JMS topic with selector. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to subscribe to
`subs_name`	Name used to identify this subscription
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`noLocal`	If set to true, then it inhibits the delivery of messages published by its own connection

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a different `messageSelector`. An unsubscribe call is needed to end the subscription to the topic.

See Also:

- "Exclusive Access to Topics"
- "MessageSelector"

Example 6-62 Creating a Durable Subscriber for a JMS Topic With 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);
```

Creating a DurableSubscriber for an Oracle Object Type Topic Without Selector

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name,
             java.lang.Object payload_factory)
      throws JMSException
```

This method creates a durable subscriber for an Oracle object type topic without selector. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to subscribe to
`subs_name`	Name used to identify this subscription
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support object type messages.

See Also:

"Exclusive Access to Topics"

Example 6-63 Creating a Durable Subscriber for an Oracle Object Type Topic Without Selector

```
/* 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 DurableSubscriber for an Oracle Object Type Topic with Selector

```
public javax.jms.TopicSubscriber createDurableSubscriber(
             javax.jms.Topic topic,
             java.lang.String subs_name,
             java.lang.String messageSelector,
             boolean noLocal,
             java.lang.Object payload_factory)
      throws JMSException
```

This method creates a durable subscriber for an Oracle object type topic with selector. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to subscribe to
`subs_name`	Name used to identify this subscription
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`noLocal`	If set to true, then it inhibits the delivery of messages published by its own connection
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support object yype messages.

See Also:

"Exclusive Access to Topics"

Example 6-64 Creating a Durable Subscriber for an Oracle Object Type Topic With Selector

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicSubscriber           subscriber1;
Topic                     shipped_orders;
int                       myport = 5521;
AQjmsAgent[]              recipList;
/* the java mapping of the oracle object type created by J Publisher */
ADTMessage                message;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE", "Shipped_Orders_Topic");
/* create a subscriber, specifying correct CustomDatumFactory and selector */
subscriber1 = jms_sess.createDurableSubscriber(
          shipped_orders, "WesternShipping",
         "priority > 1 and tab.user_data.region like 'WESTERN %'", false,
          ADTMessage.getFactory());
```

Specifying Transformations for Topic Subscribers

A transformation can be supplied when sending/publishing a message to a queue/topic. The transformation is applied before putting the message into the queue/topic.

The application can specify a transformation using the `setTransformation` interface in the `AQjmsQueueSender` and `AQjmsTopicPublisher` interfaces.

A transformation can also be specified when creating topic subscribers using the `CreateDurableSubscriber()` call. The transformation is applied to the retrieved message before returning it to the subscriber. If the subscriber specified in the `CreateDurableSubscriber()` call already exists, then its transformation is set to the specified transformation.

Example 6-65 Sending Messages to a Destination Using a Transformation

Suppose that the orders that are processed by the order entry application should be published to `WS_bookedorders_topic`. The transformation OE2WS (defined in the previous section) is supplied so that the messages are inserted into the topic in the correct format.

```
public void ship_bookedorders(
   TopicSession jms_session,
       AQjmsADTMessage adt_message)
{
    TopicPublisher  publisher;
    Topic           topic;

    try
    {
      /* get a handle to the WS_bookedorders_topic */
          topic = ((AQjmsSession)jms_session).getTopic("WS", "WS_bookedorders_topic");
      publisher = jms_session.createPublisher(topic);

      /* set the transformation in the publisher */
      ((AQjmsTopicPublisher)publisher).setTransformation("OE2WS");
      publisher.publish(topic, adt_message);
```

```
        }
        catch (JMSException ex)
        {
           System.out.println("Exception :" ex);
    }
}
```

Example 6-66 Specifying Transformations for Topic Subscribers

The Western Shipping application subscribes to the OE_bookedorders_topic with the transformation `OE2WS`. This transformation is applied to the messages and the returned message is of Oracle object type `WS.WS_orders`.

Suppose that the WSOrder java class has been generated by Jpublisher to map to the Oracle object `WS.WS_order`:

```
public AQjmsAdtMessage retrieve_bookedorders(TopicSession jms_session)
{
    TopicSubscriber     subscriber;
    Topic               topic;
    AQjmsAdtMessage     msg = null;

    try
    {
      /* get a handle to the OE_bookedorders_topic */
      topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");

      /* create a subscriber with the transformation OE2WS */
      subs = ((AQjmsSession)jms_session).createDurableSubscriber(
         topic, 'WShip', null, false, WSOrder.getFactory(), "OE2WS");
      msg = subscriber.receive(10);
    }
    catch (JMSException ex)
    {
        System.out.println("Exception :" ex);
    }
    return (AQjmsAdtMessage)msg;
}
```

Creating a Remote Subscriber for JMS Messages

```
public void createRemoteSubscriber(javax.jms.Topic topic,
                                   oracle.jms.AQjmsAgent remote_subscriber,
                                   java.lang.String messageSelector)
                            throws JMSException
```

This method creates a remote subscriber for topics of JMS messages. It has the following parameters:

Parameter	Description
`topic`	Topic to subscribe to
`remote_subscriber`	`AQjmsAgent` that refers to the remote subscriber
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.

Oracle Database Advanced Queuing allows topics to have remote subscribers, for example, subscribers at other topics in the same or different database. In order to use remote subscribers, you must set up propagation between the local and remote topic.

Remote subscribers can be a specific consumer at the remote topic or all subscribers at the remote topic. A remote subscriber is defined using the `AQjmsAgent` structure. An `AQjmsAgent` consists of a name and address. The name refers to the `consumer_name` at the remote topic. The address refers to the remote topic. Its syntax is `schema.topic_name[@dblink]`.

To publish messages to a particular consumer at the remote topic, the `subscription_name` of the recipient at the remote topic must be specified in the name field of `AQjmsAgent`, and the remote topic must be specified in the address field. To publish messages to all subscribers of the remote topic, the name field of `AQjmsAgent` must be set to null.

Note:

TxEventQ queues do not support remote subscribers.

See Also:

"MessageSelector"

Example 6-67 Creating a Remote Subscriber for Topics of JMS Messages

```
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 Oracle Object Type Messages

```
public void createRemoteSubscriber(javax.jms.Topic topic,
                                   oracle.jms.AQjmsAgent remote_subscriber,
                                   java.lang.String messageSelector,
                                   java.lang.Object payload_factory)
                            throws JMSException
```

This method creates a remote subscriber for topics of Oracle object type messages. It has the following parameters:

Parameter	Description
`topic`	Topic to subscribe to
`remote_subscriber`	`AQjmsAgent` that refers to the remote subscriber
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support remote subscribers or object type messages.

Oracle Database Advanced Queuing allows topics to have remote subscribers, for example, subscribers at other topics in the same or different database. In order to use remote subscribers, you must set up propagation between the local and remote topic.

Remote subscribers can be a specific consumer at the remote topic or all subscribers at the remote topic. A remote subscriber is defined using the `AQjmsAgent` structure. An `AQjmsAgent` consists of a name and address. The name refers to the `consumer_name` at the remote topic. The address refers to the remote topic. Its syntax is `schema.topic_name[@dblink]`.

To publish messages to a particular consumer at the remote topic, the `subscription_name` of the recipient at the remote topic must be specified in the name field of `AQjmsAgent`, and the remote topic must be specified in the address field. To publish messages to all subscribers of the remote topic, the name field of `AQjmsAgent` must be set to null.

Note:

AQ does not support the use of multiple dblink to the same destination. As a workaround, use a single database link for each destination.

See Also:

"MessageSelector"

Example 6-68 Creating a Remote Subscriber for Topics of Oracle Object Type Messages

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
```

```
TopicSubscriber            subscriber1;
Topic                      shipped_orders;
int                        my[port = 5521;
AQjmsAgent                 remoteAgent;
ADTMessage                 message;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
/* create TopicSession */
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
/* get the Shipped order topic */
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE", "Shipped_Orders_Topic");
/* create a remote agent */
remoteAgent = new AQjmsAgent("WesternRegion", "WS.shipped_orders_topic", null);
/* create a remote subscriber  with null selector*/
subscriber1 = ((AQjmsSession)jms_sess).createRemoteSubscriber(
          shipped_orders, remoteAgent, null, message.getFactory);
```

Specifying Transformations for Remote Subscribers

Oracle Database Advanced Queuing allows a remote subscriber, that is a subscriber at another database, to subscribe to a topic.

Transformations can be specified when creating remote subscribers using the `createRemoteSubscriber()` call. This enables propagation of messages between topics of different formats. When a message published at a topic meets the criterion of a remote subscriber, Oracle Database Advanced Queuing automatically propagates the message to the queue/topic at the remote database specified for the remote subscriber. If a transformation is also specified, then Oracle Database Advanced Queuing applies the transformation to the message before propagating it to the queue/topic at the remote database.

Note:

TxEventQ queues do not support remote subscribers.

Example 6-69 Specifying Transformations for Remote Subscribers

A remote subscriber is created at the OE.OE_bookedorders_topic so that messages are automatically propagated to the WS.WS_bookedorders_topic. The transformation OE2WS is specified when creating the remote subscriber so that the messages reaching the WS_bookedorders_topic have the correct format.

Suppose that the WSOrder java class has been generated by Jpublisher to map to the Oracle object `WS.WS_order`

```
public void create_remote_sub(TopicSession jms_session)
{
   AQjmsAgent          subscriber;
   Topic               topic;

   try
   {
     /* get a handle to the OE_bookedorders_topic */
     topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");
     subscriber = new AQjmsAgent("WShip", "WS.WS_bookedorders_topic");
```

```
      ((AQjmsSession )jms_session).createRemoteSubscriber(
         topic, subscriber, null, WSOrder.getFactory(),"OE2WS");
    }
    catch (JMSException ex)
    {
      System.out.println("Exception :" ex);
    }
}
```

Unsubscribing a Durable Subscription for a Local Subscriber

```
public void unsubscribe(javax.jms.Topic topic,
                        java.lang.String subs_name)
                 throws JMSException
```

This method unsubscribes a durable subscription for a local subscriber. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to unsubscribe
`subs_name`	Name used to identify this subscription

See Also:

"Exclusive Access to Topics"

Example 6-70 Unsubscribing a Durable Subscription for a Local Subscriber

```
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

```
public void unsubscribe(javax.jms.Topic topic,
                        oracle.jms.AQjmsAgent remote_subscriber)
                 throws JMSException
```

This method unsubscribes a durable subscription for a remote subscriber. It has the following parameters:

Parameter	Description
`topic`	Non-temporary topic to unsubscribe
`remote_subscriber`	`AQjmsAgent` that refers to the remote subscriber. The address field of the `AQjmsAgent` cannot be null.

Note:

TEQ queues do not support remote subscribers.

See Also:

"Exclusive Access to Topics"

Example 6-71   Unsubscribing a Durable Subscription for a Remote Subscriber

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
AQjmsAgent                remoteAgent;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "OE", "Shipped_Orders_Topic");
remoteAgent = new AQjmsAgent("WS", "WS.Shipped_Orders_Topic", null);
/* unsubscribe the remote agent from shipped_orders */
((AQjmsSession)jms_sess).unsubscribe(shipped_orders, remoteAgent);
```

Creating a TopicReceiver for a Topic of Standard JMS Type Messages

```
public oracle.jms.AQjmsTopicReceiver createTopicReceiver(
              javax.jms.Topic topic,
              java.lang.String receiver_name,
              java.lang.String messageSelector)
       throws JMSException
```

This method creates a `TopicReceiver` for a topic of standard JMS type messages. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`receiver_name`	Name of message receiver

Parameter	Description
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.

Oracle Database Advanced Queuing allows messages to be sent to specified recipients. These receivers may or may not be subscribers of the topic. If the receiver is not a subscriber to the topic, then it receives only those messages that are explicitly addressed to it. This method must be used order to create a `TopicReceiver` object for consumers that are not durable subscribers.

See Also:

"MessageSelector"

Example 6-72 Creating a TopicReceiver for Standard JMS Type Messages

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = ull;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
TopicReceiver             receiver;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "WS", "Shipped_Orders_Topic");
receiver = ((AQjmsSession)jms_sess).createTopicReceiver(
          shipped_orders, "WesternRegion", null);
```

Creating a TopicReceiver for a Topic of Oracle Object Type Messages

```
public oracle.jms.AQjmsTopicReceiver createTopicReceiver(
              javax.jms.Topic topic,
              java.lang.String receiver_name,
              java.lang.String messageSelector,
              java.lang.Object payload_factory)
       throws JMSException
```

This method creates a `TopicReceiver` for a topic of Oracle object type messages with selector. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`receiver_name`	Name of message receiver
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.

Parameter	Description
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support object type messages.

Oracle Database Advanced Queuing allows messages to be sent to all subscribers of a topic or to specified recipients. These receivers may or may not be subscribers of the topic. If the receiver is not a subscriber to the topic, then it receives only those messages that are explicitly addressed to it. This method must be used order to create a `TopicReceiver` object for consumers that are not durable subscribers.

See Also:

"MessageSelector"

Example 6-73 Creating a TopicReceiver for Oracle Object Type Messages

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
TopicReceiver             receiver;
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
          "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
          "WS", "Shipped_Orders_Topic");
receiver = ((AQjmsSession)jms_sess).createTopicReceiver(
          shipped_orders, "WesternRegion", null);
```

Creating a TopicBrowser for Standard JMS Messages

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector)
                                      throws JMSException
```

This method creates a `TopicBrowser` for topics with `TextMessage`, `StreamMessage`, `ObjectMessage`, `BytesMessage`, or `MapMessage` message bodies. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`cons_name`	Name of the durable subscriber or consumer
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

See Also:

"MessageSelector"

Example 6-74 Creating a TopicBrowser Without a Selector

```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(topic, "SUBS1");
```

Example 6-75 Creating a TopicBrowser With a Specified Selector

```
/* Create a browser for topics with a specified selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
/* create a Browser to look at messages with correlationID = RUSH  */
browser = ((AQjmsSession) jms_session).createBrowser(
          topic, "SUBS1", "JMSCorrelationID = 'RUSH'");
```

Creating a TopicBrowser for Standard JMS Messages, Locking Messages

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector,
                                             boolean locked)
                                      throws JMSException
```

This method creates a `TopicBrowser` for topics with text, stream, objects, bytes or map messages, locking messages while browsing. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`cons_name`	Name of the durable subscriber or consumer
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`locked`	If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)

Example 6-76 Creating a TopicBrowser Without a Selector, Locking Messages While Browsing

```
/* Create a browser without a selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
browser = ((AQjmsSession) jms_session).createBrowser(
          topic, "SUBS1", true);
```

Example 6-77 Creating a TopicBrowser With a Specified Selector, Locking Messages

```
/* Create a browser for topics with a specified selector */
TopicSession    jms_session;
TopicBrowser    browser;
Topic           topic;
/* create a Browser to look at messages with correlationID = RUSH in 
lock mode */
browser = ((AQjmsSession) jms_session).createBrowser(
          topic, "SUBS1", "JMSCorrelationID = 'RUSH'", true);
```

Creating a TopicBrowser for Oracle Object Type Messages

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector,
                                             java.lang.Object payload_factory)
                                      throws JMSException
```

This method creates a `TopicBrowser` for topics of Oracle object type messages. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`cons_name`	Name of the durable subscriber or consumer
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support object type messages.

The `CustomDatumFactory` for a particular Java class that maps to the SQL object type payload can be obtained using the `getFactory` static method. Assume the topic `test_topic` has payload of type `SCOTT.EMPLOYEE` and the Java class that is generated by Jpublisher for this Oracle object type is called `Employee`. The Employee class implements the `CustomDatum` interface. The `CustomDatumFactory` for this class can be obtained by using the `Employee.getFactory()` method.

See Also:

"MessageSelector"

Example 6-78 Creating a TopicBrowser for AdtMessage Messages

```
/* Create a browser for a Topic with AdtMessage messages of type EMPLOYEE*/
TopicSession jms_session
TopicBrowser browser;
Topic        test_topic;
browser = ((AQjmsSession) jms_session).createBrowser(
          test_topic, "SUBS1", Employee.getFactory());
```

Creating a TopicBrowser for Oracle Object Type Messages, Locking Messages

```
public oracle.jms.TopicBrowser createBrowser(javax.jms.Topic topic,
                                             java.lang.String cons_name,
                                             java.lang.String messageSelector,
                                             java.lang.Object payload_factory,
                                             boolean locked)
                                      throws JMSException
```

This method creates a `TopicBrowser` for topics of Oracle object type messages, locking messages while browsing. It has the following parameters:

Parameter	Description
`topic`	Topic to access
`cons_name`	Name of the durable subscriber or consumer
`messageSelector`	Only messages with properties matching the `messageSelector` expression are delivered. A value of null or an empty string indicates that there is no `messageSelector` for the message consumer.
`payload_factory`	`CustomDatumFactory` or `ORADataFactory` for the Java class that maps to the Oracle ADT
`locked`	If set to true, then messages are locked as they are browsed (similar to a SELECT for UPDATE)

Note:

- `CustomDatum` support will be deprecated in a future release. Use `ORADataFactory` payload factories instead.
- TxEventQ queues do not support object type messages.

See Also:

"MessageSelector"

Example 6-79 Creating a TopicBrowser for AdtMessage Messages, Locking Messages

```
/* Create a browser for a Topic with AdtMessage messages of type EMPLOYEE* in
lock mode/
TopicSession jms_session
TopicBrowser browser;
Topic        test_topic;
browser = ((AQjmsSession) jms_session).createBrowser(
          test_topic, "SUBS1", Employee.getFactory(), true);
```

Browsing Messages Using a TopicBrowser

```
public void purgeSeen()
               throws JMSException
```

This method browses messages using a `TopicBrowser`. Use methods in `java.util.Enumeration` to go through the list of messages. Use the method `purgeSeen` in `TopicBrowser` to purge messages that have been seen during the current browse.

Example 6-80 Creating a TopicBrowser with a Specified Selector

```
/* 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()
```

Oracle Java Message Service Shared Interfaces

The following topics describe the Java Message Service (JMS) operational interface (shared interfaces) to Oracle Database Advanced Queuing (AQ).

- Oracle Database Advanced Queuing JMS Operational Interface: Shared Interfaces
- Specifying JMS Message Properties
- Setting Default TimeToLive for All Messages Sent by a MessageProducer
- Setting Default Priority for All Messages Sent by a MessageProducer
- Creating an AQjms Agent
- Receiving a Message Synchronously

- Specifying the Navigation Mode for Receiving Messages
- Receiving a Message Asynchronously
- Getting Message ID
- Getting JMS Message Properties
- Closing and Shutting Down
- Troubleshooting

Oracle Database Advanced Queuing JMS Operational Interface: Shared Interfaces

The following topics discuss Oracle Database Advanced Queuing shared interfaces for JMS operations.

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

Starting a JMS Connection

```
public void start()
          throws JMSException
```

`AQjmsConnection.start()` starts a JMS connection for receiving messages.

Getting a JMS Connection

```
public oracle.jms.AQjmsConnection getJmsConnection()
                                            throws JMSException
```

`AQjmsSession.getJmsConnection()` gets a JMS connection from a session.

Committing All Operations in a Session

```
public void commit()
           throws JMSException
```

`AQjmsSession.commit()` commits all JMS and SQL operations performed in a session.

Rolling Back All Operations in a Session

```
public void rollback()
              throws JMSException
```

`AQjmsSession.rollback()` terminates all JMS and SQL operations performed in a session.

Getting the JDBC Connection from a Session

```
public java.sql.Connection getDBConnection()
                                    throws JMSException
```

`AQjmsSession.getDBConnection()` gets the underlying JDBC connection from a JMS session. The JDBC connection can be used to perform SQL operations as part of the same transaction in which the JMS operations are accomplished.

Example 6-81   Getting Underlying JDBC Connection from JMS Session

```
java.sql.Connection db_conn;
QueueSession     jms_sess;
db_conn = ((AQjmsSession)jms_sess).getDBConnection();
```

Getting the OracleOCIConnectionPool from a JMS Connection

```
public oracle.jdbc.pool.OracleOCIConnectionPool getOCIConnectionPool()
```

`AQjmsConnection.getOCIConnectionPool()` gets the underlying `OracleOCIConnectionPool` from a JMS connection. The settings of the `OracleOCIConnectionPool` instance can be tuned by the user depending on the connection usage, for example, the number of sessions the user wants to create using the given connection. The user should not, however, close the `OracleOCIConnectionPool` instance being used by the JMS connection.

Example 6-82   Getting Underlying OracleOCIConnectionPool from JMS Connection

```
oracle.jdbc.pool.OracleOCIConnectionPool cpool;
QueueConnection jms_conn;
cpool = ((AQjmsConnection)jms_conn).getOCIConnectionPool();
```

Creating a BytesMessage

```
public javax.jms.BytesMessage createBytesMessage()
                                          throws JMSException
```

`AQjmsSession.createBytesMessage()` creates a bytes message. It can be used only if the queue table that contains the destination queue/topic was created with the `SYS.AQ$_JMS_BYTES_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Creating a MapMessage

```
public javax.jms.MapMessage createMapMessage()
                                      throws JMSException
```

`AQjmsSession.createMapMessage()` creates a map message. It can be used only if the queue table that contains the destination queue/topic was created with the `SYS.AQ$_JMS_MAP_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Creating a StreamMessage

```
public javax.jms.StreamMessage createStreamMessage()
                                            throws JMSException
```

`AQjmsSession.createStreamMessage()` creates a stream message. It can be used only if the queue table that contains the destination queue/topic was created with the `SYS.AQ$_JMS_STREAM_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Creating an ObjectMessage

```
public javax.jms.ObjectMessage createObjectMessage(java.io.Serializable object)
                                            throws JMSException
```

`AQjmsSession.createObjectMessage()` creates an object message. It can be used only if the queue table that contains the destination queue/topic was created with the `SYS.AQ$_JMS_OBJECT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Creating a TextMessage

```
public javax.jms.TextMessage createTextMessage()
                                        throws JMSException
```

`AQjmsSession.createTextMessage()` creates a text message. It can be used only if the queue table that contains the destination queue/topic was created with the `SYS.AQ$_JMS_TEXT_MESSAGE` or `AQ$_JMS_MESSAGE` payload types.

Creating a JMS Message

```
public javax.jms.Message createMessage()
                                throws JMSException
```

`AQjmsSession.createMessage()` creates a JMS message. You can use the `AQ$_JMS_MESSAGE` construct message to construct messages of different types. The message type must be one of the following:

- `DBMS_AQ.JMS_TEXT_MESSAGE`
- `DBMS_AQ.JMS_OBJECT_MESSAGE`
- `DBMS_AQ.JMS_MAP_MESSAGE`
- `DBMS_AQ.JMS_BYTES_MESSAGE`
- `DBMS_AQ.JMS_STREAM_MESSAGE`

You can also use this ADT to create a header-only JMS message.

Creating an AdtMessage

```
public oracle.jms.AdtMessage createAdtMessage()
                                       throws JMSException
```

`AQjmsSession.createAdtMessage()` creates an `AdtMessage`. It can be used only if the queue table that contains the queue/topic was created with an Oracle ADT payload type. An `AdtMessage` must be populated with an object that implements the `CustomDatum` interface. This object must be the Java mapping of the SQL ADT defined as the payload for the queue/topic. Java classes corresponding to SQL ADT types can be generated using the Jpublisher tool.

Setting a JMS Correlation Identifier

```
public void setJMSCorrelationID(java.lang.String correlationID)
                         throws JMSException
```

`AQjmsMessage.setJMSCorrelationID()` specifies the message correlation identifier.

Specifying JMS Message Properties

Property names starting with JMS are provider-specific. User-defined properties cannot start with JMS.

The following provider properties can be set by clients using text, stream, object, bytes or map messages:

- `JMSXAppID (string)`
- `JMSXGroupID (string)`
- `JMSXGroupSeq (int)`
- `JMS_OracleExcpQ (string)`

  This message property specifies the exception queue.
- `JMS_OracleDelay (int)`

  This message property specifies the message delay in seconds.

The following properties can be set on `AdtMessage`

- `JMS_OracleExcpQ (String)`

  This message property specifies the exception queue as "*schema*.queue_name"
- `JMS_OracleDelay (int)`

  This message property specifies the message delay in seconds.

This section contains these topics:

- Setting a Boolean Message Property
- Setting a String Message Property
- Setting an Integer Message Property
- Setting a Double Message Property
- Setting a Float Message Property
- Setting a Byte Message Property
- Setting a Long Message Property
- Setting a Short Message Property
- Getting an Object Message Property

Setting a Boolean Message Property

```
public void setBooleanProperty(java.lang.String name,
                               boolean value)
                        throws JMSException
```

`AQjmsMessage.setBooleanProperty()` specifies a message property as Boolean. It has the following parameters:

Parameter	Description
`name`	Name of the Boolean property
`value`	Boolean property value to set in the message

Setting a String Message Property

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value)
                       throws JMSException
```

`AQjmsMessage.setStringProperty()` specifies a message property as string. It has the following parameters:

Parameter	Description
`name`	Name of the string property
`value`	String property value to set in the message

Setting an Integer Message Property

```
public void setIntProperty(java.lang.String name,
                           int value)
                    throws JMSException
```

`AQjmsMessage.setIntProperty()` specifies a message property as integer. It has the following parameters:

Parameter	Description
`name`	Name of the integer property
`value`	Integer property value to set in the message

Setting a Double Message Property

```
public void setDoubleProperty(java.lang.String name,
                              double value)
                       throws JMSException
```

`AQjmsMessage.setDoubleProperty()` specifies a message property as double. It has the following parameters:

Parameter	Description
`name`	Name of the double property
`value`	Double property value to set in the message

Setting a Float Message Property

```
public void setFloatProperty(java.lang.String name,
                             float value)
                      throws JMSException
```

`AQjmsMessage.setFloatProperty()` specifies a message property as float. It has the following parameters:

Parameter	Description
`name`	Name of the float property
`value`	Float property value to set in the message

Setting a Byte Message Property

```
public void setByteProperty(java.lang.String name,
                            byte value)
                     throws JMSException
```

`AQjmsMessage.setByteProperty()` specifies a message property as byte. It has the following parameters:

Parameter	Description
`name`	Name of the byte property
`value`	Byte property value to set in the message

Setting a Long Message Property

```
public void setLongProperty(java.lang.String name,
                            long value)
                     throws JMSException
```

`AQjmsMessage.setLongProperty()` specifies a message property as long. It has the following parameters:

Parameter	Description
`name`	Name of the long property
`value`	Long property value to set in the message

Setting a Short Message Property

```
public void setShortProperty(java.lang.String name,
                             short value)
                      throws JMSException
```

`AQjmsMessage.setShortProperty()` specifies a message property as short. It has the following parameters:

Parameter	Description
`name`	Name of the short property
`value`	Short property value to set in the message

Setting an Object Message Property

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value)
                       throws JMSException
```

`AQjmsMessage.setObjectProperty()` specifies a message property as object. Only objectified primitive values are supported: Boolean, byte, short, integer, long, float, double and string. It has the following parameters:

Parameter	Description
`name`	Name of the Java object property
`value`	Java object property value to set in the message

Setting Default TimeToLive for All Messages Sent by a MessageProducer

```
public void setTimeToLive(long timeToLive)
                   throws JMSException
```

This method sets the default `TimeToLive` for all messages sent by a `MessageProducer`. It is calculated after message delay has taken effect. This method has the following parameter:

Parameter	Description
`timeToLive`	Message time to live in milliseconds (zero is unlimited)

Example 6-83 Setting Default TimeToLive for All Messages Sent by a MessageProducer

```
/* Set default timeToLive value to 100000 milliseconds for all messages sent by the
QueueSender*/
QueueSender sender;
sender.setTimeToLive(100000);
```

Setting Default Priority for All Messages Sent by a MessageProducer

```
public void setPriority(int priority)
                 throws JMSException
```

This method sets the default `Priority` for all messages sent by a `MessageProducer`. It has the following parameter:

Parameter	Description
`priority`	Message priority for this message producer. The default is 4.

Priority values can be any integer. A smaller number indicates higher priority. If a priority value is explicitly specified during a `send()` operation, then it overrides the default value set by this method.

Example 6-84 Setting Default Priority Value for All Messages Sent by QueueSender

```
/* Set default priority value to 2 for all messages sent by the QueueSender*/
QueueSender sender;
sender.setPriority(2);
```

Example 6-85 Setting Default Priority Value for All Messages Sent by TopicPublisher

```
/* Set default priority value to 2 for all messages sent by the TopicPublisher*/
TopicPublisher publisher;
publisher.setPriority(1);
```

Creating an AQjms Agent

```
public void createAQAgent(java.lang.String agent_name,
                          boolean enable_http,
                   throws JMSException
```

This method creates an `AQjmsAgent`. It has the following parameters:

Parameter	Description
`agent_name`	Name of the AQ agent
`enable_http`	If set to true, then this agent is allowed to access AQ through HTTP

Receiving a Message Synchronously

You can receive a message synchronously by specifying Timeout or without waiting. You can also receive a message using a transformation:

- Using a Message Consumer by Specifying Timeout
- Using a Message Consumer Without Waiting
- Receiving Messages from a Destination Using a Transformation

Using a Message Consumer by Specifying Timeout

```
public javax.jms.Message receive(long timeout)
                          throws JMSException
```

This method receives a message using a message consumer by specifying timeout.

Parameter	Description
`timeout`	Timeout value in milliseconds

Example 6-86 Using a Message Consumer by Specifying Timeout

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              t_sess    = null;
TopicSession              jms_sess;
Topic                     shipped_orders;
int                       myport = 5521;
```

```
/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(
   "MYHOSTNAME", "MYSID", myport, "oci8");
t_conn = tc_fact.createTopicConnection("jmstopic", "jmstopic");
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic(
   "WS", "Shipped_Orders_Topic");

/* create a subscriber, specifying the correct CustomDatumFactory  and
selector */
subscriber1 = jms_sess.createDurableSubscriber(
   shipped_orders, 'WesternShipping',
   " priority > 1 and tab.user_data.region like 'WESTERN %'",
   false, AQjmsAgent.getFactory());
/* receive, blocking for 30 seconds if there were no messages */
Message = subscriber.receive(30000);
```

Example 6-87   JMS: Blocking Until a Message Arrives

```
public BolOrder get_new_order1(QueueSession jms_session)
 {
   Queue              queue;
   QueueReceiver      qrec;
   ObjectMessage      obj_message;
   BolCustomer        customer;
   BolOrder           new_order = null;
   String             state;

   try
   {
   /* get a handle to the new_orders queue */
    queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
    qrec = jms_session.createReceiver(queue);

    /* wait for a message to show up in the queue */
    obj_message = (ObjectMessage)qrec.receive();
    new_order = (BolOrder)obj_message.getObject();
    customer = new_order.getCustomer();
    state    = customer.getState();
    System.out.println("Order:  for customer " + customer.getName());
   }
  catch (JMSException ex)
   {
     System.out.println("Exception: " + ex);
   }
   return new_order;
 }
```

Using a Message Consumer Without Waiting

```
public javax.jms.Message receiveNoWait()
                                throws JMSException
```

This method receives a message using a message consumer without waiting.

Example 6-88   JMS: Nonblocking Messages

```
public BolOrder poll_new_order3(QueueSession jms_session)
 {
   Queue              queue;
   QueueReceiver      qrec;
```

```
  ObjectMessage    obj_message;
  BolCustomer      customer;
  BolOrder         new_order = null;
  String           state;

  try
  {
   /* get a handle to the new_orders queue */
   queue = ((AQjmsSession) jms_session).getQueue("OE", "OE_neworders_que");
   qrec = jms_session.createReceiver(queue);

   /* check for a message to show in the queue */
   obj_message = (ObjectMessage)qrec.receiveNoWait();
   new_order = (BolOrder)obj_message.getObject();
   customer = new_order.getCustomer();
   state    = customer.getState();

   System.out.println("Order:  for customer " + customer.getName());
  }
  catch (JMSException ex)
  {
    System.out.println("Exception: " + ex);
  }
  return new_order;
}
```

Receiving Messages from a Destination Using a Transformation

A transformation can be applied when receiving a message from a queue or topic. The transformation is applied to the message before returning it to JMS application.

The transformation can be specified using the `setTransformation()` interface of the `AQjmsQueueReceiver`, `AQjmsTopicSubscriber` or `AQjmsTopicReceiver`.

Example 6-89 JMS: Receiving Messages from a Destination Using a Transformation

Assume that the Western Shipping application retrieves messages from the OE_bookedorders_topic. It specifies the transformation `OE2WS` to retrieve the message as the Oracle object type `WS_order`. Assume that the WSOrder Java class has been generated by Jpublisher to map to the Oracle object `WS.WS_order`:

```
public AQjmsAdtMessage retrieve_bookedorders(TopicSession jms_session)
  AQjmsTopicReceiver  receiver;
  Topic               topic;
  Message             msg = null;

  try
  {
    /* get a handle to the OE_bookedorders_topic */
    topic = ((AQjmsSession)jms_session).getTopic("OE", "OE_bookedorders_topic");

    /* Create a receiver for WShip */
    receiver = ((AQjmsSession)jms_session).createTopicReceiver(
       topic, "WShip, null, WSOrder.getFactory());

    /* set the transformation in the publisher */
    receiver.setTransformation("OE2WS");
    msg = receiver.receive(10);
  }
  catch (JMSException ex)
  {
     System.out.println("Exception :", ex);
  }
       return (AQjmsAdtMessage)msg;
}
```

Specifying the Navigation Mode for Receiving Messages

```
public void setNavigationMode(int mode)
                       throws JMSException
```

This method specifies the navigation mode for receiving messages. It has the following parameter:

Parameter	Description
`mode`	New value of the navigation mode

Example 6-90 Specifying Navigation Mode for Receiving Messages

```
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 nomessage */
Message = subscriber.receive();
```

Receiving a Message Asynchronously

You can receive a message asynchronously two ways:

- Specifying a Message Listener at the Message Consumer
- Specifying a Message Listener at the Session

Specifying a Message Listener at the Message Consumer

```
public void setMessageListener(javax.jms.MessageListener myListener)
                        throws JMSException
```

This method specifies a message listener at the message consumer. It has the following parameter:

Parameter	Description
`myListener`	Sets the consumer message listener

Example 6-91 Specifying Message Listener at Message Consumer

```
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 nomessage */
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

```
public void setMessageListener(javax.jms.MessageListener listener)
                        throws JMSException
```

This method specifies a message listener at the session.

Parameter	Description
`listener`	Message listener to associate with this session

Getting Message ID

This section contains these topics:

- Getting the Correlation Identifier
- Getting the Message Identifier

Getting the Correlation Identifier

```
public java.lang.String getJMSCorrelationID()
          throws JMSException
```

`AQjmsMessage.getJMSCorrelationID()` gets the correlation identifier of a message.

Getting the Message Identifier

```
public byte[] getJMSCorrelationIDAsBytes()
          throws JMSException
```

`AQjmsMessage.getJMSMessageID()` gets the message identifier of a message as bytes or a string.

Getting JMS Message Properties

This section contains these topics:

- Getting a Boolean Message Property
- Getting a String Message Property
- Getting an Integer Message Property
- Getting a Double Message Property
- Getting a Float Message Property
- Getting a Byte Message Property
- Getting a Long Message Property
- Getting a Short Message Property
- Getting an Object Message Property

Getting a Boolean Message Property

```
public boolean getBooleanProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getBooleanProperty()` gets a message property as Boolean. It has the following parameter:

Parameter	Description
`name`	Name of the Boolean property

Getting a String Message Property

```
public java.lang.String getStringProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getStringProperty()` gets a message property as string. It has the following parameter:

Parameter	Description
`name`	Name of the string property

Getting an Integer Message Property

```
public int getIntProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getIntProperty()` gets a message property as integer. It has the following parameter:

Parameter	Description
`name`	Name of the integer property

Getting a Double Message Property

```
public double getDoubleProperty(java.lang.String name)
                         throws JMSException
```

`AQjmsMessage.getDoubleProperty()` gets a message property as double. It has the following parameter:

Parameter	Description
`name`	Name of the double property

Getting a Float Message Property

```
public float getFloatProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getFloatProperty()` gets a message property as float. It has the following parameter:

Parameter	Description
`name`	Name of the float property

Getting a Byte Message Property

```
public byte getByteProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getByteProperty()` gets a message property as byte. It has the following parameter:

Parameter	Description
`name`	Name of the byte property

Getting a Long Message Property

```
public long getLongProperty(java.lang.String name)
          throws JMSException
```

`AQjmsMessage.getLongProperty()` gets a message property as long. It has the following parameter:

Parameter	Description
`name`	Name of the long property

Getting a Short Message Property

```
public short getShortProperty(java.lang.String name)
                       throws JMSException
```

AQjmsMessage.getShortProperty() gets a message property as short. It has the following parameter:

Parameter	Description
`name`	Name of the short property

Getting an Object Message Property

```
public java.lang.Object getObjectProperty(java.lang.String name)
                                   throws JMSException
```

`AQjmsMessage.getObjectProperty()` gets a message property as object. It has the following parameter:

Parameter	Description
`name`	Name of the object property

Example 6-92 Getting Message Property as an Object

```
TextMessage message;
message.getObjectProperty("empid", new Integer(1000);
```

Closing and Shutting Down

This section contains these topics:

- Closing a MessageProducer
- Closing a Message Consumer
- Stopping a JMS Connection
- Closing a JMS Session
- Closing a JMS Connection

Closing a MessageProducer

```
public void close()
           throws JMSException
```

`AQjmsProducer.close()` closes a `MessageProducer`.

Closing a Message Consumer

```
public void close()
           throws JMSException
```

`AQjmsConsumer.close()` closes a message consumer.

Stopping a JMS Connection

```
public void stop()
          throws JMSException
```

`AQjmsConnection.stop()` stops a JMS connection.

Closing a JMS Session

```
public void close()
           throws JMSException
```

`AQjmsSession.close()` closes a JMS session.

Closing a JMS Connection

```
public void close()
          throws JMSException
```

`AQjmsConnection.close()` closes a JMS connection and releases all resources allocated on behalf of the connection. Because the JMS provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Troubleshooting

This section contains these topics:

- Getting a JMS Error Code
- Getting a JMS Error Number
- Getting an Exception Linked to a JMS Exception
- Printing the Stack Trace for a JMS Exception
- Setting an Exception Listener
- Getting an Exception Listener

Getting a JMS Error Code

```
public java.lang.String getErrorCode()
```

`AQjmsException.getErrorCode()` gets the error code for a JMS exception.

Getting a JMS Error Number

```
public int getErrorNumber()
```

`AQjmsException.getErrorNumber()` gets the error number for a JMS exception.

> **Note:**
>
> This method will be deprecated in a future release. Use `getErrorCode()` instead.

Getting an Exception Linked to a JMS Exception

```
public java.lang.String getLinkString()
```

`AQjmsException.getLinkString()` gets the exception linked to a JMS exception. In general, this contains the SQL exception raised by the database.

Printing the Stack Trace for a JMS Exception

```
public void printStackTrace(java.io.PrintStream s)
```

`AQjmsException.printStackTrace()` prints the stack trace for a JMS exception.

Setting an Exception Listener

```
public void setExceptionListener(javax.jms.ExceptionListener listener)
                          throws JMSException
```

`AQjmsConnection.setExceptionListener()` specifies an exception listener for a connection. It has the following parameter:

Parameter	Description
`listener`	Exception listener

If an exception listener has been registered, then it is informed of any serious problem detected for a connection. This is accomplished by calling the listener `onException()` method, passing it a JMS exception describing the problem. This allows a JMS client to be notified of a problem asynchronously. Some connections only consume messages, so they have no other way to learn the connection has failed.

Example 6-93 Specifying Exception Listener for Connection

```
//register an exception listener
Connection jms_connection;
jms_connection.setExceptionListener(
    new ExceptionListener() {
        public void onException (JMSException jmsException) {
            System.out.println("JMS-EXCEPTION: " + jmsException.toString());
        }
    };
  );
```

Getting an Exception Listener

```
public javax.jms.ExceptionListener getExceptionListener()
                                                  throws JMSException
```

`AQjmsConnection.getExceptionListener()` gets the exception listener for the connection.

Example 6-94 demonstrates how to use `ExceptionListener` with `MessageListener`. Ensure that the following conditions are met:

- The user `jmsuser` with password `jmsuser` is created in the database, with appropriate privileges.
- The queue `demoQueue` is created and started.

This example demonstrates how to make the MessageListener asynchronously receive the messages, where the exception listener recreates the JMS objects in case there is a connection restart.

Example 6-94 Using ExceptionListener with MessageListener

```
import java.util.Enumeration;
import java.util.Properties;

import javax.jms.Connection;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.QueueBrowser;
import javax.jms.Session;
import javax.jms.TextMessage;

import oracle.jms.AQjmsConnectionFactory;
import oracle.jms.AQjmsFactory;
import oracle.jms.AQjmsSession;
```

```
public class JMSDemo {

  static String queueName = "demoQueue";

  static String queueOwner = "jmsuser";

  static String queueOwnerPassword = "jmsuser";

  static Connection connection = null;

  static int numberOfMessages = 25000;

  static int messageCount = 0;

  static String jdbcURL = "";

  public static void main(String args[]) {
    try {
      jdbcURL = System.getProperty("JDBC_URL");

      if (jdbcURL == null)
        System.out
            .println("The system property JDBC_URL has not been set, " +
                    "usage:java -DJDBC_URL=xxx filename ");
      else {
        JMSDemo demo = new JMSDemo();
        demo.performJmsOperations();
      }
    } catch (Exception exception) {
      System.out.println("Exception : " + exception);
      exception.printStackTrace();
    } finally {
      try {
        if (connection != null)
          connection.close();
      } catch (Exception exc) {
        exc.printStackTrace();
      }
    }
    System.out.println("\nEnd of Demo aqjmsdemo11.");
  }

  public void performJmsOperations() {
    try {
      connection = getConnection(jdbcURL);
      Session session = connection.createSession(false,
          Session.AUTO_ACKNOWLEDGE);
      Queue queue = session.createQueue(queueName);

      // remove the messages from the Queue
      drainQueue(queueName, queueOwner, jdbcURL, true);

      // set the exception listener on the Connection
      connection.setExceptionListener(new DemoExceptionListener());

      MessageProducer producer = session.createProducer(queue);
      TextMessage textMessage = null;

      System.out.println("Sending " + numberOfMessages + " messages to queue "
          + queueName);
      for (int i = 0; i < numberOfMessages; i++) {
        textMessage = session.createTextMessage();
        textMessage.setText("Sample message text");
        producer.send(textMessage);
      }

      MessageConsumer consumer = session.createConsumer(queue);
      System.out.println("Setting the message listener ...");
      consumer.setMessageListener(new DemoMessageListener());
      connection.start();

      // Introduce a long wait to allow the listener to receive all the messages
      while (messageCount < numberOfMessages) {
        try {
          Thread.sleep(5000);
        } catch (InterruptedException interruptedException) {
        }
      }
    } catch (JMSException jmsException) {
      jmsException.printStackTrace();
    }
  }

  // Sample message listener
  static class DemoMessageListener implements javax.jms.MessageListener {

    public void onMessage(Message message) {
      try {
        System.out.println("Message listener received message with JMSMessageID "
                + message.getJMSMessageID());
        messageCount++;
      } catch (JMSException jmsException) {
        System.out.println("JMSException " + jmsException.getMessage());
      }
    }
  }

  // sample exception listener
  static class DemoExceptionListener implements javax.jms.ExceptionListener {

    public void onException(JMSException jmsException) {
      try {
        // As a first step close the connection
        if (connection != null)
          connection.close();
      } catch (JMSException exception) {}

      try {
        System.out.println("Re-create the necessary JMS objects ...");
        connection = getConnection(jdbcURL);
        connection.start();
        Session session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue(queueName);
        MessageConsumer consumer = session.createConsumer(queue);
        consumer.setMessageListener(new DemoMessageListener());
      } catch (JMSException newJmsException) {
        newJmsException.printStackTrace();
      }
    }
  }

  // Utility method to get a connection
  static Connection getConnection(String jdbcUrl) throws JMSException {
    Properties prop = new Properties();
    prop.put("user", queueOwner);
    prop.put("password", queueOwnerPassword);

    AQjmsConnectionFactory fact = (AQjmsConnectionFactory) AQjmsFactory
        .getConnectionFactory(jdbcUrl, prop);
    Connection conn = fact.createConnection();
    return conn;
  }

  // Utility method to remove the messages from the queue
  static void drainQueue(String queueName, String queueOwner, String jdbcUrl,
      boolean debugInfo) {
    Connection connection = null;
    Session session = null;
    long timeout = 10000;
    int count = 0;
    Message message = null;
    try {
      connection = getConnection(jdbcUrl);
      connection.start();
      session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      Queue queue = ((AQjmsSession) session).getQueue(queueOwner, queueName);

      MessageConsumer messageConsumer = session.createConsumer(queue);
      QueueBrowser browser = session.createBrowser(queue);
      Enumeration enumeration = browser.getEnumeration();

      if (enumeration.hasMoreElements()) {
        while ((message = messageConsumer.receive(timeout)) != null) {
          if (debugInfo) {
            count++;
          }
        }
      }
      messageConsumer.close();
      if (debugInfo) {
        System.out.println("Removed " + count + " messages from the queue : "
            + queueName);
      }
    } catch (JMSException jmsException) {
      jmsException.printStackTrace();
    } finally {
      try {
        if (session != null)
          session.close();

        if (connection != null)
          connection.close();
      } catch (Exception exception) {

      }
    }
  }

}
```

Example 6-95 Getting the Exception Listener for the Connection

```
//Get the exception listener
Connection jms_connection;
ExceptionListener el = jms_connection.getExceptionListener();
```

Oracle Java Message Service Types Examples

The following examples illustrate how to use Oracle JMS Types to dequeue and enqueue Oracle Database Advanced Queuing (AQ) messages.

- How to Setup the Oracle Database Advanced Queuing JMS Type Examples
- JMS BytesMessage Examples
- JMS StreamMessage Examples
- JMS MapMessage Examples
- More Oracle Database Advanced Queuing JMS Examples

How to Set Up the Oracle Database Advanced Queuing JMS Type Examples

To run Example 6-98 through Example 6-103 follow these steps:

1. Copy and save Example 6-96 as `setup.sql`.
2. Run `setup.sql` as follows:

   ```
   sqlplus /NOLOG @setup.sql
   ```

3. Log in to SQL*Plus as `jmsuser/jmsuser`.
4. Run the corresponding pair of SQL scripts for each type of message.

   For JMS `BytesMessage`, for example, run Example 6-98 and Example 6-99.
5. Ensure that your database parameter `java_pool-size` is large enough. For example, you can use `java_pool_size`=20M.

Example 6-96 Setting Up Environment for Running JMS Types Examples

```
connect sys;
enter password: password

Rem
Rem Create the JMS user: jmsuser
Rem

DROP USER jmsuser CASCADE;
CREATE USER jmsuser IDENTIFIED BY jmsuser;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;


set echo offset verify offconnect sysDROP USER jmsuser CASCADE;ACCEPT password CHAR
PROMPT 'Enter the password for JMSUSER: ' HIDECREATE USER jmsuser IDENTIFIED BY
&password;GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;GRANT EXECUTE ON
DBMS_AQADM TO jmsuser;GRANT EXECUTE ON DBMS_AQ TO jmsuser;GRANT EXECUTE ON DBMS_LOB TO
jmsuser;GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;connect jmsuser/&password
```

```
Rem
Rem Creating five AQ queue tables and five queues for five payloads:
Rem SYS.AQ$_JMS_TEXT_MESSAGE
Rem SYS.AQ$_JMS_BYTES_MESSAGE
Rem SYS.AQ$_JMS_STREAM_MESSAG
Rem SYS.AQ$_JMS_MAP_MESSAGE
Rem SYS.AQ$_JMS_MESSAGE
Rem

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_text',
    Queue_payload_type => 'SYS.AQ$_JMS_TEXT_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_bytes',
    Queue_payload_type => 'SYS.AQ$_JMS_BYTES_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_stream',
    Queue_payload_type => 'SYS.AQ$_JMS_STREAM_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_map',
    Queue_payload_type => 'SYS.AQ$_JMS_MAP_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (Queue_table => 'jmsuser.jms_qtt_general',
    Queue_payload_type => 'SYS.AQ$_JMS_MESSAGE', compatible => '8.1.0');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_text_que',
    Queue_table => 'jmsuser.jms_qtt_text');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_bytes_que',
    Queue_table => 'jmsuser.jms_qtt_bytes');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_stream_que',
    Queue_table => 'jmsuser.jms_qtt_stream');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_map_que',
    Queue_table => 'jmsuser.jms_qtt_map');
EXECUTE DBMS_AQADM.CREATE_QUEUE (Queue_name => 'jmsuser.jms_general_que',
    Queue_table => 'jmsuser.jms_qtt_general');

Rem
Rem Starting the queues and enable both enqueue and dequeue
Rem
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_text_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_bytes_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_stream_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_map_que');
EXECUTE DBMS_AQADM.START_QUEUE (Queue_name => 'jmsuser.jms_general_que');

Rem The supporting utility used in the example to help display results in SQLPLUS
enviroment

Rem
Rem Display a RAW data in SQLPLUS
Rem
create or replace procedure display_raw(rdata raw)
IS
    pos                 pls_integer;
    length              pls_integer;
BEGIN
    pos := 1;
    length := UTL_RAW.LENGTH(rdata);

    WHILE pos <= length LOOP
      IF pos+20 > length+1 THEN
        dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, length-pos+1));
      ELSE
        dbms_output.put_line(UTL_RAW.SUBSTR(rdata, pos, 20));
      END IF;
      pos := pos+20;
    END LOOP;
END display_raw;
/

show errors;

Rem
Rem Display a BLOB data in SQLPLUS
Rem
create or replace procedure display_blob(bdata blob)
IS
    pos                 pls_integer;
    length              pls_integer;
BEGIN
    length :=  dbms_lob.getlength(bdata);
    pos := 1;
    WHILE pos <= length LOOP
      display_raw(DBMS_LOB.SUBSTR(bdata, 2000, pos));
      pos := pos+2000;
    END LOOP;
END display_blob;
/

show errors;

Rem
Rem Display a VARCHAR data in SQLPLUS
Rem
create or replace procedure display_varchar(vdata varchar)
IS
    pos                 pls_integer;
    text_len            pls_integer;
BEGIN
    text_len :=  length(vdata);
    pos := 1;

    WHILE pos <= text_len LOOP
      IF pos+20 > text_len+1 THEN
        dbms_output.put_line(SUBSTR(vdata, pos, text_len-pos+1));
      ELSE
        dbms_output.put_line(SUBSTR(vdata, pos, 20));
      END IF;
      pos := pos+20;
    END LOOP;

END display_varchar;
/

show errors;

Rem
Rem Display a CLOB data in SQLPLUS
Rem
create or replace procedure display_clob(cdata clob)
IS
    pos                 pls_integer;
    length              pls_integer;
BEGIN
    length :=  dbms_lob.getlength(cdata);
    pos := 1;
    WHILE pos <= length LOOP
      display_varchar(DBMS_LOB.SUBSTR(cdata, 2000, pos));
      pos := pos+2000;
    END LOOP;
END display_clob;
/

show errors;

Rem
Rem Display a SYS.AQ$_JMS_EXCEPTION data in SQLPLUS
Rem
Rem When application receives an ORA-24197 error, It means the JAVA stored
Rem procedures has thrown some exceptions that could not be catergorized. The
Rem user can use GET_EXCEPTION procedure of SYS.AQ$_JMS_BYTES_MESSAGE,
Rem SYS.AQ$_JMS_STREAM_MESSAG or SYS.AQ$_JMS_MAP_MESSAGE
Rem to retrieve a SYS.AQ$_JMS_EXCEPTION object which contains more detailed
Rem information on this JAVA exception including the exception name, JAVA error
Rem message and stack trace.
Rem
Rem This utility function is to help display the SYS.AQ$_JMS_EXCEPTION object in
Rem SQLPLUS
Rem
create or replace procedure display_exp(exp SYS.AQ$_JMS_EXCEPTION)
IS
    pos1                pls_integer;
    pos2                pls_integer;
    text_data           varchar(2000);
BEGIN
    dbms_output.put_line('exception:'||exp.exp_name);
    dbms_output.put_line('err_msg:'||exp.err_msg);
    dbms_output.put_line('stack:'||length(exp.stack));
    pos1 := 1;
    LOOP
      pos2 := INSTR(exp.stack, chr(10), pos1);
      IF pos2 = 0 THEN
        pos2 := length(exp.stack)+1;
      END IF;

      dbms_output.put_line(SUBSTR(exp.stack, pos1, pos2-pos1));

      IF pos2 > length(exp.stack) THEN
        EXIT;
      END IF;

      pos1 := pos2+1;
    END LOOP;

END display_exp;
/

show errors;

EXIT;
```

Example 6-97   Setting Up the Examples

Example 6-96 performs the necessary setup for the JMS types examples. Copy and save it as `setup.sql`.

JMS BytesMessage Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS `BytesMessage`.

Example 6-98 shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS `BytesMessage` represented as `sys.aq$_jms_bytes_message` type in the database. This message later can be dequeued by a JAVA Oracle Java Message Service (Oracle JMS) client.

Example 6-99 illustrates how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS `BytesMessage` represented as `sys.aq$_jms_bytes_message` type in the database. This message might be enqueued by an Oracle JMS client.

Example 6-98   Populating and Enqueuing a BytesMessage

```
set echo offset verify offconnect sysDROP USER jmsuser CASCADE;ACCEPT password CHAR
PROMPT 'Enter the password for JMSUSER: ' HIDECREATE USER jmsuser IDENTIFIED BY
&password;GRANT DBA, AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE to jmsuser;GRANT EXECUTE ON
DBMS_AQADM TO jmsuser;GRANT EXECUTE ON DBMS_AQ TO jmsuser;GRANT EXECUTE ON DBMS_LOB TO
jmsuser;GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;connect jmsuser/&password

SET ECHO ON
set serveroutput on

DECLARE

    id                 pls_integer;
    agent              sys.aq$_agent := sys.aq$_agent(' ', null, 0);
    message            sys.aq$_jms_bytes_message;
    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN

    -- Consturct a empty BytesMessage object
    message := sys.aq$_jms_bytes_message.construct;

    -- Shows how to set the JMS header
    message.set_replyto(agent);
    message.set_type('tkaqpet1');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    -- Shows how to set JMS user properties
    message.set_string_property('color', 'RED');
    message.set_int_property('year', 1999);
    message.set_float_property('price', 16999.99);
    message.set_long_property('mileage', 300000);
    message.set_boolean_property('import', True);
    message.set_byte_property('password', -127);

    -- Shows how to populate the message payload of aq$_jms_bytes_message
    -- Passing -1 reserve a new slot within the message store of
sys.aq$_jms_bytes_message.
    -- The maximum number of sys.aq$_jms_bytes_message type of messges to be operated at
    -- the same time within a session is 20. Calling clean_body function with parameter
-1
    -- might result a ORA-24199 error if the messages currently operated is already 20.
    -- The user is responsible to call clean or clean_all function to clean up message
store.
    id := message.clear_body(-1);

    -- Write data into the BytesMessage paylaod. These functions are analogy of JMS JAVA
api's.
    -- See the document for detail.

    -- Write a byte to the BytesMessage payload
    message.write_byte(id, 10);

    -- Write a RAW data as byte array to the BytesMessage payload
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

    -- Write a portion of the RAW data as byte array to BytesMessage payload
    -- Note the offset follows JAVA convention, starting from 0
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')), 0, 16);

    -- Write a char to the BytesMessage payload
    message.write_char(id, 'A');

    -- Write a double to the BytesMessage payload
    message.write_double(id, 9999.99);

    -- Write a float to the BytesMessage payload
    message.write_float(id, 99.99);

    -- Write a int to the BytesMessage payload
    message.write_int(id, 12345);

    -- Write a long to the BytesMessage payload
    message.write_long(id, 1234567);

    -- Write a short to the BytesMessage payload
    message.write_short(id, 123);

    -- Write a String to the BytesMessage payload,
    -- the String is encoded in UTF8 in the message payload
    message.write_utf(id, 'Hello World!');

    -- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
    -- Without doing this, the PL/SQL message is still empty.
    message.flush(id);

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod population on this message anymore
    sys.aq$_jms_bytes_message.clean_all();
    --message.clean(id);

    -- Enqueue this message into AQ queue using DBMS_AQ package
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_bytes_que',
                    enqueue_options => enqueue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);
    EXCEPTION
    WHEN java_exp THEN
      dbms_output.put_line('exception information:');
      display_exp(sys.aq$_jms_stream_message.get_exception());

END;
/

commit;
```

Example 6-99 Dequeuing and Retrieving JMS BytesMessage Data

```
set echo off
set verify off

DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
set echo on
set serveroutput on size 20000

DECLARE

    id                 pls_integer;
    blob_data          blob;
    clob_data          clob;
    blob_len           pls_integer;
    message            sys.aq$_jms_bytes_message;
    agent              sys.aq$_agent;
    dequeue_options    dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);
    gdata              sys.aq$_jms_value;

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
    DBMS_OUTPUT.ENABLE (20000);

    -- Dequeue this message from AQ queue using DBMS_AQ package
    dbms_aq.dequeue(queue_name => 'jmsuser.jms_bytes_que',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

    -- Retrieve the header
    agent := message.get_replyto;

    dbms_output.put_line('Type: ' || message.get_type ||
                         ' UserId: ' || message.get_userid ||
                         ' AppId: ' || message.get_appid ||
                         ' GroupId: ' || message.get_groupid ||
                         ' GroupSeq: ' || message.get_groupseq);
    -- Retrieve the user properties
    dbms_output.put_line('price: ' || message.get_float_property('price'));
    dbms_output.put_line('color: ' || message.get_string_property('color'));
    IF message.get_boolean_property('import') = TRUE THEN
       dbms_output.put_line('import: Yes' );
    ELSIF message.get_boolean_property('import') = FALSE THEN
       dbms_output.put_line('import: No' );
    END IF;
    dbms_output.put_line('year: ' || message.get_int_property('year'));
    dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));
    dbms_output.put_line('password: ' || message.get_byte_property('password'));

-- Shows how to retrieve the message payload of aq$_jms_bytes_message

-- Prepare call, send the content in the PL/SQL aq$_jms_bytes_message object to
   -- Java stored procedure(Jserv) in the form of a byte array.
   -- Passing -1 reserves a new slot in msg store of sys.aq$_jms_bytes_message.
   -- Max number of sys.aq$_jms_bytes_message type of messges to be operated at
   -- the same time in a session is 20. Call clean_body fn. with parameter -1
   -- might result in ORA-24199 error if messages operated on are already 20.
   -- You must call clean or clean_all function to clean up message store.
    id := message.prepare(-1);

-- Read data from BytesMessage paylaod. These fns. are analogy of JMS Java
-- API's. See the JMS Types chapter for detail.
    dbms_output.put_line('Payload:');

    -- read a byte from the BytesMessage payload
    dbms_output.put_line('read_byte:' || message.read_byte(id));

    -- read a byte array into a blob object from the BytesMessage payload
    dbms_output.put_line('read_bytes:');
    blob_len := message.read_bytes(id, blob_data, 272);
    display_blob(blob_data);

    -- read a char from the BytesMessage payload
    dbms_output.put_line('read_char:'|| message.read_char(id));

    -- read a double from the BytesMessage payload
    dbms_output.put_line('read_double:'|| message.read_double(id));

    -- read a float from the BytesMessage payload
    dbms_output.put_line('read_float:'|| message.read_float(id));

    -- read a int from the BytesMessage payload
    dbms_output.put_line('read_int:'|| message.read_int(id));

    -- read a long from the BytesMessage payload
    dbms_output.put_line('read_long:'|| message.read_long(id));

    -- read a short from the BytesMessage payload
    dbms_output.put_line('read_short:'|| message.read_short(id));

    -- read a String from the BytesMessage payload.
    -- the String is in UTF8 encoding in the message payload
    dbms_output.put_line('read_utf:');
    message.read_utf(id, clob_data);
    display_clob(clob_data);

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod retrieving on this message anymore
    message.clean(id);
    -- sys.aq$_jms_bytes_message.clean_all();

    EXCEPTION
    WHEN java_exp THEN
      dbms_output.put_line('exception information:');
      display_exp(sys.aq$_jms_bytes_message.get_exception());

END;
/

commit;
```

JMS StreamMessage Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS `StreamMessage`.

Example 6-100 shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS `StreamMessage` represented as `sys.aq$_jms_stream_message` type in the database. This message later can be dequeued by an Oracle JMS client.

Example 6-101 shows how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS `StreamMessage` represented as `sys.aq$_jms_stream_message` type in the database. This message might be enqueued by an Oracle JMS client.

Example 6-100 Populating and Enqueuing a JMS StreamMessage

```
set echo off
set verify off


DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
SET ECHO ON
set serveroutput on

DECLARE

    id                 pls_integer;
    agent              sys.aq$_agent := sys.aq$_agent(' ', null, 0);
    message            sys.aq$_jms_stream_message;
    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN

    -- Consturct a empty StreamMessage object
    message := sys.aq$_jms_stream_message.construct;
    -- Shows how to set the JMS header
    message.set_replyto(agent);
    message.set_type('tkaqpet1');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    -- Shows how to set JMS user properties
    message.set_string_property('color', 'RED');
    message.set_int_property('year', 1999);
    message.set_float_property('price', 16999.99);
    message.set_long_property('mileage', 300000);
    message.set_boolean_property('import', True);
    message.set_byte_property('password', -127);

    -- Shows how to populate the message payload of aq$_jms_stream_message

    -- Passing -1 reserve a new slot within the message store of 
sys.aq$_jms_stream_message.
    -- The maximum number of sys.aq$_jms_stream_message type of messges to be operated at
    -- the same time within a session is 20. Calling clean_body function with parameter 
-1 
    -- might result a ORA-24199 error if the messages currently operated is already 20.
    -- The user is responsible to call clean or clean_all function to clean up message 
store.
    id := message.clear_body(-1);

    -- Write data into the message paylaod. These functions are analogy of JMS JAVA 
api's.
    -- See the document for detail.

    -- Write a byte to the StreamMessage payload
    message.write_byte(id, 10);

    -- Write a RAW data as byte array to the StreamMessage payload
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

    -- Write a portion of the RAW data as byte array to the StreamMessage payload
    -- Note the offset follows JAVA convention, starting from 0
    message.write_bytes(id, UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')), 0, 16);

    -- Write a char to the StreamMessage payload
    message.write_char(id, 'A');

    -- Write a double to the StreamMessage payload
    message.write_double(id, 9999.99);

    -- Write a float to the StreamMessage payload
    message.write_float(id, 99.99);

    -- Write a int to the StreamMessage payload
    message.write_int(id, 12345);

    -- Write a long to the StreamMessage payload
    message.write_long(id, 1234567);

    -- Write a short to the StreamMessage payload
    message.write_short(id, 123);

    -- Write a String to the StreamMessage payload
    message.write_string(id, 'Hello World!');

    -- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
    -- Without doing this, the PL/SQL message is still empty.
    message.flush(id);

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod population on this message anymore
    sys.aq$_jms_stream_message.clean_all();
    --message.clean(id);

    -- Enqueue this message into AQ queue using DBMS_AQ package
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_stream_que',
                    enqueue_options => enqueue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);


    EXCEPTION
    WHEN java_exp THEN
      dbms_output.put_line('exception information:');
      display_exp(sys.aq$_jms_stream_message.get_exception());

END;
/

commit;
```

Example 6-101   Dequeuing and Retrieving Data From a JMS StreamMessage

```
set echo off
set verify off


DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password
set echo on
set serveroutput on

DECLARE

    id                 pls_integer;
    blob_data          blob;
    clob_data          clob;
    message            sys.aq$_jms_stream_message;
    agent              sys.aq$_agent;
    dequeue_options    dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);
    gdata              sys.aq$_jms_value;

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
```

```
BEGIN
    DBMS_OUTPUT.ENABLE (20000);

    -- Dequeue this message from AQ queue using DBMS_AQ package
    dbms_aq.dequeue(queue_name => 'jmsuser.jms_stream_que',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

    -- Retrieve the header
    agent := message.get_replyto;

    dbms_output.put_line('Type: ' || message.get_type ||
                     ' UserId: ' || message.get_userid ||
                     ' AppId: ' || message.get_appid ||
                     ' GroupId: ' || message.get_groupid ||
                     ' GroupSeq: ' || message.get_groupseq);

    -- Retrieve the user properties
    dbms_output.put_line('price: ' || message.get_float_property('price'));
    dbms_output.put_line('color: ' || message.get_string_property('color'));
    IF message.get_boolean_property('import') = TRUE THEN
       dbms_output.put_line('import: Yes' );
    ELSIF message.get_boolean_property('import') = FALSE THEN
       dbms_output.put_line('import: No' );
    END IF;
    dbms_output.put_line('year: ' || message.get_int_property('year'));
    dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));
    dbms_output.put_line('password: ' || message.get_byte_property('password'));

    -- Shows how to retrieve the message payload of aq$_jms_stream_message

    -- The prepare call send the content in the PL/SQL aq$_jms_stream_message object to
    -- JAVA stored procedure(Jserv) in the form of byte array.
    -- Passing -1 reserve a new slot within the message store of
sys.aq$_jms_stream_message.
    -- The maximum number of sys.aq$_jms_stream_message type of messges to be operated at
    -- the same time within a session is 20. Calling clean_body function with parameter
-1
    -- might result a ORA-24199 error if the messages currently operated is already 20.
    -- The user is responsible to call clean or clean_all function to clean up message
store.
    id := message.prepare(-1);


    -- Assume the users know the types of data in the StreamMessage payload.
    -- The user can use the specific read function corresponding with the data type.
    -- These functions are analogy of JMS JAVA api's. See the document for detail.
    dbms_output.put_line('Retrieve payload by Type:');

    -- Read a byte from the StreamMessage payload
    dbms_output.put_line('read_byte:' || message.read_byte(id));

    -- Read a byte array into a blob object from the StreamMessage payload
    dbms_output.put_line('read_bytes:');
    message.read_bytes(id, blob_data);
    display_blob(blob_data);

    -- Read another byte array into a blob object from the StreamMessage payload
    dbms_output.put_line('read_bytes:');
    message.read_bytes(id, blob_data);
    display_blob(blob_data);

    -- Read a char from the StreamMessage payload
    dbms_output.put_line('read_char:'|| message.read_char(id));

    -- Read a double from the StreamMessage payload
    dbms_output.put_line('read_double:'|| message.read_double(id));

    -- Read a float from the StreamMessage payload
    dbms_output.put_line('read_float:'|| message.read_float(id));

    -- Read a int from the StreamMessage payload
    dbms_output.put_line('read_int:'|| message.read_int(id));

    -- Read a long from the StreamMessage payload
    dbms_output.put_line('read_long:'|| message.read_long(id));

    -- Read a short from the StreamMessage payload
    dbms_output.put_line('read_short:'|| message.read_short(id));

    -- Read a String into a clob data from the StreamMessage payload
    dbms_output.put_line('read_string:');
    message.read_string(id, clob_data);
    display_clob(clob_data);


    -- Assume the users do not know the types of data in the StreamMessage payload.
    -- The user can use read_object method to read the data into a sys.aq$_jms_value
object
    -- These functions are analogy of JMS JAVA api's. See the document for detail.

    -- Reset the stream pointer to the begining of the message so that we can read
throught
    -- the message payload again.
    message.reset(id);

    LOOP
      message.read_object(id, gdata);
      IF gdata IS NULL THEN
        EXIT;
      END IF;

      CASE gdata.type
        WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTE      THEN
                 dbms_output.put_line('read_object/byte:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_SHORT     THEN
                 dbms_output.put_line('read_object/short:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_INTEGER   THEN
                 dbms_output.put_line('read_object/int:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_LONG      THEN
                 dbms_output.put_line('read_object/long:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_FLOAT     THEN
                 dbms_output.put_line('read_object/float:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_DOUBLE    THEN
                 dbms_output.put_line('read_object/double:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_BOOLEAN   THEN
                 dbms_output.put_line('read_object/boolean:' || gdata.num_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_CHARACTER THEN
                 dbms_output.put_line('read_object/char:' || gdata.char_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_STRING    THEN
                 dbms_output.put_line('read_object/string:');
                 display_clob(gdata.text_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTES     THEN
                 dbms_output.put_line('read_object/bytes:');
                 display_blob(gdata.bytes_val);
        ELSE dbms_output.put_line('No such data type');
      END CASE;

    END LOOP;

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod retrieving on this message anymore
    message.clean(id);
    -- sys.aq$_jms_stream_message.clean_all();

    EXCEPTION
    WHEN java_exp THEN
      dbms_output.put_line('exception information:');
      display_exp(sys.aq$_jms_stream_message.get_exception());

END;
/

commit;
```

JMS MapMessage Examples

This section includes examples that illustrate enqueuing and dequeuing of a JMS `MapMessage`.

Example 6-102 shows how to use JMS type member functions with `DBMS_AQ` functions to populate and enqueue a JMS `MapMessage` represented as `sys.aq$_jms_map_message` type in the database. This message later can be dequeued by an Oracle JMS client.

Example 6-103 illustrates how to use JMS type member functions with `DBMS_AQ` functions to dequeue and retrieve data from a JMS `MapMessage` represented as `sys.aq$_jms_map_message` type in the database. This message can be enqueued by an Oracle JMS client.

Example 6-102 Populating and Enqueuing a JMS MapMessage

```
set echo off
set verify off


DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password

SET ECHO ON
set serveroutput on

DECLARE

    id                  pls_integer;
    agent               sys.aq$_agent := sys.aq$_agent(' ', null, 0);
    message             sys.aq$_jms_map_message;
    enqueue_options     dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN

    -- Consturct a empty map message object
    message := sys.aq$_jms_map_message.construct;

    -- Shows how to set the JMS header
    message.set_replyto(agent);
    message.set_type('tkaqpet1');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    -- Shows how to set JMS user properties
    message.set_string_property('color', 'RED');
    message.set_int_property('year', 1999);
    message.set_float_property('price', 16999.99);
    message.set_long_property('mileage', 300000);
    message.set_boolean_property('import', True);
    message.set_byte_property('password', -127);

    -- Shows how to populate the message payload of aq$_jms_map_message

    -- Passing -1 reserve a new slot within the message store of sys.aq$_jms_map_message.
    -- The maximum number of sys.aq$_jms_map_message type of messges to be operated at
    -- the same time within a session is 20. Calling clean_body function with parameter
-1
    -- might result a ORA-24199 error if the messages currently operated is already 20.
    -- The user is responsible to call clean or clean_all function to clean up message
store.
    id := message.clear_body(-1);

    -- Write data into the message paylaod. These functions are analogy of JMS JAVA
api's.
    -- See the document for detail.

    -- Set a byte entry in map message payload
    message.set_byte(id, 'BYTE', 10);

    -- Set a byte array entry using RAW data in map message payload
    message.set_bytes(id, 'BYTES', UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')));

    -- Set a byte array entry using only a portion of the RAW data in map message payload
    -- Note the offset follows JAVA convention, starting from 0
    message.set_bytes(id, 'BYTES_PART', UTL_RAW.XRANGE(HEXTORAW('00'), HEXTORAW('FF')),
0, 16);

    -- Set a char entry in map message payload
    message.set_char(id, 'CHAR', 'A');

    -- Set a double entry in map message payload
    message.set_double(id, 'DOUBLE', 9999.99);

    -- Set a float entry in map message payload
    message.set_float(id, 'FLOAT', 99.99);

    -- Set a int entry in map message payload
    message.set_int(id, 'INT', 12345);

    -- Set a long entry in map message payload
    message.set_long(id, 'LONG', 1234567);

    -- Set a short entry in map message payload
    message.set_short(id, 'SHORT', 123);

    -- Set a String entry in map message payload
    message.set_string(id, 'STRING', 'Hello World!');

    -- Flush the data from JAVA stored procedure (JServ) to PL/SQL side
    -- Without doing this, the PL/SQL message is still empty.
    message.flush(id);

    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod population on this message anymore
    sys.aq$_jms_map_message.clean_all();
    --message.clean(id);

    -- Enqueue this message into AQ queue using DBMS_AQ package
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_map_que',
                    enqueue_options => enqueue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

END;
/

commit;
```

Example 6-103 Dequeuing and Retrieving Data From a JMS MapMessage

```
set echo off
set verify off


DROP USER jmsuser CASCADE;

ACCEPT password CHAR PROMPT 'Enter the password for JMSUSER: ' HIDE

CREATE USER jmsuser IDENTIFIED BY &password;
GRANT EXECUTE ON DBMS_AQADM TO jmsuser;
GRANT EXECUTE ON DBMS_AQ TO jmsuser;
GRANT EXECUTE ON DBMS_LOB TO jmsuser;
GRANT EXECUTE ON DBMS_JMS_PLSQL TO jmsuser;
connect jmsuser/&password

set echo on
set serveroutput on

DECLARE

    id                 pls_integer;
    blob_data          blob;
    clob_data          clob;
    message            sys.aq$_jms_map_message;
    agent              sys.aq$_agent;
    dequeue_options    dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid              raw(16);
    name_arr           sys.aq$_jms_namearray;
    gdata              sys.aq$_jms_value;

    java_exp           exception;
    pragma EXCEPTION_INIT(java_exp, -24197);
BEGIN
    DBMS_OUTPUT.ENABLE (20000);

    -- Dequeue this message from AQ queue using DBMS_AQ package
    dbms_aq.dequeue(queue_name => 'jmsuser.jms_map_que',
                    dequeue_options => dequeue_options,
                    message_properties => message_properties,
                    payload => message,
                    msgid => msgid);

    -- Retrieve the header
    agent := message.get_replyto;

    dbms_output.put_line('Type: ' || message.get_type ||
                         ' UserId: ' || message.get_userid ||
                         ' AppId: ' || message.get_appid ||
                         ' GroupId: ' || message.get_groupid ||
                         ' GroupSeq: ' || message.get_groupseq);

    -- Retrieve the user properties
    dbms_output.put_line('price: ' || message.get_float_property('price'));
    dbms_output.put_line('color: ' || message.get_string_property('color'));
    IF message.get_boolean_property('import') = TRUE THEN
       dbms_output.put_line('import: Yes' );
    ELSIF message.get_boolean_property('import') = FALSE THEN
       dbms_output.put_line('import: No' );
    END IF;
    dbms_output.put_line('year: ' || message.get_int_property('year'));
    dbms_output.put_line('mileage: ' || message.get_long_property('mileage'));
    dbms_output.put_line('password: ' || message.get_byte_property('password'));


    -- Shows how to retrieve the message payload of aq$_jms_map_message

    -- 'Prepare' sends the content in the PL/SQL aq$_jms_map_message object to
    -- Java stored procedure(Jserv) in the form of byte array.
    -- Passing -1 reserve a new slot within the message store of
    -- sys.aq$_jms_map_message. The maximum number of sys.aq$_jms_map_message
    -- type of messges to be operated at the same time within a session is 20.
    -- Calling clean_body function with parameter -1
    -- might result a ORA-24199 error if the messages currently operated is
    -- already 20. The user is responsible to call clean or clean_all function
    -- to clean up message store.
    id := message.prepare(-1);

    -- Assume the users know the names and types in the map message payload.
    -- The user can use names to get the corresponsing values.
    -- These functions are analogous to JMS Java API's. See JMS Types chapter
    -- for detail.
    dbms_output.put_line('Retrieve payload by Name:');

    -- Get a byte entry from the map message payload
    dbms_output.put_line('get_byte:' || message.get_byte(id, 'BYTE'));

    -- Get a byte array entry from the map message payload
    dbms_output.put_line('get_bytes:');
    message.get_bytes(id, 'BYTES', blob_data);
```

```
display_blob(blob_data);

-- Get another byte array entry from the map message payload
dbms_output.put_line('get_bytes:');
message.get_bytes(id, 'BYTES_PART', blob_data);
display_blob(blob_data);

-- Get a char entry from the map message payload
dbms_output.put_line('get_char:'|| message.get_char(id, 'CHAR'));

-- get a double entry from the map message payload
dbms_output.put_line('get_double:'|| message.get_double(id, 'DOUBLE'));

-- Get a float entry from the map message payload
dbms_output.put_line('get_float:'|| message.get_float(id, 'FLOAT'));

-- Get a int entry from the map message payload
dbms_output.put_line('get_int:'|| message.get_int(id, 'INT'));

-- Get a long entry from the map message payload
dbms_output.put_line('get_long:'|| message.get_long(id, 'LONG'));

-- Get a short entry from the map message payload
dbms_output.put_line('get_short:'|| message.get_short(id, 'SHORT'));

-- Get a String entry from the map message payload
dbms_output.put_line('get_string:');
message.get_string(id, 'STRING', clob_data);
display_clob(clob_data);

-- Assume users do not know names and types in map message payload.
-- User can first retrieve the name array containing all names in the
-- payload and iterate through the name list and get the corresponding
-- value. These functions are analogous to JMS Java API's.
-- See JMS Type chapter for detail.
dbms_output.put_line('Retrieve payload by iteration:');

-- Get the name array from the map message payload
name_arr := message.get_names(id);

-- Iterate through the name array to retrieve the value for each of the name.
FOR i IN name_arr.FIRST..name_arr.LAST LOOP

-- Test if a name exist in the map message payload
-- (It is not necessary in this case, just a demostration on how to use it)
  IF message.item_exists(id, name_arr(i)) THEN
    dbms_output.put_line('item exists:'||name_arr(i));

-- Because we do not know the type of entry, we must use sys.aq$_jms_value
-- type object for the data returned
   message.get_object(id, name_arr(i), gdata);
   IF gdata IS NOT NULL THEN
    CASE gdata.type
    WHEN   sys.dbms_jms_plsql.DATA_TYPE_BYTE
      THEN dbms_output.put_line('get_object/byte:' || gdata.num_val);
    WHEN   sys.dbms_jms_plsql.DATA_TYPE_SHORT
      THEN dbms_output.put_line('get_object/short:' || gdata.num_val);
    WHEN   sys.dbms_jms_plsql.DATA_TYPE_INTEGER
      THEN dbms_output.put_line('get_object/int:' || gdata.num_val);
    WHEN   sys.dbms_jms_plsql.DATA_TYPE_LONG
      THEN dbms_output.put_line('get_object/long:' || gdata.num_val);
    WHEN   sys.dbms_jms_plsql.DATA_TYPE_FLOAT
```

```
          THEN dbms_output.put_line('get_object/float:' || gdata.num_val);
        WHEN   sys.dbms_jms_plsql.DATA_TYPE_DOUBLE
          THEN dbms_output.put_line('get_object/double:' || gdata.num_val);
        WHEN   sys.dbms_jms_plsql.DATA_TYPE_BOOLEAN
          THEN dbms_output.put_line('get_object/boolean:' || gdata.num_val);
        WHEN   sys.dbms_jms_plsql.DATA_TYPE_CHARACTER
          THEN dbms_output.put_line('get_object/char:' || gdata.char_val);
        WHEN   sys.dbms_jms_plsql.DATA_TYPE_STRING
          THEN dbms_output.put_line('get_object/string:');
               display_clob(gdata.text_val);
        WHEN sys.dbms_jms_plsql.DATA_TYPE_BYTES
        THEN
             dbms_output.put_line('get_object/bytes:');
             display_blob(gdata.bytes_val);
          ELSE dbms_output.put_line('No such data type');
          END CASE;
        END IF;
      ELSE
        dbms_output.put_line('item not exists:'||name_arr(i));
      END IF;

    END LOOP;


    -- Use either clean_all or clean to clean up the message store when the user
    -- do not plan to do paylaod population on this message anymore
    message.clean(id);
    -- sys.aq$_jms_map_message.clean_all();

    EXCEPTION
    WHEN java_exp THEN
      dbms_output.put_line('exception information:');
      display_exp(sys.aq$_jms_stream_message.get_exception());

END;
/

commit;
```

More Oracle Database Advanced Queuing JMS Examples

The sample program in Example 6-104 enqueues a large `TextMessage` (along with JMS user properties) in an Oracle Database Advanced Queuing queue created through the Oracle JMS administrative interfaces to hold JMS `TEXT` messages. Both the `TextMessage` and `BytesMessage` enqueued in this example can be dequeued using Oracle JMS clients.

The sample program in Example 6-105 enqueues a large `BytesMessage`.

Example 6-104   Enqueuing a Large TextMessage

```
DECLARE

    text        varchar2(32767);
    agent       sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
    message     sys.aq$_jms_text_message;

    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid               raw(16);

BEGIN
```

```
    message := sys.aq$_jms_text_message.construct;

    message.set_replyto(agent);
    message.set_type('tkaqpet2');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq');
    message.set_groupid('st');
    message.set_groupseq(1);

    message.set_boolean_property('import', True);
    message.set_string_property('color', 'RED');
    message.set_short_property('year', 1999);
    message.set_long_property('mileage', 300000);
    message.set_double_property('price', 16999.99);
    message.set_byte_property('password', 127);

    FOR i IN 1..500 LOOP
        text := CONCAT (text, '1234567890');
    END LOOP;

    message.set_text(text);

    dbms_aq.enqueue(queue_name => 'jmsuser.jms_text_t1',
                       enqueue_options => enqueue_options,
                       message_properties => message_properties,
                       payload => message,
                       msgid => msgid);

END;
```

Example 6-105 Enqueuing a Large BytesMessage

```
DECLARE

    text        VARCHAR2(32767);
    bytes       RAW(32767);
    agent       sys.aq$_agent   := sys.aq$_agent(' ', null, 0);
    message     sys.aq$_jms_bytes_message;
    body        BLOB;
    position    INT;

    enqueue_options    dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);

BEGIN

    message := sys.aq$_jms_bytes_message.construct;

    message.set_replyto(agent);
    message.set_type('tkaqper4');
    message.set_userid('jmsuser');
    message.set_appid('plsql_enq_raw');
    message.set_groupid('st');
    message.set_groupseq(1);

    message.set_boolean_property('import', True);
    message.set_string_property('color', 'RED');
    message.set_short_property('year', 1999);
    message.set_long_property('mileage', 300000);
    message.set_double_property('price', 16999.99);
```

```
--  prepare a huge payload into a blob

    FOR i IN 1..1000 LOOP
        text := CONCAT (text, '0123456789ABCDEF');
    END LOOP;

    bytes := HEXTORAW(text);

    dbms_lob.createtemporary(lob_loc => body, cache => TRUE);
    dbms_lob.open (body, DBMS_LOB.LOB_READWRITE);
    position := 1 ;
    FOR i IN 1..10 LOOP
        dbms_lob.write ( lob_loc => body,
                amount => FLOOR((LENGTH(bytes)+1)/2),
                offset => position,
                buffer => bytes);
        position := position + FLOOR((LENGTH(bytes)+1)/2) ;
    END LOOP;

-- end of the preparation

    message.set_bytes(body);
    dbms_aq.enqueue(queue_name => 'jmsuser.jms_bytes_t1',
                       enqueue_options => enqueue_options,
                       message_properties => message_properties,
                       payload => message,
                       msgid => msgid);

    dbms_lob.freetemporary(lob_loc => body);
END;
```

7

Oracle Database Advanced Queuing Operations Using PL/SQL

These topics describes the Oracle Database Advanced Queuing (AQ) PL/SQL operational interface.

See Also:

- Oracle Database Advanced Queuing: Programmatic Interfaces for a list of available functions in each programmatic interface
- "DBMS_AQ" in *Oracle Database PL/SQL Packages and Types Reference* for more information on the PL/SQL interface
- *Oracle Database Advanced Queuing Java API Reference* for more information on the Java interface
- "More OCI Relational Functions" in *Oracle Call Interface Programmer's Guide*
- "OCI Programming Advanced Topics" in *Oracle Call Interface Programmer's Guide* for more information on the Oracle Call Interface (OCI)

Using Secure Queues

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter.

See "MESSAGE_PROPERTIES_T Type" in *Oracle Database PL/SQL Packages and Types Reference* for more information about `sender_id`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Database Advanced Queuing agent using `DBMS_AQADM.CREATE_AQ_AGENT`.

- You must map `sender_id` to a database user with enqueue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this.

See Also:

– "Creating an Oracle Database Advanced Queuing Agent"
– "Enabling Database Access"

Enqueuing Messages

This procedure adds a message to the specified queue.

```
DBMS_AQ.ENQUEUE(
   queue_name          IN      VARCHAR2,
   enqueue_options     IN      enqueue_options_t,
   message_properties  IN      message_properties_t,
   payload             IN      "type_name",
   msgid               OUT     RAW);
```

It is not possible to update the message payload after a message has been enqueued. If you want to change the message payload, then you must dequeue the message and enqueue a new message.

To store a payload of type `RAW`, Oracle Database Advanced Queuing creates a queue table with `LOB` column as the payload repository. The maximum size of the payload is determined by which programmatic interface you use to access Oracle Database Advanced Queuing. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G.

If a message is enqueued to a multiconsumer queue with no recipient and the queue has no subscribers (or rule-based subscribers that match this message), then Oracle error ORA 24033 is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

If several messages are enqueued in the same second, then they all have the same `enq_time`. In this case the order in which messages are dequeued depends on `step_no`, a variable that is monotonically increasing for each message that has the same `enq_time`. There is no situation when both `enq_time` and `step_no` are the same for two messages enqueued in the same session.

Enqueue Options

The `enqueue_options` parameter specifies the options available for the enqueue operation. It has the following attributes:

- `visibility`

  The `visibility` attribute specifies the transactional behavior of the enqueue request. `ON_COMMIT` (the default) makes the enqueue is part of the current transaction. `IMMEDIATE` makes the enqueue operation an autonomous transaction which commits at the end of the operation.

  Do not use the `IMMEDIATE` option when you want to use LOB locators. LOB locators are valid only for the duration of the transaction. Your locator will not be valid, because the `immediate` option automatically commits the transaction.

You must set the `visibility` attribute to `IMMEDIATE` to use buffered messaging.

- `relative_msgid`

  The `relative_msgid` attribute specifies the message identifier of the message referenced in the sequence deviation operation. This parameter is ignored unless `sequence_deviation` is specified with the `BEFORE` attribute.

- `sequence_deviation`

  The `sequence_deviation` attribute specifies when the message should be dequeued, relative to other messages already in the queue. `BEFORE` puts the message ahead of the message specified by `relative_msgid`. `TOP` puts the message ahead of any other messages.

  Specifying `sequence_deviation` for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

  > **Note:**
  >
  > The `sequence_deviation` attribute has no effect in releases prior to Oracle Database Advanced Queuing 10*g* Release 1 (10.1) if `message_grouping` is set to `TRANSACTIONAL`.
  >
  > The sequence deviation feature is deprecated in Oracle Database Advanced Queuing 10*g* Release 2 (10.2).

- `transformation`

  The `transformation` attribute specifies a transformation that will be applied before enqueuing the message. The return type of the transformation function must match the type of the queue.

- `delivery_mode`

  If the `delivery_mode` attribute is the default `PERSISTENT`, then the message is enqueued as a persistent message. If it is set to `BUFFERED`, then the message is enqueued as an buffered message. Null values are not allowed.

Message Properties

The `message_properties` parameter contains the information that Oracle Database Advanced Queuing uses to manage individual messages. It has the following attributes:

- `priority`

  The `priority` attribute specifies the priority of the message. It can be any number, including negative numbers. A smaller number indicates higher priority.

- `delay`

  The `delay` attribute specifies the number of seconds during which a message is in the `WAITING` state. After this number of seconds, the message is in the `READY` state and available for dequeuing. If you specify `NO_DELAY`, then the message is available for immediate dequeuing. Dequeuing by `msgid` overrides the `delay` specification.

> **Note:**
>
> Delay is not supported with buffered messaging.

- `expiration`

  The `expiration` attribute specifies the number of seconds during which the message is available for dequeuing, starting from when the message reaches the `READY` state. If the message is not dequeued before it expires, then it is moved to the exception queue in the `EXPIRED` state. If you specify `NEVER`, then the message does not expire.

  > **Note:**
  >
  > Message delay and expiration are enforced by the queue monitor (`QMN`) background processes. You must start the QMN processes for the database if you intend to use the delay and expiration features of Oracle Database Advanced Queuing.

- `correlation`

  The `correlation` attribute is an identifier supplied by the producer of the message at enqueue time.

- `attempts`

  The `attemps` attribute specifies the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.

- `recipient_list`

  The `recipient_list` parameter is valid only for queues that allow multiple consumers. The default recipients are the queue subscribers.

- `exception_queue`

  The `exception_queue` attribute specifies the name of the queue into which the message is moved if it cannot be processed successfully. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log.

- `delivery_mode`

  Any value for `delivery_mode` specified in message properties at enqueue time is ignored. The value specified in enqueue options is used to set the delivery mode of the message. If the delivery mode in enqueue options is left unspecified, then it defaults to persistent.

- `enqueue_time`

  The `enqueue_time` attribute specifies the time the message was enqueued. This value is always in Universal Coordinated Time (UTC), and is determined by the system and cannot be set by the user at enqueue time.

> **Note:**
>
> Because information about seasonal changes in the system clock (switching between standard time and daylight-saving time, for example) is stored with each queue table, seasonal changes are automatically reflected in `enqueue_time`. If the system clock is changed for some other reason, then you must restart the database for Oracle Database Advanced Queuing to pick up the changed time.

- `state`

  The `state` attribute specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time.
- `sender_id`

  The `sender_id` attribute is an identifier of type `aq$_agent` specified at enqueue time by the message producer.
- `original_msgid`

  The original_msgid attribute is used by Oracle Database AQ for propagating messages.
- `transaction_group`

  The `transaction_group` attribute specifies the transaction group for the message. This attribute is set only by `DBMS_AQ.DEQUEUE_ARRAY`. This attribute cannot be used to set the transaction group of a message through `DBMS_AQ.ENQUEUE` or `DBMS_AQ.ENQUEUE_ARRAY`.
- `user_property`

  The `user_property` attribute is optional. It is used to store additional information about the payload.

The examples in the following topics use the same users, message types, queue tables, and queues as do the examples in Oracle Database Advanced Queuing Administrative Interface. If you have not already created these structures in your test environment, then you must run the following examples:

- Example 12-1
- Example 12-2
- Example 12-3
- Example 12-5
- Example 12-7
- Example 12-8
- Example 12-23
- Example 12-25
- Example 12-26
- Example 12-27
- Example 12-28
- Example 12-36

For Example 12-1, you must connect as a user with administrative privileges. For the other examples in the preceding list, you can connect as user `test_adm`. After you have created the queues, you must start them as shown in "Starting a Queue". Except as noted otherwise, you can connect as ordinary queue user 'test' to run all examples.

Enqueuing a LOB Type Message

Example 7-3 creates procedure blobenqueue() using the test.lob_type message payload object type created in Example 12-1. On enqueue, the LOB attribute is set to EMPTY_BLOB. After the enqueue completes, but before the transaction is committed, the LOB attribute is selected from the user_data column of the test.lob_qtab queue table. The LOB data is written to the queue using the LOB interfaces (which are available through both OCI and PL/SQL). The actual enqueue operation is shown in

On dequeue, the message payload will contain the LOB locator. You can use this LOB locator after the dequeue, but before the transaction is committed, to read the LOB data. This is shown in Example 7-14.

Enqueuing Multiple Messages to a Single-Consumer Queue

Example 7-5 enqueues six messages to test.obj_queue. These messages are dequeued in Example 7-17.

Enqueuing Multiple Messages to a Multiconsumer Queue

Example 7-6 requires that you connect as user 'test_adm' to add subscribers RED and GREEN to queue test.multiconsumer_queue. The subscribers are required for Example 7-7.

Example 7-7 enqueues multiple messages from sender 001. MESSAGE 1 is intended for all queue subscribers. MESSAGE 2 is intended for RED and BLUE. These messages are dequeued in Example 7-17.

Enqueuing Grouped Messages

Example 7-8 enqueues three groups of messages, with three messages in each group. These messages are dequeued in Example 7-16.

Enqueuing a Message with Delay and Expiration

In Example 7-9, an application wants a message to be dequeued no earlier than a week from now, but no later than three weeks from now. Because expiration is calculated from the earliest dequeue time, this requires setting the expiration time for two weeks.

Example 7-1 Enqueuing a Message, Specifying Queue Name and Payload

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   message := test.message_typ(001, 'TEST MESSAGE', 'First message to obj_queue');
   DBMS_AQ.ENQUEUE(
      queue_name              => 'test.obj_queue',
      enqueue_options         => enqueue_options,
      message_properties      => message_properties,
      payload                 => message,
      msgid                   => message_handle);
   COMMIT;
END;
/
```

Example 7-2   Enqueuing a Message, Specifying Priority

```
DECLARE
   enqueue_options       DBMS_AQ.enqueue_options_t;
   message_properties    DBMS_AQ.message_properties_t;
   message_handle        RAW(16);
   message               test.order_typ;
BEGIN
   message := test.order_typ(002, 'PRIORITY MESSAGE', 'priority 30');
   message_properties.priority := 30;
   DBMS_AQ.ENQUEUE(
      queue_name              => 'test.priority_queue',
      enqueue_options         => enqueue_options,
      message_properties      => message_properties,
      payload                 => message,
      msgid                   => message_handle);
   COMMIT;
END;
/
```

Example 7-3   Creating an Enqueue Procedure for LOB Type Messages

```
CREATE OR REPLACE PROCEDURE blobenqueue(msgno IN NUMBER) AS
   enq_userdata          test.lob_typ;
   enq_msgid             RAW(16);
   enqueue_options       DBMS_AQ.enqueue_options_t;
   message_properties    DBMS_AQ.message_properties_t;
   lob_loc               BLOB;
   buffer                RAW(4096);
BEGIN
   buffer       := HEXTORAW(RPAD('FF', 4096, 'FF'));
   enq_userdata := test.lob_typ(msgno, 'Large Lob data', EMPTY_BLOB(), msgno);
   DBMS_AQ.ENQUEUE(
      queue_name          => 'test.lob_queue',
      enqueue_options     => enqueue_options,
      message_properties  => message_properties,
      payload             => enq_userdata,
      msgid               => enq_msgid);
   SELECT t.user_data.data INTO lob_loc
      FROM lob_qtab t
      WHERE t.msgid = enq_msgid;
   DBMS_LOB.WRITE(lob_loc, 2000, 1, buffer );
   COMMIT;
END;
/
```

Example 7-4   Enqueuing a LOB Type Message

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

Example 7-5   Enqueuing Multiple Messages

```
SET SERVEROUTPUT ON
DECLARE
   enqueue_options      DBMS_AQ.enqueue_options_t;
   message_properties   DBMS_AQ.message_properties_t;
   message_handle       RAW(16);
   message              test.message_typ;
BEGIN
   message := test.message_typ(001, 'ORANGE', 'ORANGE enqueued first.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   message := test.message_typ(001, 'ORANGE', 'ORANGE also enqueued second.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   message := test.message_typ(001, 'YELLOW', 'YELLOW enqueued third.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   message := test.message_typ(001, 'VIOLET', 'VIOLET enqueued fourth.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   message := test.message_typ(001, 'PURPLE', 'PURPLE enqueued fifth.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   message := test.message_typ(001, 'PINK', 'PINK enqueued sixth.');
   DBMS_AQ.ENQUEUE(
         queue_name           => 'test.obj_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
   COMMIT;
END;
/
```

Example 7-6 Adding Subscribers RED and GREEN

```
DECLARE
   subscriber          sys.aq$_agent;
BEGIN
   subscriber     :=  sys.aq$_agent('RED', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name  =>  'test.multiconsumer_queue',
      subscriber  =>  subscriber);

   subscriber     :=  sys.aq$_agent('GREEN', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name  =>  'test.multiconsumer_queue',
      subscriber  =>  subscriber);
```

```
END;
/
```

Example 7-7 Enqueuing Multiple Messages to a Multiconsumer Queue

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   recipients          DBMS_AQ.aq$_recipient_list_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   message := test.message_typ(001, 'MESSAGE 1','For queue subscribers');
   DBMS_AQ.ENQUEUE(
      queue_name          => 'test.multiconsumer_queue',
      enqueue_options     => enqueue_options,
      message_properties  => message_properties,
      payload             => message,
      msgid               => message_handle);

   message := test.message_typ(001, 'MESSAGE 2', 'For two recipients');
   recipients(1) := sys.aq$_agent('RED', NULL, NULL);
   recipients(2) := sys.aq$_agent('BLUE', NULL, NULL);
   message_properties.recipient_list := recipients;
   DBMS_AQ.ENQUEUE(
      queue_name          => 'test.multiconsumer_queue',
      enqueue_options     => enqueue_options,
      message_properties  => message_properties,
      payload             => message,
      msgid               => message_handle);
   COMMIT;
END;
/
```

Example 7-8 Enqueuing Grouped Messages

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
  FOR groupno in 1..3 LOOP
    FOR msgno in 1..3 LOOP
      message := test.message_typ(
               001,
               'GROUP ' || groupno,
               'Message ' || msgno || ' in group ' || groupno);
      DBMS_AQ.ENQUEUE(
         queue_name             => 'test.group_queue',
         enqueue_options        => enqueue_options,
         message_properties     => message_properties,
         payload                => message,
         msgid                  => message_handle);
    END LOOP;
    COMMIT;
  END LOOP;
END;
/
```

Example 7-9   Enqueuing a Message, Specifying Delay and Expiration

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   message := test.message_typ(001, 'DELAYED', 'Message is delayed one week.');
   message_properties.delay      := 7*24*60*60;
   message_properties.expiration := 2*7*24*60*60;
   DBMS_AQ.ENQUEUE(
      queue_name           => 'test.obj_queue',
      enqueue_options      => enqueue_options,
      message_properties   => message_properties,
      payload              => message,
      msgid                => message_handle);
   COMMIT;
END;
/
```

Example 7-10   Enqueuing a Message, Specifying a Transformation

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   message := test.message_typ(001, 'NORMAL MESSAGE', 'enqueued to obj_queue');
   enqueue_options.transformation := 'message_order_transform';
   DBMS_AQ.ENQUEUE(
      queue_name              => 'test.priority_queue',
      enqueue_options         => enqueue_options,
      message_properties      => message_properties,
      payload                 => message,
      msgid                   => message_handle);
   COMMIT;
END;
/
```

Enqueuing an Array of Messages

Use the `ENQUEUE_ARRAY` function to enqueue an array of payloads using a corresponding array of message properties.

```
DBMS_AQ.ENQUEUE_ARRAY(
   queue_name                IN   VARCHAR2,
   enqueue_options           IN   enqueue_options_t,
   array_size                IN   PLS_INTEGER,
   message_properties_array  IN   message_properties_array_t,
   payload_array             IN   VARRAY,
   msid_array                OUT  msgid_array_t)
RETURN PLS_INTEGER;
```

The output is an array of message identifiers of the enqueued messages. The function returns the number of messages successfully enqueued.

Array enqueuing is not supported for buffered messages, but you can still use `DBMS_AQ.ENQUEUE_ARRAY()` to enqueue buffered messages by setting `array_size` to `1`.

The `message_properties_array` parameter is an array of message properties. Each element in the payload array must have a corresponding element in this record. All messages in an array have the same delivery mode.

The payload structure can be a VARRAY or nested table. The message IDs are returned into an array of RAW(16) entries of type `DBMS_AQ.msgid_array_t`.

As with array operations in the relational world, it is not possible to provide a single optimum array size that will be correct in all circumstances. Application developers must experiment with different array sizes to determine the optimal value for their particular applications.

See Also:

- "Enqueue Options"
- "Message Properties"

Example 7-11 Enqueuing an Array of Messages

```
DECLARE
  enqueue_options       DBMS_AQ.enqueue_options_t;
  msg_prop_array        DBMS_AQ.message_properties_array_t;
  msg_prop              DBMS_AQ.message_properties_t;
  payload_array         test.msg_table;
  msgid_array           DBMS_AQ.msgid_array_t;
  retval                PLS_INTEGER;
BEGIN
  payload_array  := msg_table(
      message_typ(001, 'MESSAGE  1', 'array enqueued to obj_queue'),
      message_typ(001, 'MESSAGE  2', 'array enqueued to obj_queue'));
  msg_prop_array := DBMS_AQ.message_properties_array_t(msg_prop, msg_prop);

  retval := DBMS_AQ.ENQUEUE_ARRAY(
               queue_name               => 'test.obj_queue',
               enqueue_options          => enqueue_options,
               array_size               => 2,
               message_properties_array => msg_prop_array,
               payload_array            => payload_array,
               msgid_array              => msgid_array);
  COMMIT;
END;/
```

Listening to One or More Queues

This procedure specifies which queue or queues to monitor.

```
DBMS_AQ.LISTEN(
   agent_list             IN    aq$_agent_list_t,
   wait                   IN    BINARY_INTEGER DEFAULT FOREVER,
   listen_delivery_mode   IN    PLS_INTEGER DEFAULT PERSISTENT,
   agent                  OUT   sys.aq$_agent
   message_delivery_mode  OUT   PLS_INTEGER);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

This call takes a list of agents as an argument. Each agent is identified by a unique combination of name, address, and protocol.

You specify the queue to be monitored in the address field of each agent listed. Agents must have dequeue privileges on each monitored queue. You must specify the name of the agent when monitoring multiconsumer queues; but you must not specify an agent name for single-consumer queues. Only local queues are supported as addresses. Protocol is reserved for future use.

Note:

Listening to multiconsumer queues is not supported in the Java API.

The `listen_delivery_mode` parameter specifies what types of message interest the agent. If it is the default `PERSISTENT`, then the agent is informed about persistent messages only. If it is set to `BUFFERED`, then the agent is informed about buffered messages only. If it is set to `PERSISTENT_OR_BUFFERED`, then the agent is informed about both types.

This is a blocking call that returns the agent and message type when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, then only the first agent listed is returned. If there are no messages found when the wait time expires, then an error is raised.

A successful return from the `listen` call is only an indication that there is a message for one of the listed agents in one of the specified queues. The interested agent must still dequeue the relevant message.

Note:

You cannot call `LISTEN` on nonpersistent queues.

Even though both `test.obj_queue` and `test.priority_queue` contain messages (enqueued in Example 7-1 and Example 7-2 respectively) Example 7-12 returns only:

```
Message in Queue: "TEST"."OBJ_QUEUE"
```

If the order of agents in `test_agent_list` is reversed, so `test.priority_queue` appears before `test.obj_queue`, then the example returns:

```
Message in Queue: "TEST"."PRIORITY_QUEUE"
```

See Also:

"AQ Agent Type"

Example 7-12 Listening to a Single-Consumer Queue with Zero Timeout

```
SET SERVEROUTPUT ON
DECLARE
   agent             sys.aq$_agent;
   test_agent_list   DBMS_AQ.aq$_agent_list_t;
BEGIN
   test_agent_list(1) := sys.aq$_agent(NULL, 'test.obj_queue',  NULL);
   test_agent_list(2) := sys.aq$_agent(NULL, 'test.priority_queue', NULL);
   DBMS_AQ.LISTEN(
      agent_list   =>   test_agent_list,
      wait         =>   0,
      agent        =>   agent);
   DBMS_OUTPUT.PUT_LINE('Message in Queue: ' ||  agent.address);
END;
/
```

Dequeuing Messages

This procedure dequeues a message from the specified queue.

```
DBMS_AQ.DEQUEUE(
   queue_name          IN      VARCHAR2,
   dequeue_options     IN      dequeue_options_t,
   message_properties  OUT     message_properties_t,
   payload             OUT     "type_name",
   msgid               OUT     RAW);
```

You can choose to dequeue only persistent messages, only buffered messages, or both. See delivery_mode in the following list of dequeue options.

See Also:

"Message Properties"

Dequeue Options

The dequeue_options parameter specifies the options available for the dequeue operation. It has the following attributes:

- consumer_name

  A consumer can dequeue a message from a queue by supplying the name that was used in the AQ$_AGENT type of the DBMS_AQADM.ADD_SUBSCRIBER procedure or the recipient list of the message properties. If a value is specified, then only those messages matching consumer_name are accessed. If a queue is not set up for multiple consumers, then this field must be set to NULL (the default).

- dequeue_mode

  The dequeue_mode attribute specifies the locking behavior associated with the dequeue. If BROWSE is specified, then the message is dequeued without acquiring any lock. If LOCKED is specified, then the message is dequeued with a write lock that lasts for the duration of the transaction. If REMOVE is specified, then the message is dequeued and deleted (the default). The message can be retained in the queue table based on the retention properties. If REMOVE_NO_DATA is specified, then the message is marked as updated or deleted.

- navigation

  The navigation attribute specifies the position of the dequeued message. If FIRST_MESSAGE is specified, then the first available message matching the search criteria is dequeued. If NEXT_MESSAGE is specified, then the next available message matching the search criteria is dequeued (the default). If the previous message belongs to a message group, then the next available message matching the search criteria in the message group is dequeued.

If `NEXT_TRANSACTION` is specified, then any messages in the current transaction group are skipped and the first message of the next transaction group is dequeued. This setting can only be used if message grouping is enabled for the queue.

- `visibility`

  The `visibility` attribute specifies when the new message is dequeued. If `ON_COMMIT` is specified, then the dequeue is part of the current transaction (the default). If `IMMEDIATE` is specified, then the dequeue operation is an autonomous transaction that commits at the end of the operation. The `visibility` attribute is ignored in `BROWSE` dequeue mode.

  Visibility must always be `IMMEDIATE` when dequeuing messages with delivery mode `DBMS_AQ.BUFFERED` or `DBMS_AQ.PERSISTENT_OR_BUFFERED`.

- `wait`

  The `wait` attribute specifies the wait time if there is currently no message available matching the search criteria. If a number is specified, then the operation waits that number of seconds. If `FOREVER` is specified, then the operation waits forever (the default). If `NO_WAIT` is specified, then the operation does not wait.

- `msgid`

  The `msgid` attribute specifies the message identifier of the dequeued message. Only messages in the `READY` state are dequeued unless `msgid` is specified.

- `correlation`

  The correlation attribute specifies the correlation identifier of the dequeued message. The correlation identifier cannot be changed between successive dequeue calls without specifying the `FIRST_MESSAGE` navigation option.

  Correlation identifiers are application-defined identifiers that are not interpreted by Oracle Database Advanced Queuing. You can use special pattern matching characters, such as the percent sign and the underscore. If more than one message satisfies the pattern, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

  > **Note:**
  >
  > Although dequeue options `correlation` and `deq_condition` are both supported for buffered messages, it is not possible to create indexes to optimize these queries.

- `deq_condition`

  The `deq_condition` attribute is a Boolean expression similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions.

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

  The `deq_condition` attribute cannot exceed 4000 characters. If more than one message satisfies the dequeue condition, then the order of dequeuing is indeterminate, and the sort order of the queue is not honored.

- `transformation`

The `transformation` attribute specifies a transformation that will be applied after the message is dequeued but before returning the message to the caller.

- `delivery_mode`

  The `delivery_mode` attribute specifies what types of messages to dequeue. If it is set to `DBMS_AQ.PERSISTENT`, then only persistent messages are dequeued. If it is set to `DBMS_AQ.BUFFERED`, then only buffered messages are dequeued.

  If it is the default `DBMS_AQ.PERSISTENT_OR_BUFFERED`, then both persistent and buffered messages are dequeued. The `delivery_mode` attribute in the message properties of the dequeued message indicates whether the dequeued message was buffered or persistent.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the message identifier and correlation identifier in dequeue options.

The database consistent read mechanism is applicable for queue operations. For example, a `BROWSE` call may not see a message that is enqueued after the beginning of the browsing transaction.

In a commit-time queue, messages are not visible to `BROWSE` or `DEQUEUE` calls until a deterministic order can be established among them based on an approximate CSCN.

If the `navigation` attribute of the `dequeue_conditions` parameter is `NEXT_MESSAGE` (the default), then subsequent dequeues retrieve messages from the queue based on the snapshot obtained in the first dequeue. A message enqueued after the first dequeue command, therefore, will be processed only after processing all remaining messages in the queue. This is not a problem if all the messages have already been enqueued or if the queue does not have priority-based ordering. But if an application must process the highest-priority message in the queue, then it must use the `FIRST_MESSAGE` navigation option.

Note:

It can also be more efficient to use the `FIRST_MESSAGE` navigation option when there are messages being concurrently enqueued. If the `FIRST_MESSAGE` option is not specified, then Oracle Database Advanced Queuing continually generates the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST_MESSAGE` option is specified, then Oracle Database Advanced Queuing uses a new snapshot for every dequeue command.

Messages enqueued in the same transaction into a queue that has been enabled for message grouping form a group. If only one message is enqueued in the transaction, then this effectively forms a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group locks the entire group. This is useful when all the messages in a group must be processed as a unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use `NEXT_TRANSACTION` to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue times out after the period specified in the `wait` attribute of `dequeue_options`.

Typically, you expect the consumer of messages to access messages using the dequeue interface. You can view processed messages or messages still to be processed by browsing by message ID or by using `SELECT` commands.

Example 7-13 returns the message enqueued in Example 7-1. It returns:

```
From Sender No.1
Subject: TEST MESSAGE
Text: First message to obj_queue
```

See Also:

- "Dequeue Modes"

Dequeuing LOB Type Messages

Example 7-14 creates procedure `blobdequeue()` to dequeue the LOB type messages enqueued in Example 7-4. The actual dequeue is shown in Example 7-15. It returns:

```
Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000
Amount of data read: 2000
```

Dequeuing Grouped Messages

You can dequeue the grouped messages enqueued in Example 7-8 by running Example 7-16. It returns:

```
GROUP 1: Message 1 in group 1
GROUP 1: Message 2 in group 1
GROUP 1: Message 3 in group 1
Finished GROUP 1
GROUP 2: Message 1 in group 2
GROUP 2: Message 2 in group 2
GROUP 2: Message 3 in group 2
Finished GROUP 2
GROUP 3: Message 1 in group 3
GROUP 3: Message 2 in group 3
GROUP 3: Message 3 in group 3
Finished GROUP 3
No more messages
```

Dequeuing from a Multiconsumer Queue

You can dequeue the messages enqueued for `RED` in Example 7-7 by running Example 7-17. If you change `RED` to `GREEN` and then to `BLUE`, you can use it to dequeue their messages as well. The output of the example will be different in each case.

`RED` is a subscriber to the multiconsumer queue and is also a specified recipient of MESSAGE 2, so it gets both messages:

```
Message: MESSAGE 1 .. For queue subscribers
Message: MESSAGE 2 .. For two recipients
No more messages for RED
```

GREEN is only a subscriber, so it gets only those messages in the queue for which no recipients have been specified (in this case, MESSAGE 1):

```
Message: MESSAGE 1 .. For queue subscribers
No more messages for GREEN
```

BLUE, while not a subscriber to the queue, is nevertheless specified to receive MESSAGE 2.

```
Message: MESSAGE 2 .. For two recipients
No more messages for BLUE
```

Example 7-18 browses messages enqueued in Example 7-5 until it finds PINK, which it removes. The example returns:

```
Browsed Message Text: ORANGE enqueued first.
Browsed Message Text: ORANGE also enqueued second.
Browsed Message Text: YELLOW enqueued third.
Browsed Message Text: VIOLET enqueued fourth.
Browsed Message Text: PURPLE enqueued fifth.
Browsed Message Text: PINK enqueued sixth.
Removed Message Text: PINK enqueued sixth.
```

Dequeue Modes

Example 7-19 previews in locked mode the messages enqueued in Example 7-5 until it finds PURPLE, which it removes. The example returns:

```
Locked Message Text: ORANGE enqueued first.
Locked Message Text: ORANGE also enqueued second.
Locked Message Text: YELLOW enqueued third.
Locked Message Text: VIOLET enqueued fourth.
Locked Message Text: PURPLE enqueued fifth.
Removed Message Text: PURPLE enqueued fifth.
```

Example 7-13 Dequeuing Object Type Messages

```
SET SERVEROUTPUT ON
DECLARE
dequeue_options     DBMS_AQ.dequeue_options_t;
message_properties  DBMS_AQ.message_properties_t;
message_handle      RAW(16);
message             test.message_typ;
BEGIN
   dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
   DBMS_AQ.DEQUEUE(
      queue_name          =>     'test.obj_queue',
      dequeue_options     =>     dequeue_options,
      message_properties  =>     message_properties,
      payload             =>     message,
      msgid               =>     message_handle);
   DBMS_OUTPUT.PUT_LINE('From Sender No.'|| message.sender_id);
   DBMS_OUTPUT.PUT_LINE('Subject: '||message.subject);
   DBMS_OUTPUT.PUT_LINE('Text: '||message.text);
   COMMIT;
END;
/
```

Example 7-14 Creating a Dequeue Procedure for LOB Type Messages

```
CREATE OR REPLACE PROCEDURE blobdequeue(msgno IN NUMBER) AS
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   msgid               RAW(16);
   payload             test.lob_typ;
   lob_loc             BLOB;
   amount              BINARY_INTEGER;
   buffer              RAW(4096);
BEGIN
   DBMS_AQ.DEQUEUE(
      queue_name           =>  'test.lob_queue',
      dequeue_options      =>   dequeue_options,
      message_properties   =>   message_properties,
      payload              =>   payload,
      msgid                =>   msgid);
   lob_loc                 :=   payload.data;
   amount                  :=   2000;
   DBMS_LOB.READ(lob_loc, amount, 1, buffer);
   DBMS_OUTPUT.PUT_LINE('Amount of data read: '|| amount);
   COMMIT;
END;
/
```

Example 7-15   Dequeuing LOB Type Messages

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

Example 7-16   Dequeuing Grouped Messages

```
SET SERVEROUTPUT ON
DECLARE
   dequeue_options       DBMS_AQ.dequeue_options_t;
   message_properties    DBMS_AQ.message_properties_t;
   message_handle        RAW(16);
   message               test.message_typ;
   no_messages           exception;
   end_of_group          exception;
   PRAGMA EXCEPTION_INIT (no_messages, -25228);
   PRAGMA EXCEPTION_INIT (end_of_group, -25235);
BEGIN
   dequeue_options.wait       := DBMS_AQ.NO_WAIT;
   dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
   LOOP
     BEGIN
     DBMS_AQ.DEQUEUE(
        queue_name         => 'test.group_queue',
        dequeue_options    => dequeue_options,
        message_properties => message_properties,
        payload            => message,
        msgid              => message_handle);
     DBMS_OUTPUT.PUT_LINE(message.subject || ': ' || message.text );
     dequeue_options.navigation := DBMS_AQ.NEXT_MESSAGE;
     EXCEPTION
       WHEN end_of_group THEN
         DBMS_OUTPUT.PUT_LINE ('Finished ' || message.subject);
         COMMIT;
         dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
     END;
   END LOOP;
   EXCEPTION
     WHEN no_messages THEN
       DBMS_OUTPUT.PUT_LINE ('No more messages');
```

```
END;
/
```

Example 7-17 Dequeuing Messages for RED from a Multiconsumer Queue

```
SET SERVEROUTPUT ON
DECLARE
  dequeue_options       DBMS_AQ.dequeue_options_t;
  message_properties    DBMS_AQ.message_properties_t;
  message_handle        RAW(16);
  message               test.message_typ;
  no_messages           exception;
  PRAGMA EXCEPTION_INIT (no_messages, -25228);
BEGIN
  dequeue_options.wait          := DBMS_AQ.NO_WAIT;
  dequeue_options.consumer_name := 'RED';
  dequeue_options.navigation    := DBMS_AQ.FIRST_MESSAGE;
  LOOP
   BEGIN
    DBMS_AQ.DEQUEUE(
      queue_name         => 'test.multiconsumer_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;
   END;
  END LOOP;
  EXCEPTION
    WHEN no_messages THEN
    DBMS_OUTPUT.PUT_LINE ('No more messages for RED');
  COMMIT;
END;
/
```

Example 7-18 Dequeue in Browse Mode and Remove Specified Message

```
SET SERVEROUTPUT ON
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;
   LOOP
      DBMS_AQ.DEQUEUE(
        queue_name             => 'test.obj_queue',
        dequeue_options        => dequeue_options,
        message_properties     => message_properties,
        payload                => message,
        msgid                  => message_handle);
      DBMS_OUTPUT.PUT_LINE ('Browsed Message Text: ' || message.text);
      EXIT WHEN message.subject = 'PINK';
   END LOOP;
   dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
   dequeue_options.msgid        := message_handle;
   DBMS_AQ.DEQUEUE(
           queue_name          => 'test.obj_queue',
           dequeue_options     => dequeue_options,
           message_properties  => message_properties,
           payload             => message,
            msgid                => message_handle);
   DBMS_OUTPUT.PUT_LINE('Removed Message Text: ' || message.text);
   COMMIT;
END;
/
```

Example 7-19   Dequeue in Locked Mode and Remove Specified Message

```
SET SERVEROUTPUT ON
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             test.message_typ;
BEGIN
   dequeue_options.dequeue_mode := DBMS_AQ.LOCKED;
   LOOP
      DBMS_AQ.dequeue(
        queue_name         => 'test.obj_queue',
        dequeue_options    => dequeue_options,
        message_properties => message_properties,
        payload            => message,
        msgid              => message_handle);
      DBMS_OUTPUT.PUT_LINE('Locked Message Text: ' || message.text);
      EXIT WHEN message.subject = 'PURPLE';
   END LOOP;
   dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
   dequeue_options.msgid        := message_handle;
   DBMS_AQ.DEQUEUE(
     queue_name           => 'test.obj_queue',
     dequeue_options      => dequeue_options,
     message_properties   => message_properties,
     payload              => message,
     msgid                => message_handle);
   DBMS_OUTPUT.PUT_LINE('Removed Message Text: ' || message.text);
   COMMIT;
END;
/
```

Dequeuing an Array of Messages

Use the `DEQUEUE_ARRAY` function to dequeue an array of payloads and a corresponding array of message properties.

```
DBMS_AQ.DEQUEUE_ARRAY(
   queue_name                IN      VARCHAR2,
   dequeue_options           IN      dequeue_options_t,
   array_size                IN      PLS_INTEGER,
   message_properties_array  OUT     message_properties_array_t,
   payload_array             OUT     VARRAY,
   msgid_array               OUT     msgid_array_t)
RETURN PLS_INTEGER;
```

The output is an array of payloads, message IDs, and message properties of the dequeued messages. The function returns the number of messages successfully dequeued.

Array dequeuing is not supported for buffered messages, but you can still use `DBMS_AQ.DEQUEUE_ARRAY()` to dequeue buffered messages by setting `array_size` to `1`.

The payload structure can be a VARRAY or nested table. The message identifiers are returned into an array of RAW(16) entries of type DBMS_AQ.msgid_array_t. The message properties are returned into an array of type DBMS_AQ.message_properties_array_t.

As with array operations in the relational world, it is not possible to provide a single optimum array size that will be correct in all circumstances. Application developers must experiment with different array sizes to determine the optimal value for their particular applications.

All dequeue options available with DBMS_AQ.DEQUEUE are also available with DBMS_AQ.DEQUEUE_ARRAY. You can choose to dequeue only persistent messages, only buffered messages, or both. In addition, the navigation attribute of dequeue_options offers two options specific to DBMS_AQ.DEQUEUE_ARRAY.

When dequeuing messages, you might want to dequeue all the messages for a transaction group with a single call. You might also want to dequeue messages that span multiple transaction groups. You can specify either of these methods by using one of the following navigation methods:

- NEXT_MESSAGE_ONE_GROUP
- FIRST_MESSAGE_ONE_GROUP
- NEXT_MESSAGE_MULTI_GROUP
- FIRST_MESSAGE_MULTI_GROUP

Navigation method NEXT_MESSAGE_ONE_GROUP dequeues messages that match the search criteria from the next available transaction group into an array. Navigation method FIRST_MESSAGE_ONE_GROUP resets the position to the beginning of the queue and dequeues all the messages in a single transaction group that are available and match the search criteria.

The number of messages dequeued is determined by an array size limit. If the number of messages in the transaction group exceeds array_size, then multiple calls to DEQUEUE_ARRAY must be made to dequeue all the messages for the transaction group.

Navigation methods NEXT_MESSAGE_MULTI_GROUP and FIRST_MESSAGE_MULTI_GROUP work like their ONE_GROUP counterparts, but they are not limited to a single transaction group. Each message that is dequeued into the array has an associated set of message properties. Message property transaction_group determines which messages belong to the same transaction group.

Example 7-20 dequeues the messages enqueued in Example 7-11. It returns:

```
Number of messages dequeued: 2
```

See Also:

"Dequeuing Messages"

Example 7-20 Dequeuing an Array of Messages

```
SET SERVEROUTPUT ON
DECLARE
  dequeue_options       DBMS_AQ.dequeue_options_t;
  msg_prop_array        DBMS_AQ.message_properties_array_t :=
                        DBMS_AQ.message_properties_array_t();
  payload_array         test.msg_table;
  msgid_array           DBMS_AQ.msgid_array_t;
  retval                PLS_INTEGER;
BEGIN
  retval := DBMS_AQ.DEQUEUE_ARRAY(
              queue_name               => 'test.obj_queue',
              dequeue_options          => dequeue_options,
              array_size               => 2,
              message_properties_array => msg_prop_array,
              payload_array            => payload_array,
              msgid_array              => msgid_array);
  DBMS_OUTPUT.PUT_LINE('Number of messages dequeued: ' || retval);
END;/
```

Registering for Notification

This procedure registers an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

```
DBMS_AQ.REGISTER(
   reg_list      IN SYS.AQ$_REG_INFO_LIST,
   reg_count     IN NUMBER);
```

Note:

Starting from 12*c* Release 2 (12.2.), the maximum length of user-generated queue names is 122 bytes. See "Creating a Queue".

The `reg_list` parameter is a list of `SYS.AQ$_REG_INFO` objects. You can specify notification quality of service with the `qosflags` attribute of `SYS.AQ$_REG_INFO`.

The `reg_count` parameter specifies the number of entries in the `reg_list`. Each subscription requires its own `reg_list` entry. Interest in several subscriptions can be registered at one time.

When PL/SQL notification is received, the Oracle Database Advanced Queuing message properties descriptor that the callback is invoked with specifies the `delivery_mode` of the message notified as `DBMS_AQ.PERSISTENT` or `DBMS_AQ.BUFFERED`.

If you register for e-mail notifications, then you must set the host name and port name for the SMTP server that will be used by the database to send e-mail notifications. If required, you should set the send-from e-mail address, which is set by the database as the `sent from` field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, then you might want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

An internal queue called `SYS.AQ_SRVNTFN_TABLE_Q` stores the notifications to be processed by the job queue processes. If notification fails, then Oracle Database Advanced Queuing retries the failed notification up to `MAX_RETRIES` attempts.

Note:

You can change the `MAX_RETRIES` and `RETRY_DELAY` properties of `SYS.AQ_SRVNTFN_TABLE_Q`. The new settings are applied across all notifications.

See Also:

- "AQ Registration Information Type" for more information on `SYS.AQ$_REG_INFO` objects
- "AQ Notification Descriptor Type" for more information on the message properties descriptor

Example 7-21 Registering for Notifications

```
DECLARE
  reginfo             sys.aq$_reg_info;
  reg_list            sys.aq$_reg_info_list;
BEGIN
  reginfo := sys.aq$_reg_info(
                      'test.obj_queue',
                      DBMS_AQ.NAMESPACE_ANONYMOUS,
                      'http://www.company.com:8080',
                      HEXTORAW('FF'));
  reg_list  := sys.aq$_reg_info_list(reginfo);
  DBMS_AQ.REGISTER(
    reg_list     => reg_list,
    reg_count    => 1);
  COMMIT;
END;
/
```

Unregistering for Notification

This procedure unregisters an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

```
DBMS_AQ.UNREGISTER(
   reg_list      IN SYS.AQ$_REG_INFO_LIST,
   reg_count     IN NUMBER);
```

Posting for Subscriber Notification

This procedure posts to a list of anonymous subscriptions, allowing all clients who are registered for the subscriptions to get notifications of persistent messages.

```
DBMS_AQ.POST(
  post_list       IN   SYS.AQ$_POST_INFO_LIST,
  post_count      IN   NUMBER);
```

This feature is not supported with buffered messages.

The `count` parameter specifies the number of entries in the `post_list`. Each posted subscription must have its own entry in the `post_list`. Several subscriptions can be posted to at one time.

The `post_list` parameter specifies the list of anonymous subscriptions to which you want to post. It has three attributes:

- `name`

  The `name` attribute specifies the name of the anonymous subscription to which you want to post.

- `namespace`

  The `namespace` attribute specifies the namespace of the subscription. To receive notifications from other applications through `DBMS_AQ.POST` the namespace must be `DBMS_AQ.NAMESPACE_ANONYMOUS`.

- `payload`

  The `payload` attribute specifies the payload to be posted to the anonymous subscription. It is possible for no payload to be associated with this call.

This call provides a best-effort guarantee. A notification goes to registered clients at most once. This call is primarily used for lightweight notification. If an application needs more rigid guarantees, then it can enqueue to a queue.

Example 7-22 Posting Object-Type Messages

```
DECLARE
  postinfo            sys.aq$_post_info;
  post_list           sys.aq$_post_info_list;
BEGIN
  postinfo  := sys.aq$_post_info('test.obj_queue',0,HEXTORAW('FF'));
  post_list := sys.aq$_post_info_list(postinfo);
  DBMS_AQ.POST(
    post_list       => post_list,
    post_count      => 1);
  COMMIT;
END;
/
```

Adding an Agent to the LDAP Server

This procedure creates an entry for an Oracle Database Advanced Queuing agent in the LDAP server.

```
DBMS_AQ.BIND_AGENT(
   agent        IN SYS.AQ$_AGENT,
   certificate  IN VARCHAR2 default NULL);
```

The `agent` parameter specifies the Oracle Database Advanced Queuing Agent that is to be registered in Lightweight Directory Access Protocol (LDAP) server.

The `certificate` parameter specifies the location (LDAP distinguished name) of the `OrganizationalPerson` entry in LDAP whose digital certificate (attribute `usercertificate`) is to be used for this agent. For example, "`cn=OE, cn=ACME, cn=com`" is a distinguished name for a `OrganizationalPerson OE` whose certificate will be used with the specified agent. If the agent does not have a digital certificate, then this parameter is defaulted to null.

See Also:

"AQ Agent Type"

Removing an Agent from the LDAP Server

This procedure removes the entry for an Oracle Database Advanced Queuing agent from the LDAP server.

```
DBMS_AQ.UNBIND_AGENT(
   agent     IN SYS.AQ$_AGENT);
```

8 Oracle Transactional Event Queues and Advanced Queuing Performance and Scalability

These topics discuss performance and scalability issues relating to Transactional Event Queues (TxEventQ) and Advanced Queuing (AQ).

Transactional Event Queues

A transactional event queue (TxEventQ) increases enqueue-dequeue throughput, especially across Oracle Real Application Clusters (Oracle RAC) instances, because messages from different enqueue sessions are allowed to be dequeued in parallel. Each event stream of the queue is ordered based on enqueue time within a session and ordering across event streams is best-effort. TxEventQs automatically manage table partitions so that enqueuers and dequeuers do not contend among themselves. In addition, TxEventQs use an in-memory message cache to optimize performance and reduce the disk and CPU overhead of enqueues and dequeues.

The advantages and tradeoffs of TxEventQs include the following:

- TxEventQs provide scalability of a single queue on Oracle RAC, especially in the case where each subscriber has multiple dequeuers on each instance.
- Oracle Real Application Clusters (Oracle RAC)s trades off increased memory usage to obtain performance.

This section contains the following topics:

- Transactional Event Queues Tuning

Transactional Event Queues and the Message Cache

TxEventQs introduce a special purpose message cache which lets you trade off SGA usage for increased throughput, reduced latency, and increased concurrency. When combined with partitioning, the message cache reduces the need for some queries, DML operations, and indexes. The message cache is most effective when all dequeuers keep up with enqueuers and when the message cache is big enough to store messages (including payloads) for each TxEventQ's enqueuers and dequeuers. The message cache uses the Streams pool. If TxEventQs share the Streams pool on the same instance as Streams replication functionality, you can use `DBMS_AQADM` procedures such as `SET_MIN_STREAMS_POOL` and `SET_MAX_STREAMS_POOL` to fine tune the allocation of Streams Pool memory.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information

Transactional Event Queues and Enqueuing / Dequeuing Messages

To improve throughput and reduce overhead and latency, enqueues and dequeues are optimized to use the message cache, the rules engine, and background processing when possible. For example,

- TxEventQs take advantage of new rules engine improvements
- a message that has its payload in the message cache does not have to be re-read from disk during a dequeue
- dequeue by correlation id or other JMS properties can often be evaluated without accessing the disk
- partition operations on TxEventQs implements efficient bulk processing.

Transactional Event Queues and Native JMS Support

TxEventQs have native support for:

- Non-Durable Subscribers
- JMS payloads
- Priorities

TxEventQs support both persistent and nonpersistent messages. Nonpersistent messages are stored in memory inside the message cache and are not stored on disk. As a result, nonpersistent messages are lost upon instance crash or shutdown.

TxEventQs natively support two kinds of subscribers to meet the JMS requirements:

- Non-durable subscribers: These subscribers receive messages on their chosen topic, only if the messages are published while the subscriber is active. This subscription is not sharable among different sessions.
- Durable subscribers: These subscribers receive all the messages published on a topic, including those that are published while the subscriber is inactive. Multiple database sessions can share the same subscription.

TxEventQs do not use ADTs to store the JMS payload. The JMS message is stored in scalar columns of the database. JMS message types such as `TEXT`, `BYTES`, `MAP`, `STREAM` and `OBJECT` store the JMS payload in scalar `TEXT`/`RAW` or `CLOB`/`BLOB` columns in the queue table depending on payload size and type. The JMS message properties are stored in a `CLOB` (SecureFile) column in the queue table with attribute access functions defined for the user defined properties. The payload and user properties are pickled into `RAW`, `VARCHAR2` or Secure File columns instead of being stored as an ADT. JMS Header properties and JMS provider information are stored in their own scalar columns.

TxEventQs support integer priority values ranging between 0 (lowest priority) and 9 (highest priority), with the default being priority 4, as defined by the JMS standard.

Transactional Event Queues and Partitioning

TxEventQs automatically manage the underlying partitioned tables used for the queue table. Such partition management may occur in the foreground or the background. Each event stream provides session-level ordering of enqueued messages. Each enqueuing session is assigned an event stream. Each event stream is composed of a series of event stream partitions. Each event stream partition maps to a single partition. Messages are automatically assigned to a table partition upon enqueue.

New partitions are automatically created as needed, as when the queue table needs to grow when dequeuers do not keep up with enqueuers. Partitions are truncated and reused when all messages in the partition are dequeued and no longer needed. The message cache automatically loads messages from partitions into memory as required by dequeuers. Global indexes should not be created on the partitioned table underlying a TxEventQ. Local indexes are not typically recommended on the partitioned table either. If such indexes are desired and result in performance degradation, then AQ queues should be considered.

Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC)

TxEventQs automatically provides enqueue session ordering while avoiding cross-instance communication when possible. Sometimes cross instance communication is required. For example, if a TxEventQ has a single enqueuing session on one Oracle RAC instance and a single dequeuing session on another instance, then TxEventQs will forward messages between the Oracle RAC instances. The forwarding of messages is non-atomic to the enqueuing transaction to improve performance. Dequeuers may get an ORA-25228 if they are connected to an instance that has no messages in its event streams.

In most cases, consider having multiple dequeuers for each subscriber or single consumer queue on each Oracle RAC instance to improve throughput and reduce cross-instance overhead. An exception to this guideline is when you are using dequeue selectors that specify a single message. If you want to dequeue a message from a TxEventQ by its message identifier in an Oracle RAC database, then you have to connect to the instance that is assigned dequeue ownership for the event stream containing the message. Otherwise, the message will not be available for dequeue to the dequeue session. If all dequeues are performed at a single instance, then messages will be automatically forwarded to this instance. Hence, for a single-consumer TxEventQ that extensively dequeues by message ID, consider having all dequeue sessions for the TxEventQ connect to a single instance. Similarly, for a multiconsumer TxEventQ that extensively dequeues by message ID, consider having all dequeue sessions for each subscriber connect to a single instance. Services can be used to simplify connecting dequeue sessions to a particular instance.

Transactional Event Queues and Message Retention

Starting from Oracle Database Release 21c, message retention is supported by TxEventQ . AQ queue already has this feature.

Message retention is the time for which a message is retained in the TxEventQ after being enqueued or dequeued as desired. The default is 0, which means that the message will be removed as soon as possible after it is dequeued by all of its subscribers. This helps users to retain the messages in the queue even after they are processed.

Applications can specify retention time while creating a TxEventQ. Applications can change the retention time and its type as needed after creation of the TxEventQ.

TxEventQ supports only dequeue time based retention. A event stream partition stores a set of messages. The event stream partition will be removed from the queue when the highest dequeue time for any message-subscriber pair in that event stream partition plus retention time is over. This scheme will ensure consumption of messages before retention comes in play. This is also the only retention policy available in AQ queues.

Transactional Event Queues and Seekable Subscribers

A seek operation for a subscriber can be for all event streams (queue level seek) or a set of specific event streams of choice (event stream level seek).

All the dequeue calls after a seek operation would dequeue messages from the seek point onwards. All the messages below seek point will never be dequeued or browsed by the subscriber unless the subscriber seeks back again.

Seek Granularity

A subscriber can perform seek in all event streams or a set of event streams of choice in the queue. The choice of the message to seek to can be explicitly specified in the seek operation or can be deduced from the inputs of seek operation.

Following are the different types of seek option inputs.

- Seek to end – With this seek option, the subscriber is not interested in existing messages. The subscriber will be able to dequeue only newly enqueued messages after the seek operation. This is the default behavior when a new subscriber is created.
- Seek to start - With this seek option, the subscriber is interested in existing messages including "retained". The subscriber will also be able to dequeue newly enqueued messages after the seek operation.
- Seek to a specific time - With this seek option, the subscriber is interested in existing messages including "retained" with enqueue time higher than input time. Seek stops if start or end is reached.
- Seek to a specific message – With this seek option, the subscriber is interested in existing messages including "retained" from the input message onwards. The input in this case is a specific message id so this seek always a event stream level seek. A separate unique message id per event stream is specified in the input for all the event streams on which seek needs to be performed.

Message Ordering

A seek action can break message ordering as it results in out of order dequeues. If the new seek point is a message which is not the first message of an enqueue transaction or an enqueue session and messages from new seek point onwards are dequeued, then the application can get only some messages of the enqueue session or enqueue transaction as remaining messages were enqueued before the new seek point.

It is application's responsibility to either choose correct seek point or be tolerant to such behavior.

Transactional Event Queues Restrictions

The following Oracle Database features are not currently supported for TxEventQs:

- Transaction grouping
- Anonymous posting for subscriber notification and OCI callback notification are not supported. PL/SQL callback notification is supported.
- Messaging Gateway
- Oracle extensions for JMS such as JMS propagation and remote subscribers
- Multiple queues per queue table. TxEventQs are created via the `CREATE_TRANSACTIONAL_EVENT_QUEUE` interface.
- Ordering other than message priority followed by enqueue time (as specified in the JMS standard)
- The JDBC thick (OCI) driver.
- Propagation between TxEventQ and AQ queues
- Message transformations

Transactional Event Queues Tuning

TxEventQs perform best under the following conditions:

- Dequeuers for each subscriber are located on each instance
- Subscribers keep up with the enqueuers. Consider having multiple dequeuers for each subscriber on each Oracle RAC instance

The message cache is most effective when dequeuers keep up with enqueuers and where the cache is big enough to store messages (including payloads) for each TxEventQ's enqueuers and dequeuers. When using TxEventQs, Oracle requires that you do one of the following:

- Setting parameter `STREAMS_POOL_SIZE`

  This parameter controls the size of shared memory available to the Oracle Database for the TxEventQ message cache. If unspecified, up to 10% of the shared pool size may be allocated for the Streams pool.

  Oracle's Automatic Shared Memory Management feature manages the size of the Streams pool when the `SGA_TARGET` initialization parameter is set to a nonzero value. If the `STREAMS_POOL_SIZE` initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Streams pool.

  If the `STREAMS_POOL_SIZE` initialization parameter is set to a nonzero value, and the `SGA_TARGET` parameter is set to `0` (zero), then the Streams pool size is the value specified by the `STREAMS_POOL_SIZE` parameter, in bytes.

  If both the `STREAMS_POOL_SIZE` and the `SGA_TARGET` initialization parameters are set to `0` (zero), then, by default, the first use of the Streams pool in a database transfers an amount of memory equal to 10% of the shared pool from the buffer cache to the Streams pool.

See Also:

– `DBMS_AQADM.set_min_streams_pool()` and `DBMS_AQADM.set_max_streams_pool( )` in *Oracle Database PL/SQL Packages and Types Reference* for a finer grained control over `STREAMS_POOL` sharing with Streams processing.

- Turning on SGA autotuning

  Oracle will automatically allocate the appropriate amount of memory from the SGA for the Streams pool, based on Streams pool usage as well as usage of other components that use the SGA. Examples of such other components are buffer cache and library cache. If `STREAMS_POOL_SIZE` is specified, it is used as the lower bound.

- Manually tuning TxEventQ queues

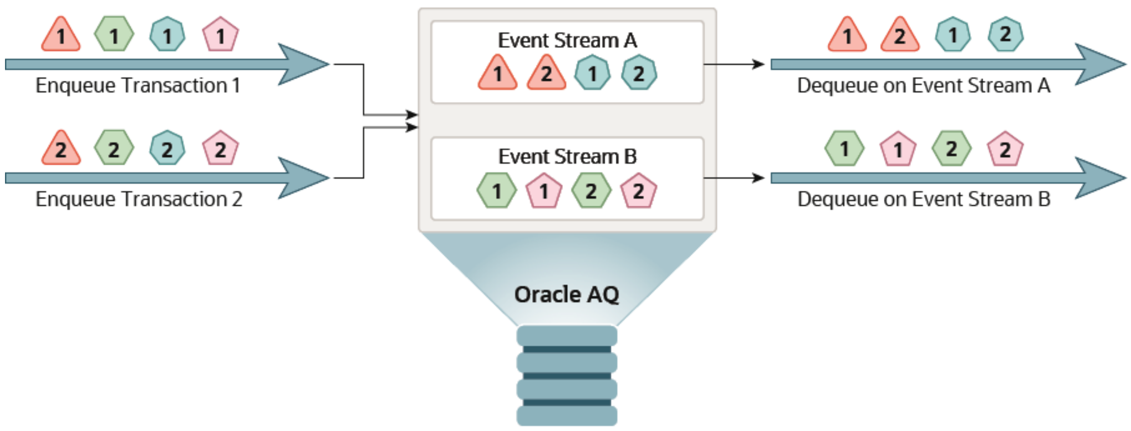
  TxEventQs can be tuned by allocating `STREAMS_POOL` memory for the message cache. The view `GV$AQ_MESSAGE_CACHE_ADVICE` provides advice on how much `STREAMS_POOL` should be allocated for TxEventQs based on a snapshot of the current messaging load. During periods of high load, select the columns `INST_ID`, `SIZE_FOR_ESTIMATE`, and `ESTD_SIZE_TYPE`. `ESTD_SIZE_TYPE` is one of three values: `MINIMUM`, `PREFERRED`, or `MAXIMUM`. Find the maximum value of `SIZE_FOR_ESTIMATE` across Oracle RAC instances for each of the `ESTD_SIZE_TYPE` values. It is highly recommended that `STREAMS_POOL` be set at least to the `MINIMUM` recommendation to provide any message cache performance gains. There is little additional performance gains to setting `STREAMS_POOL` greater than the `MAXIMUM` recommendation value. Setting `STREAMS_POOL` to the `PREFERRED` recommendation tries to provide a reasonable space-performance tradeoff. If the `MAXIMUM` size recommendation is much greater than the `PREFERRED` recommendation, then check that the TxEventQ has no orphaned subscribers, or whether more dequeuers should be added to the instance, so that dequeuers can keep up with the enqueue load. `STREAMS_POOL` tuning should be done over multiple periods of high load and whenever messaging load characteristics change.

User Event Streaming

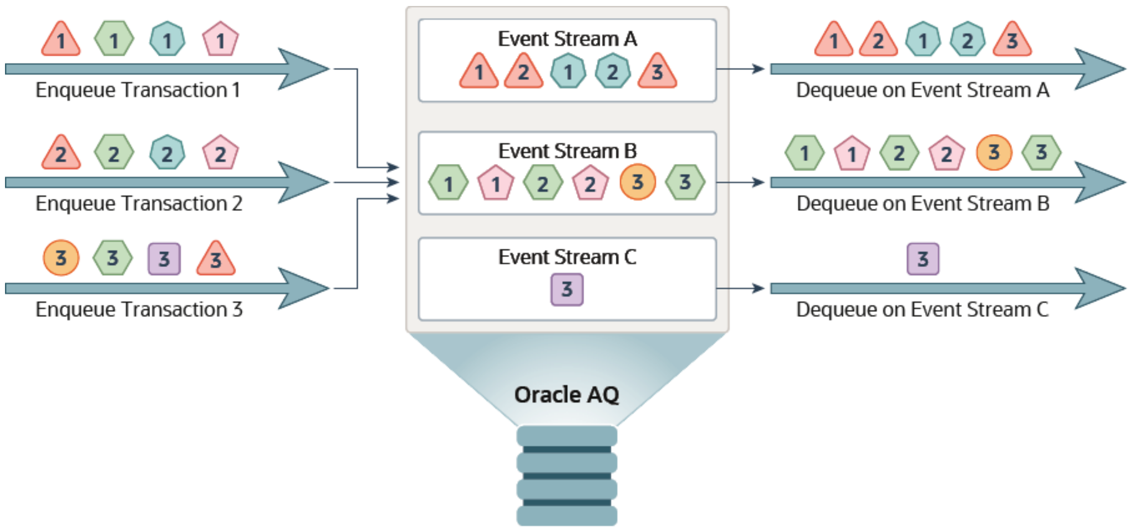
An application can decide the way messages are event streamed in the TxEventQ. In such cases, the application explicitly specifies to enqueue a message in a specific event stream.

For example, assume the application has four types of messages with different keys named red, green, blue, and pink. Each enqueue session can enqueue any of those messages in a transaction. Event Stream A is set to store the red and blue messages. Event Stream B is set to store the green and pink messages. Also, each event stream is set to have only one active dequeue session for a single-consumer queue or JMS Queue. Similarly, each event stream is set to have only one dequeue session per subscriber for a multi-consumer queue or JMS Topic. That dequeue session will stick to that event stream for the dequeuer session’s lifetime.

In the following examples, enqueue transactions are performing enqueues in parallel.



Applications can add new event streams at run time. Applications can also add new types of messages at run time by adding new keys. For example, two new types are introduced with keys orange and purple, and a third Event Stream C is added. Event Stream B is set to store the orange messages. Event Stream C is set to store the purple messages.



In an Oracle RAC database, an event stream is always owned by an instance. Initially, the event stream is owned by the instance where the first message is enqueued in that event stream. The owner instance of the event stream may change when database instances are shut down.

With user event streaming, a user can attempt to enqueue messages in an event stream which is not owned by the instance in which the session is running. In such cases, a cross instance enqueue is triggered. To support cross instance enqueues, the enqueue requests received at other instances are forwarded to the `OWNER INSTANCE` of the event stream over the cluster interconnect for the Oracle RAC database. The `REMOTE_LISTENER` parameter in listener.ora must also be set to enable forwarding of cross instance enqueue requests to the correct instance. Internally, TxEventQ queues on an Oracle RAC database may use database links between instances. Definer's rights PL/SQL packages that perform cross instance enqueues in TxEventQ queues on an Oracle RAC database must grant `INHERIT REMOTE PRIVILEGES` to users of the package.

Example 8-1 Setting REMOTE_LISTENER Parameter for Cross Instance Enqueues

In order to enable cross instance enqueue for TxEventQ, the `REMOTE_LISTENER` parameter must be set for each instance to have listener addresses for every other instance available in Oracle RAC.

For example, consider an Oracle RAC cluster with 4 nodes. In such setup, `REMOTE_LISTSENER` at each node should have listener addresses of all four nodes of the Oracle RAC as follows.

```
REMOTE_LISTENER = (DESCRIPTION= (ADDRESS_LIST=
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node1 PORT>) (HOST=<node1 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node2 PORT>) (HOST=<node2 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node3 PORT>) (HOST=<node3 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node4 PORT>) (HOST=<node4 IP>))
))
```

Alternatively, if TNS aliases are set with listener addresses of Oracle RAC nodes in `tnsnames.ora` as follows,

```
nd1=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node1 PORT>)(HOST=<node1 IP>)))
nd2=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node2 PORT>)(HOST=<node2 IP>)))
nd3=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node3 PORT>)(HOST=<node3 IP>)))
nd4=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node4 PORT>)(HOST=<node4 IP>)))
```

then, `REMOTE_LISTENER` can be set as:

```
REMOTE_LISTENER='nd1,nd2,nd3,nd4'
```

Once `REMOTE_LISTENER` is set correctly at each node, then a message with any key can be enqueued at any instance using the AQJMS client library and the JDBC thin driver in TxEventQ. In the example below, key `"RED"` can be enqueued at any Oracle RAC instance. It will be published in the correct event stream which stores all `"RED"` messages. Please note that it is advisable that a message publisher connects to the owner instance of the event stream to achieve high performance.

```
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(...);
t_conn = tc_fact.createTopicConnection();
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);

/* Create TextMessage */
TextMessage     text_message = jms_sess.createTextMessage();

/* Set key as correlation */
text_message.setJMSCorrelationID("RED");

/* Publish  */
publisher1.publish(text_message);
```

Limitations of User Event Streaming

User event streaming has the following limitations:

- Cross instance enqueues are not enabled for PL/SQL enqueue calls.
- Cross instance enqueues are not enabled for array enqueues.

Cross instance enqueues can be done through Java and OCI clients.

> **See Also:**
>
> DBMS_AQADM in *Oracle Database PL/SQL Packages and Types Reference* for more information.

AQ Queues

This section includes the following topics:

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

Persistent Messaging Basic Tuning Tips

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

> **See Also:**
>
> *Oracle Database Performance Tuning Guide* for tuning recommendations

Memory Requirements

Streams pool size should be at least 20 MB for optimal multi-consumer dequeue performance in a non-Oracle RAC database.

Persistent queuing dequeue operations use the streams pool to optimize performance, especially under concurrency situations. This is, however, not a requirement and the code automatically switches to a less optimal code path.

TxEventQs introduces a message cache for optimal performance of high throughput messaging systems. Ideally the Streams pool size should be large enough to cache the expected backlog of messages in TxEventQs.

Using Storage Parameters

Storage parameters can be specified when creating a queue table using the `storage_clause` parameter.

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

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

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

Storage parameters specified at queue table creation are shared by the queue table, IOTs and indexes. These may be individually altered by an online redefinition using `DBMS_REDEFINITION`.

I/O Configuration

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

See Also:

"I/O Configuration and Design" in *Oracle Database Performance Tuning Guide*

Running Enqueue and Dequeue Processes Concurrently in a Single Queue Table

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

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

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

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

For information about tuning TxEventQs refer to Transactional Event Queues Tuning.

Running Enqueue and Dequeue Processes Serially in a Single Queue Table

When enqueue and dequeue processes are running serially, contention on the same data segment is lower than in the case of concurrent processes. The total time taken to deliver messages by the system, however, is longer than when they run concurrently.

Increasing the number of processes helps both enqueuing and dequeuing. The message throughput rate may be higher for enqueuers than for dequeuers when the number of processes is increased, especially with single consumer queues. Dequeue processes on multi-consumer queues scale much better.

Creating Indexes on a Queue Table

Creating an index on a queue table is useful if you meet these conditions.

- Dequeue using correlation ID

  An index created on the column `corr_id` of the underlying queue table `AQ$_`*`QueueTableName`* expedites dequeues.
- Dequeue using a condition

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

Other Tips for Queues

These are some other persistent messaging basic tuning tips.

- Ensure that statistics are being gathered so that the optimal query plans for retrieving messages are being chosen. By default, queue tables are locked out from automatic gathering of statistics. The recommended use is to gather statistics with a representative queue message load and lock them.
- The queue table indexes and IOTs are automatically coalesced by AQ background processes. However, they must continue to be monitored and coalesced if needed. With automatic space segment management (ASSM), an online shrink operation may be used for the same purpose. A well balanced index reduces queue monitor CPU consumption, and ensures optimal enqueue-dequeue performance.
- Ensure that there are enough queue monitor processes running to perform the background tasks. The queue monitor must also be running for other crucial background activity. Multiple `qmn` processes share the load; make sure that there are enough of them. These are auto-tuned, but can be forced to a minimum number, if needed.
- It is recommended that dequeue with a wait time is only used with dedicated server processes. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes.
- Long running dequeue transactions worsen dequeue contention on the queue, and must be avoided.
- Batching multiple dequeue operations on multi-consumer queues into a single transaction gives best throughput.

- Use NEXT as navigation mode, if not using message priorities. This offers the same semantics but improved performance.
- Use the REMOVE_NODATA dequeue mode if dequeuing in BROWSE mode followed by a REMOVE.

Propagation Tuning Tips

Propagation can be considered a special kind of dequeue operation with an additional INSERT at the remote (or local) queue table. Propagation from a single schedule is not parallelized across multiple job queue processes. Rather, they are load balanced.Propagation can be considered a special kind of dequeue operation with an additional INSERT at the remote (or local) queue table. Propagation from a single schedule is not parallelized across multiple job queue processes. Rather, they are load balanced.

For better scalability, configure the number of propagation schedules according to the available system resources (CPUs).

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

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

Buffered Messaging Tuning

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

Persistent Messaging Performance Overview for Queues

When persistent messages are enqueued, they are stored in database tables. The performance characteristics of queue operations on persistent messages are similar to underlying database operations.

The code path of an enqueue operation is comparable to SELECT and INSERT into a multicolumn queue table with three index-organized tables. The code path of a dequeue operation is comparable to a SELECT operation on the multi-column table and a DELETE operation on the dequeue index-organized table. In many scenarios, for example when Oracle RAC is not used and there is adequate streams pool memory, the dequeue operation is optimized and is comparable to a SELECT operation on a multi-column table.

Note:

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

Queues and Oracle Real Application Clusters

Oracle Real Application Clusters (Oracle RAC) can be used to ensure highly available access to queue data.

The entry and exit points of a queue, commonly called its tail and head respectively, can be extreme hot spots. Because Oracle RAC may not scale well in the presence of hot spots, limit usual access to a queue from one instance only. If an instance failure occurs, then messages managed by the failed instance can be processed immediately by one of the remaining instances. If AQ queues are experiencing hot spots, then consider using TxEventQs instead.

You can associate Oracle RAC instance affinities with 8.1-compatible queue tables. If you are using `q1` and `q2` in different instances, then you can use `ALTER_QUEUE_TABLE` or `CREATE_QUEUE_TABLE` on the queue table and set `primary_instance` to the appropriate `instance_id`.

See Also:

- Creating a Queue Table
- Altering a Queue Table
- Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC)

Oracle Database Advanced Queuing in a Shared Server Environment

Queue operation scalability is similar to the underlying database operation scalability.

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

See Also:

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

Performance Views

Oracle provides these views to monitor system performance and troubleshooting.

- V$AQ_MESSAGE_CACHE_STAT: Memory Management for Sharded Queues
- V$AQ_SHARDED_SUBSCRIBER_STAT: Sharded Queue Subscriber Statistics
- V$AQ_MESSAGE_CACHE_ADVICE: Simulated Metrics
- V$AQ_REMOTE_DEQUEUE_AFFINITY: Dequeue Affinity Instance List

- V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance
- V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance
- V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance
- V$BUFFERED_QUEUES: All Buffered Queues in the Instance.
- V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance
- V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance
- V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues
- V$AQ: Number of Messages in Different States in Database
- V$AQ_BACKGROUND_COORDINATOR: Performance Statistics for AQ's Master Background Coordinator Process (AQPC)
- V$AQ_JOB_COORDINATOR: Performance Statistics per Coordinator
- V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations
- V$AQ_SERVER_POOL: Performance Statistics for all Servers
- V$AQ_CROSS_INSTANCE_JOBS: Cross Process Jobs Description
- V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations
- V$AQ_NOTIFICATION_CLIENTS: Secure OCI Client Connections
- V$AQ_SUBSCRIBER_LOAD: Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER: Non-Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER_LWM: LWM of Non Durable Subscriber
- V$AQ_MESSAGE_CACHE: Performance Statistics

These views are integrated with the Automatic Workload Repository (AWR). Users can generate a report based on two AWR snapshots to compute enqueue rate, dequeue rate, and other statistics per queue/subscriber.

Migrating from AQ to TxEventQ

- Flowchart: Migration from AQ to TxEventQ
- Example Walkthrough
- Steps to Migrate from AQ to TxEventQ
- Overview of How Migration Functions
- Limitations and Workarounds

Transactional Event Queues (TxEventQ) is the next generation messaging system by Oracle that offers many benefits and features over Advanced Queuing (AQ), like scalability, performance, key-based sharding, and Kafka compatibility with a Java client. The online migration tool offers user-friendly interfaces designed to streamline the process of migrating from AQ to TxEventQ. Its intuitive user experience simplifies even the most intricate scenarios, sparing users the need to delve into the complexities of AQ objects. The tool incorporates built-in compatibility diagnostics between AQ and TxEventQ. This diagnosis involves the validation of AQ metadata and data, enabling interactive solutions to any encountered issues.

Furthermore, the tool supplies straightforward procedures to address fallback and recovery scenarios in the event of any potential failures.

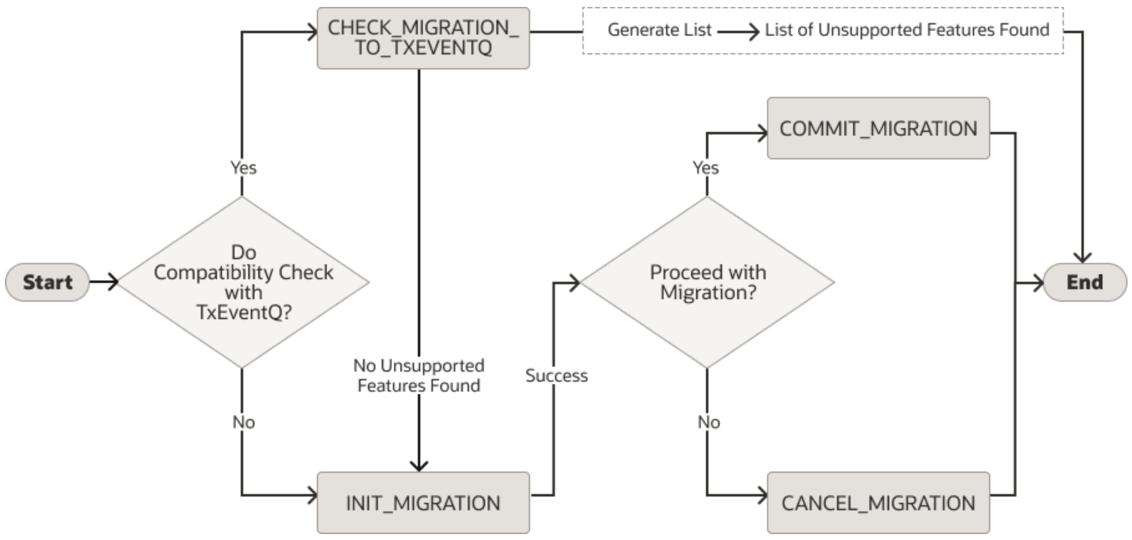
Upgrading AQ to TxEventQ is not a standard procedure like typical upgrades due to substantial feature disparities, significant data representation changes, and other factors. Manually migrating entails system downtime and a deep understanding of the internal mechanisms of both AQ and TxEventQ. The intricacies of handling internal data from various versions of AQ or TxEventQ, alongside existing messages, can present significant challenges. Moreover, rewriting applications to integrate TxEventQ can be expensive. To address these complexities, Oracle Database 23ai introduces the PL/SQL package `DBMS_AQMIGTOOL`, designed to automate the various steps involved in migration. This package offers multiple migration modes, including `AUTOMATIC`, `INTERACTIVE`, `OFFLINE`, and `ONLY_DEFINITION`, providing flexibility to align with specific requirements.

The migration tool interface provides the following functionalities:

- Inspect AQ definition and data, and report unsupported features.
- Approve or prevent migration based on detected unsupported features.
- Initiate migration with modes: `AUTOMATIC`, `INTERACTIVE`, `OFFLINE`, `ONLY_DEFINITION`.
- Offer migration commit or AQ fallback via the user-initiated procedure in the migration interface.
- Keep track of messages that are not currently in the `PROCESSED` state in both the AQ and TxEventQ.
- Migration history record of any queue.
- Provide an option to purge AQ messages if they don't want to consume old messages.
- The migration procedures support rolling upgrades. The migration procedures also support Oracle GoldenGate replicas(OGG) replication.

Flowchart: Migration from AQ to TxEventQ

Figure 8-1 Flowchart: Migration from AQ to TxEventQ


CHECK_MIGRATION_TO_TXEVENTQ
Generate List
List of Unsupported Features Found
COMMIT_MIGRATION
Yes
Yes
Start
Do Compatibility Check with TxEventQ?
Proceed with Migration?
End
No Unsupported Features Found
Success
No
No
INIT_MIGRATION
CANCEL_MIGRATION

Example Walkthrough

Let's go through a few hypothetical scenarios to enhance our understanding of how to use `DBMS_AQMIGTOOL` procedures.

Scenario 1

Consider a scenario involving a user named `aquser` with an AQ named `JSON_QUEUE`. This AQ stores customer events in JSON payload format. If the user wishes to determine whether any features used by AQ are unsupported in TxEventQ, then they can run the `DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ` procedure.

```
SQL> DECLARE
  2    migration_report sys.TxEventQ_MigReport_Array := sys.TxEventQ_MigReport_Array();
  3  BEGIN
  4    DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ('aquser', 'JSON_QUEUE', 
migration_report);
  5    dbms_output.put_line('Migration report unsupported events count: ' || 
migration_report.COUNT);
  6  
  7  END;
  8  /
Migration report unsupported events count: 0

PL/SQL procedure successfully completed.
```

After examining `JSON_QUEUE`, it is evident that no unsupported features are present. Consequently, the user can proceed with migration by initiating the `DBMS_AQMIGTOOL.INIT_MIGRATION` procedure.

```
SQL> EXECUTE DBMS_AQMIGTOOL.INIT_MIGRATION(cqschema => 'aquser', cqname => 'JSON_QUEUE');

PL/SQL procedure successfully completed.

SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues where 
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> FALSE
JSON_QUEUE_M -> TRUE
```

The migration process begins by creating an interim TxEventQ `JSON_QUEUE_M` (with the default suffix `M`) with the same configuration as AQ. Upon executing this procedure, the user can carry out data manipulation operations such as enqueue and dequeue.

Note:

No adjustments are required in the enqueue and dequeue calls; the workload can proceed unchanged.

Users can verify successful evaluation by querying in the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name = 
'JSON_QUEUE' AND event = 'Init_Migration';
```

```
EVENT
--------------------------------------------------------------------------------
Init_Migration
```

Pre-requisite requirement for `DBMS_AQMIGTOOL.COMMIT_MIGRATION` is to have empty AQ (all messages are either in the `PROCESSED` or `EXPIRED` state). The user can utilize the `DBMS_AQMIGTOOL.CHECK_MIGRATED_MESSAGES` procedure to monitor messages within the AQ and interim TxEventQ.

```
SQL> DECLARE
     2    migrated_q_msg_cnt    number := 0;
     3    aq_msg_cnt            number := 0;
     4    BEGIN
     5        DBMS_AQMIGTOOL.CHECK_MIGRATED_MESSAGES(
     6                    cqschema                  => 'aquser',
     7                    cqname                    => 'JSON_QUEUE',
     8                    txeventq_migrated_message  => migrated_q_msg_cnt,
     9                    cq_pending_messages        => aq_msg_cnt);
    10       dbms_output.put_line('AQ ready state message count: ' || aq_msg_cnt);
    11       dbms_output.put_line('Migrated TxEventQ message count: ' ||
migrated_q_msg_cnt);
    12    END;
    13    /
AQ ready state message count: 1000
Migrated TxEventQ message count: 3500

PL/SQL procedure successfully completed.
```

Once the `READY` state message count in AQ reaches zero, the user has the option to complete the migration by executing the `DBMS_AQMIGTOOL.COMMIT_MIGRATION` procedure.

```
SQL> EXECUTE DBMS_AQMIGTOOL.COMMIT_MIGRATION(cqschema => 'aquser', cqname =>
'JSON_QUEUE');

PL/SQL procedure successfully completed.

SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues where
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> TRUE
```

Following the successful migration, the `JSON_QUEUE` has been transformed from an AQ to a TxEventQ. The user can seamlessly proceed with their everyday operations as usual on `JSON_QUEUE`.

Users can verify successful evaluation by querying in the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name =
'JSON_QUEUE' AND event = 'Commit_Migration';

EVENT
--------------------------------------------------------------------------------
Commit_Migration
```

Scenario 2

Let's consider the same example outlined in Scenario 1, continuing until the successful execution of `DBMS_AQMIGTOOL.INIT_MIGRATION`.

In certain situations, after executing a workload, it's possible that unsupported features may be detected. For example, if a user configured a transformation in enqueue operation. In such cases, the user can use the `DBMS_AQMIGTOOL.CHECK_STATUS` procedure to determine if the migration process can be successfully finalized or needs an application change to clear the error.

```
SQL> DECLARE
     2        mig_STATUS    VARCHAR2(128);
     3        mig_comments  VARCHAR2(1024);
     4    BEGIN
     5        DBMS_AQMIGTOOL.CHECK_STATUS(
     6                   cqschema                 => 'aquser',
     7                   cqname                   => 'JSON_QUEUE',
     8                   status                   => mig_STATUS,
     9                   migration_comment        => mig_comments);
    10       dbms_output.put_line('Migration Status: ' || mig_STATUS);
    11       dbms_output.put_line('Migration comments: ' || mig_comments);
    12    END;
    13    /
Migration Status: Compatibility Error: Transformation in Enq Unsupported Feature
Migration comments: Unsupported parameter in Enqueue

PL/SQL procedure successfully completed.
```

Additionally, the user has the option to examine the list of captured unsupported features by querying the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name =
'JSON_QUEUE' AND event_status = 2;

EVENT
--------------------------------------------------------------------------------
Transformation in Enq
```

If the user later decides to cancel the migration process without any message loss, they can use the `DBMS_AQMIGTOOL.RESTORE` option in the `DBMS_AQMIGTOOL.CANCEL_MIGRATION` procedure, which is set by default.

```
SQL> SELECT count(*) FROM JSON_QUEUE_TABLE WHERE q_name = 'JSON_QUEUE';

  COUNT(*)
----------
      1000

1 row selected.


SQL> EXECUTE DBMS_AQMIGTOOL.CANCEL_MIGRATION(cqschema => 'aquser', cqname =>
'JSON_QUEUE', cancelmode => DBMS_AQMIGTOOL.RESTORE);

PL/SQL procedure successfully completed.
```

```
SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues WHERE
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> FALSE


SQL> SELECT count(*) FROM JSON_QUEUE_TABLE WHERE q_name = 'JSON_QUEUE';

  COUNT(*)
----------
      4500

1 row selected.
```

As evident, the message counts within the queue table following the execution of `DBMS_AQMIGTOOL.CANCEL_MIGRATION` reflects the initial count (`1000` messages) and the messages restored from TxEventQ (`3500` messages).

Steps to migrate from AQ to TxEventQ

To migrate from AQ to TxEventQ, perform the following steps:

1. Identify if source AQ is compatible with migration.

   Start by determining whether the source AQ is suitable for migration to TxEventQ. The procedure below enables the identification of any features utilized by AQ that are not supported in TxEventQ.

   ```
   DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ(cqschema, cqname, migration_report,
   checkmode)
   ```

   The `DBMS_AQMIGTOOL.CURRENT` option for the `checkmode` parameter only examines current queue metadata and existing messages. However, future queue operations won't be examined. Furthermore, like transformations, particular features don't rely on persistent queue data, which cannot be identified without runtime enqueue/dequeue operations. To comprehensively identify all unsupported features, it's advisable to use the `DBMS_AQMIGTOOL.ENABLE_EVALUATION` option in the `checkmode` parameter and subsequently run your user application.

   The `migration_report` will generate no data if the output is compatible. Once it is confirmed that there is no incompatibility, the user can proceed with `DBMS_AQMIGTOOL.INIT_MIGRATION`. Users should not initiate migration APIs if any of the features listed under Limitations is used in their application. Most of these are not very commonly used.

2. Start migration and create an interim TxEventQ.

   To begin the migration process, start by creating an interim TxEventQ using this method:

   ```
   DBMS_AQMIGTOOL.INIT_MIGRATION(cqschema, cqname, txeventqschema, txeventqname,
   mig_mode, ordering, suffix)
   ```

   Through the execution of `DBMS_AQMIGTOOL.INIT_MIGRATION`, compatibility checks are conducted using both AQ definitions and data. If any unsupported features are detected, an exception will be raised. In such instances, users have the opportunity to modify their applications, address the identified incompatibilities (refer to the Limitations and Workarounds section for guidance), and subsequently reattempt the execution of `INIT_MIGRATION`.

If found compatible, this procedure initiates the migration process and creates an interim TxEventQ with the same configuration of the AQ, encompassing queue properties, payload types, and subscriber data. It is important to note that following the execution of this procedure, all the administrative operations on the AQ are restricted.

3. Finalize migration either by committing or canceling migration.

   Concluding the migration process involves two options: committing or canceling the migration.

   - In cases where all messages from AQ have been consumed (indicating that all messages are in a `PROCESSED` state) and no incompatibilities found, the user can execute the following to complete the migration:

     ```
     DBMS_AQMIGTOOL.COMMIT_MIGRATION(cqschema, cqname, ignore_warning)
     ```

     This procedure drops the AQ and renames the interim TxEventQ to the AQ's name. Additionally, the TxEventQ is enabled for all administrative operations, completing the migration process.

   - Suppose a user gets any compatibility error after `DBMS_AQMIGTOOL.INIT_MIGRATION`, that operation will be blocked, and the user will be asked to review compatibility issues. Unsupported features will be recorded in the `USER_TXEVENTQ_MIGRATION_STATUS` view. Users facing such issues have the option to suitably modify their application to address the incompatibility or invoke the procedure to restore the AQ.

     ```
     DBMS_AQMIGTOOL.CANCEL_MIGRATION(cqschema, cqname, cancelmode)
     ```

     Utilizing `DBMS_AQMIGTOOL.CANCEL_MIGRATION`, users can restore TxEventQ messages and their states back to AQ based on the specified `cancelmode`. This process ensures no data loss. It's important to note that executing `CANCEL_MIGRATION` may lead to extended downtime due to the movement of messages and their states from TxEventQ back to AQ.

   In summary, this final migration stage offers flexibility, allowing users to choose between commitment and restoration based on their specific needs and circumstances.

Overview of how Migration Functions

The DBMS_AQMIGTOOL package helps with the seamless migration from AQ to TxEventQ.

- To start the migration process, `DBMS_AQMIGTOOL.INIT_MIGRATION` sets up an interim TxEventQ copying the AQ's configuration, including payload type, rules, subscribers, privileges, PLSQL notifications, and more. It restricts AQ from administrative changes to maintain TxEventQ configuration integrity until the migration is completed or canceled.
- During migration, the new workload gradually transitions to interim TxEventQ. New enqueue requests go to interim TxEventQ; dequeue requests check AQ first, then interim TxEventQ, progressively emptying AQ. The message ordering is maintained as per `INIT_MIGRATION`'s `ordering` configuration.
- To complete migration, `DBMS_AQMIGTOOL.COMMIT_MIGRATION` drops AQ and renames interim TxEventQ to AQ's name, ensuring application compatibility.
- To cancel migration, `DBMS_AQMIGTOOL.CANCEL_MIGRATION` in `RESTORE` mode moves messages and their states from interim TxEventQ to AQ to prevent data loss, then drops interim TxEventQ.
- For capturing unsupported features, `DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ` examines AQ's metadata and messages to capture unsupported features. The

`DBMS_AQMIGTOOL.ENABLE_EVALUATION` option captures runtime-specific unsupported features.

Limitations and Workarounds

Before starting the migration process, it is recommended that the user review the list of the following unsupported features.

Table 8-1 Unsupported Features and Workarounds

Name	Feature Type	Description	Workaround
retry delay See Also: MESSAGE_PROPERTIES_T.	Queue-level	Specifies the delay in seconds before a message is available for dequeue after an application rollback	Set `retry_delay` to zero using `DBMS_AQADM.ALTER_QUEUE`.
transformation	Message-level	Allows message transformation before enqueueing or after the subscriber dequeue the message	Move the transformation to the application layer.
listen See Also: Listen Procedures.	others	Listens on one or more queues for a list of agents/subscribers	Implement single queue listening with dequeue browse.
invalid priority value (valid range - `0` to `9`)	Message-level	TxEventQ only allows priority values between `0` to `9`. The default priority value for AQ is `1`, and for TxEventQ is `4`. Also, AQ interprets the lower value as the higher priority, But TxEventQ interprets the opposite, that is, the higher value is interpreted as the higher priority.	Ensure the application adheres to a valid range (`0` - `9`). Also, keep in mind the difference in interpretation of the priority value of AQ and TxEventQ.

If propagation is already scheduled on AQ before migration, it will result in an incompatibility error. To address this, users can unschedule the propagation using `DBMS_AQADM.UNSCHEDULE_PROPAGATION`. After migration, the propagation can be scheduled on TxEventQ using `DBMS_AQADM.SCHEDULE_PROPAGATION`.

Table 8-2 Unsupported Features without Workarounds

Name	Feature Type
message grouping (transactional grouping) See Also: • CREATE_QUEUE_TABLE Procedure • MESSAGE_PROPERTIES_T Type	Queue-level
sequence deviation and relative msgid See Also: ENQUEUE_OPTIONS_T Type	Message-level
recipient list See Also: MESSAGE_PROPERTIES_T Type	Message-level

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_AQMIGTOOL`

Monitoring TxEventQ with Prometheus/Grafana

Some of the advantages of having a real-time monitoring framework for a high throughput messaging system are as follows.

- Know overall messaging system health at a glance and be able to adjust resources up or down with how heavy or light the messaging work load is.
- Monitor high level key performance indicators: enqueue rates, dequeue rates, queue depth, etc.
- Find the messaging bottlenecks due to the database load or the system load, by monitoring CPU load, memory utilization, and the database wait class from messaging activity.
- Check the health condition of each queue to quickly identify under-performing ones easily.
- Access messaging metrics from anywhere, enabling developers to monitor any overheads from applications and debug message related issues.
- Respond quickly by setting alerts when something goes wrong with the feature in Grafana.

See Also:

Monitoring Transactional Event Queues for more information.

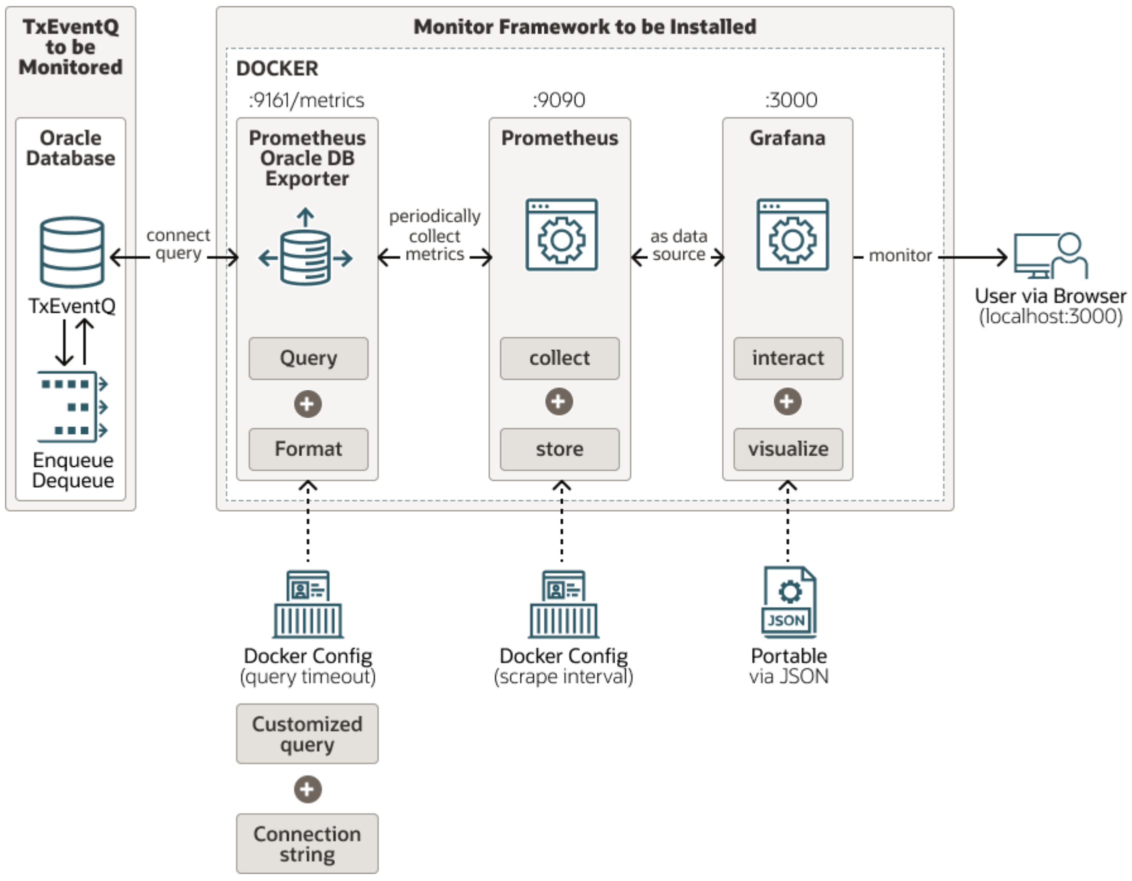
Monitoring Data Flow and UI Framework Setup

The TxEventQ monitor system consists of three independent open-source components. A Docker container is used to help manage all environments, services, and dependencies on the machine where the monitoring framework is installed.

- Oracle DB Exporter: A Prometheus exporter for Oracle Database, which connects to the database, queries metrics, and formats metrics into Prometheus-like metrics.
- Prometheus: A monitor system and time-series database, which manages metrics collecting from Oracle DB Exporter in time-series-fashion.
- Grafana: An analytics and interactive visualization platform, which specifies Prometheus as data source.

TxEventQ Monitor System consists of three services including Prometheus Oracle DB Exporter, Prometheus, and Grafana. The system is designed to run with Docker, which lets user use the system as a lightweight, portable, self-sufficient container, which can run virtually anywhere. Exporter is the connector to Oracle DB and formats the query results to Prometheus-like metrics. Prometheus is a time-series database and periodically controls Exporter to query and collect/store metrics. Grafana uses Prometheus as a data source to show the metrics and visually. Grafana is a user-interface with charting and computation built-in. The whole services is configured, managed and handled by Docker-compose.

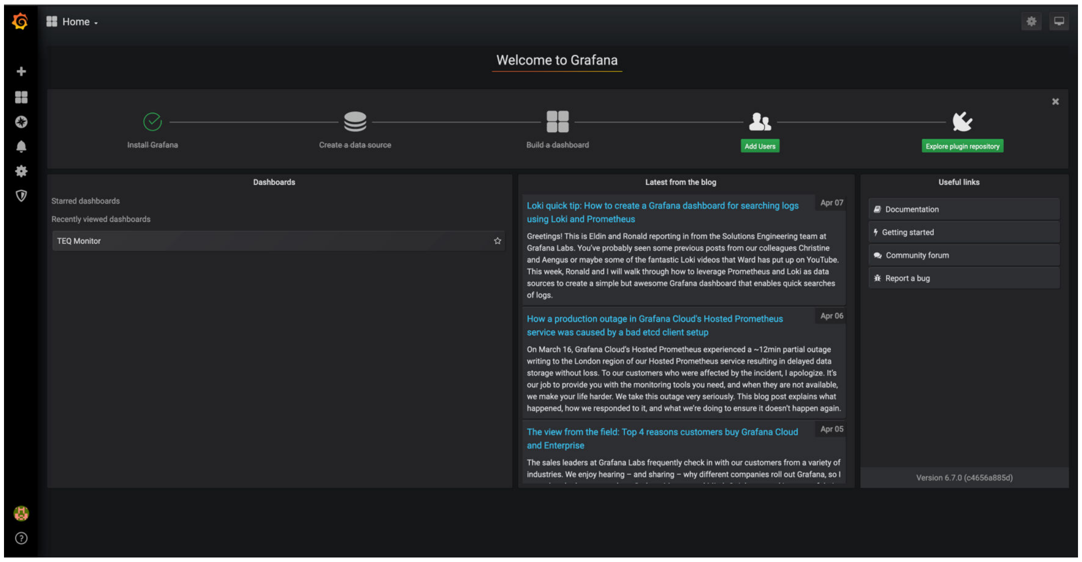
Figure 8-2 Monitoring Transaction Event Queue



To monitor the TxEventQ dashboards using Grafana, perform the following steps.

1. Login to the Grafana dashboard using admin user name and password. The **Welcome Page** is displayed.

Figure 8-3 Welcome Page



Dashboards

Starred dashboards

Recently viewed dashboards

TEQ Monitor

Latest from the blog

Loki quick tip: How to create a Grafana dashboard for searching logs using Loki and Prometheus Apr 07

Greetings! This is Eldin and Ronald reporting in from the Solutions Engineering team at Grafana Labs. You've probably seen some previous posts from our colleagues Christine and Aengus or maybe some of the fantastic Loki videos that Ward has put up on YouTube. This week, Ronald and I will walk through how to leverage Prometheus and Loki as data sources to create a simple but awesome Grafana dashboard that enables quick searches of logs.

How a production outage in Grafana Cloud's Hosted Prometheus service was caused by a bad etcd client setup Apr 06

On March 16, Grafana Cloud's Hosted Prometheus experienced a ~12min partial outage writing to the London region of our Hosted Prometheus service resulting in delayed data storage without loss. To our customers who were affected by the incident, I apologize. It's our job to provide you with the monitoring tools you need, and when they are not available, we make your life harder. We take this outage very seriously. This blog post explains what happened, how we responded to it, and what we're doing to ensure it doesn't happen again.

The view from the field: Top 4 reasons customers buy Grafana Cloud and Enterprise Apr 05

The sales leaders at Grafana Labs frequently check in with our customers from a variety of industries. We enjoy hearing – and sharing – why different companies roll out Grafana, so I

Useful links

Documentation

Getting started

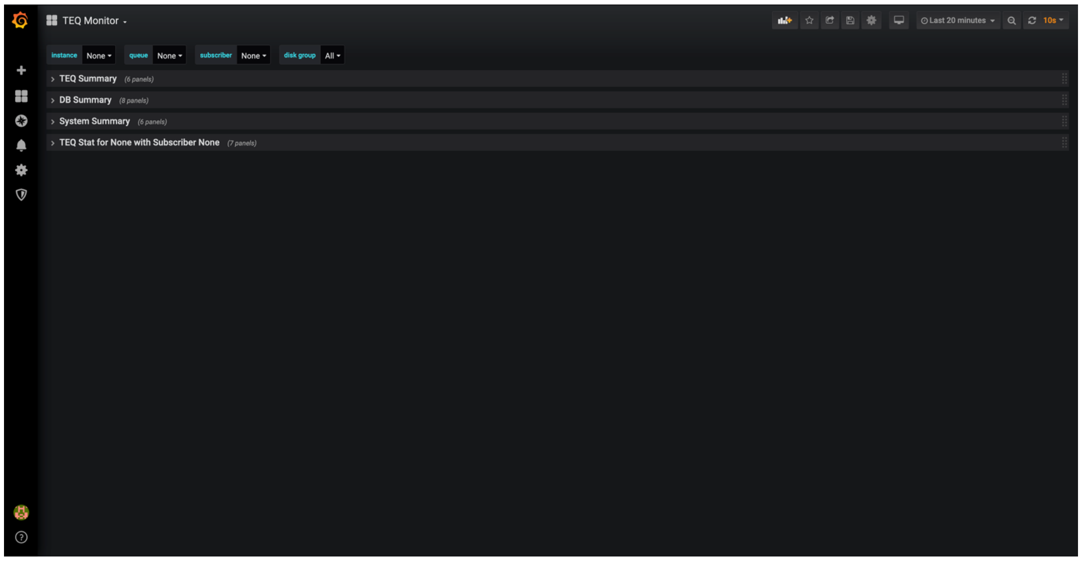
Community forum

Report a bug

Version 6.7.0 (c4656a885d)

2. Click **TxEventQ Monitor** on the Welcome Page. Once Grafana is setup, the metrics are presented in four selections, and the top level selections are for an instance, queue, subscriber and disk group. The four selections are as follows:

   - Summary across all TxEventQs
   - Database metrics summary
   - System metrics summary
   - Subscriber summary for each TxEventQ



instance None queue None subscriber None disk group All

TEQ Summary (6 panels)

DB Summary (8 panels)

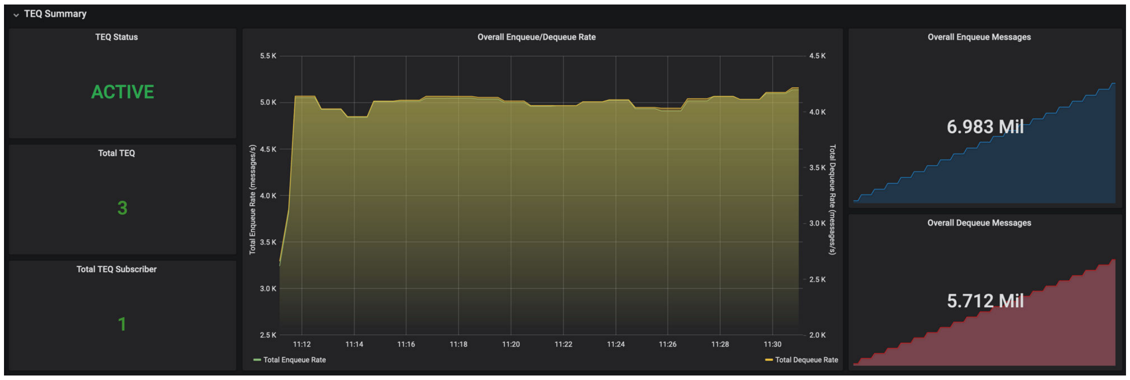
System Summary (6 panels)

TEQ Stat for None with Subscriber None (7 panels)

3. Click on each summary to view information about the summary.

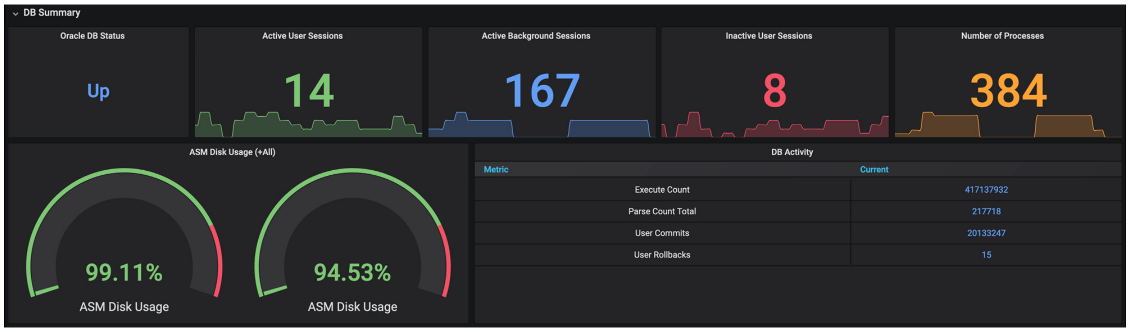
The following figures shows the dashboards of TxEventQ Summary, DB Summary, Database Wait Class Latency, and System Summary respectively.

The TxEventQ Summary dashboard shows overall aggregated TxEventQ stats including status, number of queues, number of subscribers, enqueue/dequeue rate and number of messages



The Database Summary dashboard shows overall DB performance and stats.

Figure 8-4   Database Summary



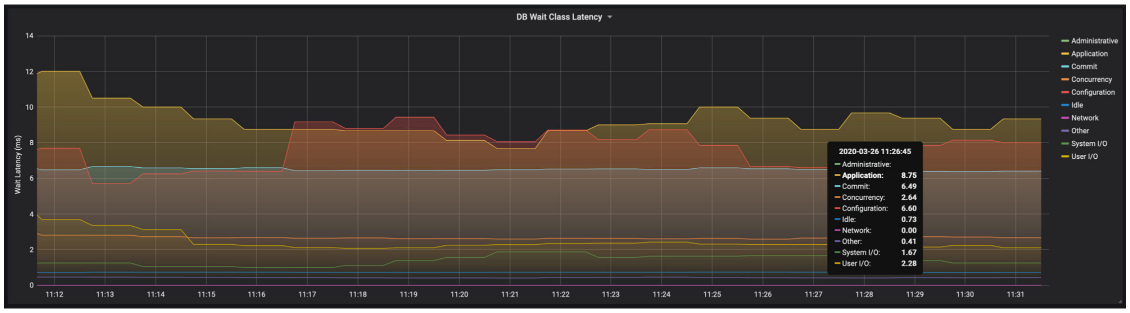
Metric	Current
Execute Count	417137932
Parse Count Total	217718
User Commits	20133247
User Rollbacks	15

The screen tiles are as follows.

- Oracle DB Status – Up or Down
- Active User Sessions – number of user sessions active
- Active Background sessions – number of background sessions active
- Inactive user sessions – Number of inactive user sessions
- Number of processes – Number of database processes
- ASM Disk Usage – Percent of disk free for each disk volume
- DB Activity – SQL activity for the number of execute counts, parse count total, user commits, user rollbacks.

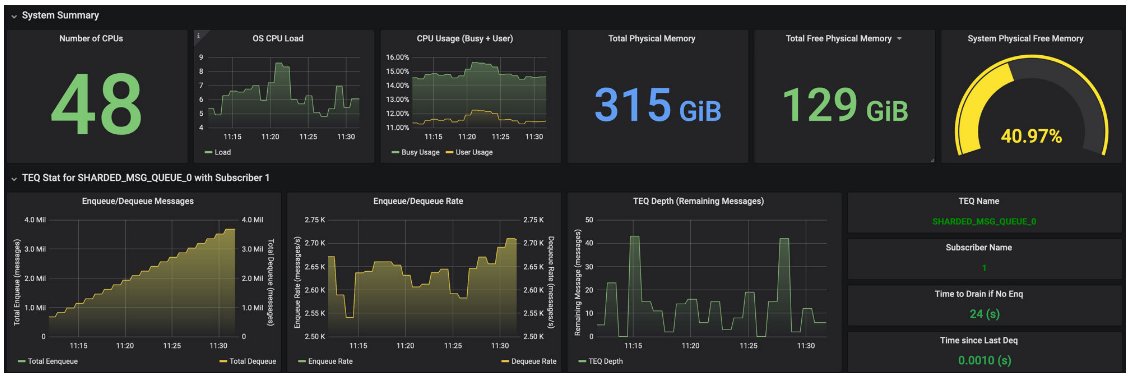
The database wait class latencies are shown in the DB Wait Class Latency dashboard. Wait class latency is the wait class events latency in milliseconds in the database and can be used to guide overhead analysis through a more detailed AWR report analysis.

Figure 8-5 Database Wait Class Latency



The System Summary dashboard shows system level metrics and also the queue level metrics. It reflects the overall performance and status of the system running Oracle DB based on CPU utilization and memory used.

Figure 8-6 System Summary



System Level Statistics

- Number of CPUs – Total number of CPUs on the system
- OS CPU Load - The percentage of CPU capability currently used by all System and User processes
- CPU Usage: % of CPU busy (for all processes) and % of CPU busy for user processes
- Total Physical Memory: Total memory on the system, one instance in case of an Oracle RAC database
- Total Free Physical Memory: Total amount of free memory on the instance
- System Physical Memory free: % of free physical memory

TxEventQ Queue Level Stats

It displays the statistics of one specific queue, which the user can select from the drop-down menu including rate, total messages, queue depth, estimated time to consume and time since last dequeue.

- Enqueue/Dequeue Messages: Number of messages enqueued; number of messages dequeued

- Enqueue/Dequeue rate: Number of messages per second that are enqueued and dequeued
- TxEventQ Depth – Remaining messages in the queue
- TxEventQ Name - Name of the queue
- Subscriber Name – Name of the subscriber
- Time to drain if no enq – Estimate of time to drain the queue if there are no new enqueues
- Time since last dequeue – Time elapsed since the last dequeue on the queue

Key Metrics Measured

This section provides a little more detail on the metrics seen in the previous section and how to get these from the Grafana screen. The drop-down menu options are at the level of a: database instance, queue, and a subscriber. AQ/TxEventQ Summary metrics and Database metrics are for the database instance the user selects in the drop-down menu.

- AQ/TxEventQ Summary Metrics
  - TxEventQ Status: if TxEventQs are running or not
  - Total Number of TxEventQs: the number of TxEventQs running
  - Total TxEventQ Subscribers: the total number of subscribers for all TxEventQs
  - Overall Enq/Deq Rates: aggregate enq/deq rates for all TxEventQs
  - Overall Enqueued Messages: total enqueued messages for the entire queue system
  - Overall Dequeued Messages: total dequeued messages for the entire queue system
- Database Summary Metrics
  - Oracle DB Status: if Oracle DB is running or not.
  - Active User Sessions: the number of active user sessions
  - Active Background Sessions: the number of active background sessions
  - Inactive User Sessions: the number of inactive user sessions
  - Number of Processes: the number of Oracle processes running
  - ASM Disk Usage: Oracle Automatic Storage Management disk group memory usage (e.g. +DATA, +RECO)
  - DB Activity: the number of specific DB operations that occurred including execute count, user commits, parse count total, user rollbacks.
  - DB Wait Class Latency: average latency for DB wait class in ms including administrative, application, commit, concurrency, configuration, idle, network, other, system I/O, user I/O
- System Summary Metrics
  - Number of CPUs: the number of CPU of the system running Oracle DB
  - OS CPU Load: current number of processes that are either running or in the ready state, waiting to be selected by the operating-system scheduler to run. On many platforms, this statistic reflects the average load over the past minute
  - CPU Usage (Busy + User): the CPU usage in percentage in real time including CPU in busy state or CPU in executing user code.

  - Total Physical Memory: total physical memory of the system.
  - Total Free Physical Memory: total free physical memory of the system.
  - System Free Physical Memory: the percentage of free memory in the system.
- Queue Level Metrics
  - Enq/Deq Messages: total messages enqueued/dequeued to/from the TxEventQ
  - Enq/Deq Rate: enqueue/dequeue rate for the TxEventQ
  - TxEventQ Depth: total messages remaining in the queue.
  - TxEventQ Name: the name of TxEventQ
  - Subscriber Name: the name of TxEventQ subscriber
  - Time to Drain if No Enq: total amount of time to consume all messages if there are no more messages enqueued on the TxEventQ
  - Time since Last Deq: time difference between current time and the time since the last dequeue operation on the TxEventQ

See Also:

Monitoring Transactional Event Queues for more information.

9 Oracle Transactional Event Queue and Advanced Queuing Views

These topics describe the Transactional Event Queue (TxEventQ) and AQ Advanced Queuing (AQ) administrative interface views.

Note:

All views not detailed in the following sections are described in the *Oracle Database Reference*.

Oracle TxEventQ Views

- V$EQ_CACHED_PARTITIONS
- V$EQ_CROSS_INSTANCE_JOBS
- V$EQ_DEQUEUE_SESSIONS
- V$EQ_INACTIVE_PARTITIONS
- V$EQ_MESSAGE_CACHE
- V$EQ_MESSAGE_CACHE_ADVICE
- V$EQ_MESSAGE_CACHE_STAT
- V$EQ_NONDUR_SUBSCRIBER
- V$EQ_NONDUR_SUBSCRIBER_LWM
- V$EQ_PARTITION_STATS
- V$EQ_REMOTE_DEQUEUE_AFFINITY
- V$EQ_SUBSCRIBER_LOAD
- V$EQ_SUBSCRIBER_STAT
- V$EQ_UNCACHED_PARTITIONS

Oracle AQ Views

- V$AQ_MESSAGE_CACHE_STAT: Memory Management for Sharded Queues
- V$AQ_SHARDED_SUBSCRIBER_STAT: Sharded Queue Subscriber Statistics
- V$AQ_MESSAGE_CACHE_ADVICE: Simulated Metrics
- V$AQ_REMOTE_DEQUEUE_AFFINITY: Dequeue Affinity Instance List
- DBA_QUEUE_TABLES: All Queue Tables in Database
- USER_QUEUE_TABLES: Queue Tables in User Schema
- ALL_QUEUE_TABLES: Queue Tables Accessible to the Current User
- DBA_QUEUES: All Queues in Database

- USER_QUEUES: Queues In User Schema
- ALL_QUEUES: Queues for Which User Has Any Privilege
- DBA_QUEUE_SCHEDULES: All Propagation Schedules
- USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema
- QUEUE_PRIVILEGES: Queues for Which User Has Queue Privilege
- AQ$<Queue_Table_Name>: Messages in Queue Table
- AQ$<Queue_Table_Name_S>: Queue Subscribers
- AQ$<Queue_Table_Name_R>: Queue Subscribers and Their Rules
- DBA_QUEUE_SUBSCRIBERS: All Queue Subscribers in Database
- USER_QUEUE_SUBSCRIBERS: Queue Subscribers in User Schema
- ALL_QUEUE_SUBSCRIBERS: Subscribers for Queues Where User Has Queue Privileges
- DBA_TRANSFORMATIONS: All Transformations
- DBA_ATTRIBUTE_TRANSFORMATIONS: All Transformation Functions
- USER_TRANSFORMATIONS: User Transformations
- USER_ATTRIBUTE_TRANSFORMATIONS: User Transformation Functions
- DBA_SUBSCR_REGISTRATIONS: All Subscription Registrations
- USER_SUBSCR_REGISTRATIONS: User Subscription Registrations
- AQ$INTERNET_USERS: Oracle Database Advanced Queuing Agents Registered for Internet Access
- V$AQ: Number of Messages in Different States in Database
- V$BUFFERED_QUEUES: All Buffered Queues in the Instance.
- V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance
- V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance
- V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance
- V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance
- V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance
- V$PROPAGATION_SENDER: Buffer Queue Propagation Schedules on the Sending (Source) Side
- V$PROPAGATION_RECEIVER: Buffer Queue Propagation Schedules on the Receiving (Destination) Side
- V$SUBSCR_REGISTRATION_STATS: Diagnosability of Notifications
- V$METRICGROUP: Information About the Metric Group
- V$AQ_BACKGROUND_COORDINATOR: Performance Statistics for AQ's Master Background Coordinator Process (AQPC)
- V$AQ_JOB_COORDINATOR: Performance Statistics per Coordinator
- V$AQ_SERVER_POOL: Performance Statistics for all Servers
- V$AQ_CROSS_INSTANCE_JOBS: Cross Process Jobs Description
- V$AQ_IPC_ACTIVE_MSGS

- V$AQ_IPC_MSG_STATS
- V$AQ_IPC_PENDING_MSGS
- V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations
- V$AQ_NOTIFICATION_CLIENTS: Secure OCI Client Connections
- V$AQ_SUBSCRIBER_LOAD: Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER: Non-Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER_LWM: LWM of Non Durable Subscriber
- V$AQ_MESSAGE_CACHE: Performance Statistics

DBA_QUEUE_TABLES: All Queue Tables in Database

This view contains information about the owner instance for a queue table.

A queue table can contain multiple queues. In this case, each queue in a queue table has the same owner instance as the queue table. The `DBA_QUEUE_TABLES` columns are the same as those in `ALL_QUEUE_TABLES`.

See Also:

Oracle Database Reference for more information about `DBA_QUEUE_TABLES`.

USER_QUEUE_TABLES: Queue Tables in User Schema

This view is the same as `DBA_QUEUE_TABLES` with the exception that it only shows queue tables in the user's schema.

`USER_QUEUE_TABLES` does not contain a column for `OWNER`.

See Also:

Oracle Database Reference for more information about `USER_QUEUE_TABLES`.

ALL_QUEUE_TABLES: Queue Tables Accessible to the Current User

This view describes queue tables accessible to the current user.

See Also:

Oracle Database Reference for more information about `ALL_QUEUE_TABLES`.

DBA_QUEUES: All Queues in Database

The `DBA_QUEUES` view specifies operational characteristics for every queue in a database.

Its columns are the same as those `ALL_QUEUES`. Oracle Database 12*c* Release 1 (12.1) introduces a new column `SHARDED` with data type `VARCHAR2(5)`. The value for this column is `TRUE` for sharded queue, otherwise `FALSE`.

See Also:

Oracle Database Reference for more information about `DBA_QUEUES`.

USER_QUEUES: Queues In User Schema

The `USER_QUEUES` view is the same as `DBA_QUEUES` with the exception that it only shows queues in the user's schema.

Oracle Database 12*c* Release 1 (12.1) introduces a new column `SHARDED` with data type `VARCHAR2(5)`. The value for this column is `TRUE` for sharded queue, otherwise `FALSE`.

See Also:

Oracle Database Reference for more information about `USER_QUEUES`.

ALL_QUEUES: Queues for Which User Has Any Privilege

The `ALL_QUEUES` view describes all queues on which the current user has enqueue or dequeue privileges.

If the user has any Advanced Queuing system privileges, like `MANAGE ANY QUEUE`, `ENQUEUE ANY QUEUE` or `DEQUEUE ANY QUEUE`, this view describes all queues in the database. Oracle Database 12*c* Release 1 (12.1) introduces a new column `SHARDED` with data type `VARCHAR2(5)`. The value for this column is `TRUE` for sharded queue, otherwise `FALSE`.

See Also:

Oracle Database Reference for more information about `ALL_QUEUES`.

DBA_QUEUE_SCHEDULES: All Propagation Schedules

The `DBA_QUEUE_SCHEDULES` view describes all the current schedules in the database for propagating messages.

See Also:

Oracle Database Reference for more information about `DBA_QUEUE_SCHEDULES`.

USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema

The `USER_QUEUE_SCHEDULES` view is the same as `DBA_QUEUE_SCHEDULES` with the exception that it only shows queue schedules in the user's schema.

See Also:

Oracle Database Reference for more information about `USER_QUEUE_SCHEDULES`.

QUEUE_PRIVILEGES: Queues for Which User Has Queue Privilege

The `QUEUE_PRIVILEGES` view describes queues for which the user is the grantor, grantee, or owner.

It also shows queues for which an enabled role on the queue is granted to `PUBLIC`.

See Also:

Oracle Database Reference for more information about `QUEUE_PRIVILEGES`.

AQ$<*Queue_Table_Name*>: Messages in Queue Table

The `AQ$<`*`Queue_Table_Name`*`>` view describes the queue table in which message data is stored.

This view is automatically created with each queue table and should be used for querying the queue data. The dequeue history data (time, user identification and transaction identification) is only valid for single-consumer queues.

In a queue table that is created with the `compatible` parameter set to '8.1' or higher, messages that were not dequeued by the consumer are shown as "`UNDELIVERABLE`". You can dequeue these messages by `msgid`. If the Oracle Database Advanced Queuing queue process monitor is running, then the messages are eventually moved to an exception queue. You can dequeue these messages from the exception queue with an ordinary dequeue.

A multiconsumer queue table created without the `compatible` parameter, or with the `compatible` parameter set to '8.0', does not display the state of a message on a consumer basis, but only displays the global state of the message.

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

When a message is dequeued using the `REMOVE` mode, `DEQ_TIME`, `DEQ_USER_ID`, and `DEQ_TXN_ID` are updated for the consumer that dequeued the message.

You can use `MSGID` and `ORIGINAL_MSGID` to chain propagated messages. When a message with message identifier `m1` is propagated to a remote queue, `m1` is stored in the `ORIGINAL_MSGID` column of the remote queue.

Beginning with Oracle Database 10*g*, `AQ$`*`Queue_Table_Name`* includes buffered messages. For buffered messages, the value of `MSG_STATE` is one of the following:

- `IN MEMORY`

  Buffered messages enqueued by a user
- `DEFERRED`

  Buffered messages enqueued by a capture process
- `SPILLED`

  User-enqueued buffered messages that have been spilled to disk
- `DEFERRED SPILLED`

  Capture-enqueued buffered messages that have been spilled to disk
- `BUFFERED EXPIRED`

  Expired buffered messages

For JMS Sharded Queues, the columns `RETRY_COUNT`, `EXCEPTION_QUEUE_OWNER`, `EXCEPTION_QUEUE`, `PROPAGATED_MSGID`, `SENDER_NAME`, `SENDER_ADDRESS`, `SENDER_PROTOCOL`, `ORIGINAL_MSGID`, `ORIGINAL_QUEUE_NAME`, `ORIGINAL_QUEUE_OWNER`, `EXPIRATION_REASON` are always `NULL`.

For JMS Sharded Queues, this view shows messages only for durable subscribers because non durable subscribers are session specific. The view returns data from the in-memory Sharded Queue message cache if available, otherwise from the values on disk. A user is required to be one of the following in order to query from `AQ$<`*`queue_name`*`>` view for Sharded Queues:

- user is the owner
- user has "dequeue" privilege on queue
- user has "dequeue any queue" privilege

The view has the following difference for Sharded Queues for 12*c* and future releases:

- `MSG_PRIORITY` is defined as `NUMBER(38)`
- `MSG_STATE` in a queue table does not have `BUFFERED_EXPIRED` hence the max length of `UNDELIVERABLE` is taken as length got `MSG_STATE`.
- `EXPIRATION` is defined as `TIMESTAMP(6) WITH TIME ZONE` in a queue table.
- `USER_DATA` column is defined using a decode on `USERDATA_RAW` and `USERDATA_BLOB` with `UTL_RAW.CAST_TO_VARCHAR2`.
- `CONSUMER_NAME` is defined as `VARCHAR2(128)`

Table 9-1 AQ$<Queue_Table_Name> View

Column	Datatype	NULL	For JMS Sharded Queues 12c Release 1 (12.1)	Description
`QUEUE`	`VARCHAR2(30)`	-		Queue name
`SHARD_ID`	`NUMBER`	-		`N/A` for 11*g*
`SUBSHARD_ID`	`NUMBER`	-		`N/A` for 11*g*
`MSG_ID`	`RAW(16)`	`NOT NULL`		Unique identifier of the message
`CORR_ID`	`VARCHAR2(128)`	-		User-provided correlation identifier
`MSG_PRIORITY`	`NUMBER`	-	`NUMBER(38)`	Message priority
`MSG_STATE`	`VARCHAR2(16)`	-		Message state. 12*c* Release 1 (12.1) queue table doesnt have `BUFFERED_EXPIRED`. Hence for 12c Release 1 (12.1) the max length of `UNDELIVERABLE` is taken as length got `MSG_STATE`
`DELAY`	`DATE`	-		Time in date format at which the message in waiting state would become ready. Equals `ENQUEUE_TIME` + user specified `DELAY`
`DELAY_TIMESTAMP`	`TIMESTAMP`	-		Time as a timestamp format at which the message in waiting state would become ready. Equals `ENQUEUE_TIMESTAMP` + user specified `DELAY`
`EXPIRATION`	`NUMBER`	-	`TIMESTAMP(6) WITH TIME ZONE`	Number of seconds in which the message expires after being `READY`
`RETENTION_TIMESTAMP`	`TIMESTAMP(6)`	-		`N/A` for 11*g*
`ENQ_TIME`	`DATE`	-		Enqueue time
`ENQ_TIMESTAMP`	`TIMESTAMP`	-		Enqueue time
`ENQ_USER_ID`	`NUMBER`	-		Enqueue user ID
`ENQ_USER_ID` (10.1 queue tables)	`VARCHAR2(30)`	-		Enqueue user name
`ENQ_TXN_ID`	`VARCHAR2(30)`	-		Enqueue transaction ID

Table 9-1 (Cont.) AQ$<Queue_Table_Name> View

Column	Datatype	NULL	For JMS Sharded Queues 12c Release 1 (12.1)	Description
DEQ_TIME	DATE	-		Dequeue time
DEQ_TIMESTAMP	TIMESTAMP	-		Dequeue time
DEQ_USER_ID	NUMBER	-		Dequeue user ID
DEQ_USER_ID (10.1 queue tables)	VARCHAR2(30)	-		Dequeue user name
DEQ_TXN_ID	VARCHAR2(30)	-		Dequeue transaction ID
RETRY_COUNT	NUMBER	-	NULL	Number of retries
EXCEPTION_QUEUE_OWNER	VARCHAR2(30)	-	NULL	Exception queue schema
EXCEPTION_QUEUE	VARCHAR2(30)	-	NULL	Exception queue name
USER_DATA	-	-		User data. USER_DATA column is defined using a decode on USERDATA_RAW and USERDATA_BLOB with UTL_RAW.CAST_TO_VARCHAR2 for 12c Release 1 (12.1).
SENDER_NAME	VARCHAR2(30)	-	NULL	Name of the agent enqueuing the message (valid only for 8.1-compatible queue tables)
SENDER_ADDRESS	VARCHAR2(1024)	-	NULL	Queue name and database name of the source (last propagating) queue (valid only for 8.1-compatible queue tables). The database name is not specified if the source queue is in the local database.
SENDER_PROTOCOL	NUMBER	-	NULL	Protocol for sender address (reserved for future use and valid only for 8.1-compatible queue tables)
ORIGINAL_MSGID	RAW(16)	-	NULL	Message ID of the message in the source queue (valid only for 8.1-compatible queue tables)
CONSUMER_NAME	VARCHAR2(30)	-	VARCHAR2(128)	Name of the agent receiving the message (valid only for 8.1-compatible multiconsumer queue tables)
ADDRESS	VARCHAR2(1024)	-		Queue name and database link name of the agent receiving the message.The database link name is not specified if the address is in the local database. The address is NULL if the receiving agent is local to the queue (valid only for 8.1-compatible multiconsumer queue tables)

Table 9-1 (Cont.) AQ$<Queue_Table_Name> View

Column	Datatype	NULL	For JMS Sharded Queues 12c Release 1 (12.1)	Description
`PROTOCOL`	`NUMBER`	-		Protocol for address of receiving agent (valid only for 8.1-compatible queue tables)
`PROPAGATED_MSGID`	`RAW(16)`	-	`NULL`	Message ID of the message in the queue of the receiving agent (valid only for 8.1-compatible queue tables)
`ORIGINAL_QUEUE_NAME`	`VARCHAR2(30)`	-	`NULL`	Name of the queue the message came from
`ORIGINAL_QUEUE_OWNER`	`VARCHAR2(30)`	-	`NULL`	Owner of the queue the message came from
`EXPIRATION_REASON`	`VARCHAR2(19)`	-	`NULL`	Reason the message came into exception queue. Possible values are `TIME_EXPIRATION` (message expired after the specified expired time), `MAX_RETRY_EXCEEDED` (maximum retry count exceeded), and `PROPAGATION_FAILURE` (message became undeliverable during propagation).

Note:

A message is moved to an exception queue if `RETRY_COUNT` is greater than `MAX_RETRIES`. If a dequeue transaction fails because the server process dies (including `ALTER SYSTEM KILL SESSION`) or `SHUTDOWN ABORT` on the instance, then `RETRY_COUNT` is not incremented.

AQ$<Queue_Table_Name_S>: Queue Subscribers

The `AQ$<Queue_Table_Name_S>` view provides information about subscribers for all the queues in any given queue table.

It shows subscribers created by users with `DBMS_AQADM.ADD_SUBSCRIBER` and subscribers created for the apply process to apply user-created events. It also displays the transformation for the subscriber, if it was created with one. It is generated when the queue table is created.

This view provides functionality that is equivalent to the `DBMS_AQADM.QUEUE_SUBSCRIBERS()` procedure. For these queues, Oracle recommends that the view be used instead of this procedure to view queue subscribers. This view is created only for 8.1-compatible queue tables.

Table 9-2 AQ$<Queue_Table_Name_S> View

Column	Datatype	NULL	Description
`QUEUE`	`VARCHAR2(30)`	`NOT NULL`	Name of queue for which subscriber is defined
`NAME`	`VARCHAR2(30)`	-	Name of agent
`ADDRESS`	`VARCHAR2(1024)`	-	Address of agent
`PROTOCOL`	`NUMBER`	-	Protocol of agent
`TRANSFORMATION`	`VARCHAR2(61)`	-	Name of the transformation (can be null)

AQ$<Queue_Table_Name_R>: Queue Subscribers and Their Rules

The `AQ$<`*`Queue_Table_Name`*`_R>` view displays only the subscribers based on rules for all queues in a given queue table, including the text of the rule defined by each subscriber.

It also displays the transformation for the subscriber, if one was specified. It is generated when the queue table is created.

This view is created only for 8.1-compatible queue tables.

Table 9-3 AQ$<Queue_Table_Name_R> View

Column	Datatype	NULL	Description
`QUEUE`	`VARCHAR2(30)`	`NOT NULL`	Name of queue for which subscriber is defined
`NAME`	`VARCHAR2(30)`	-	Name of agent
`ADDRESS`	`VARCHAR2(1024)`	-	Address of agent
`PROTOCOL`	`NUMBER`	-	Protocol of agent
`RULE`	`CLOB`	-	Text of defined rule
`RULE_SET`	`VARCHAR2(65)`	-	Set of rules
`TRANSFORMATION`	`VARCHAR2(61)`	-	Name of the transformation (can be null)

AQ$Queue_Name_R: Queue Subscribers and Their Rules for Multi-consumer Queue

This table shows queue subscribers and their rules for multi-consumer queue.

Table 9-4 AQ$Queue_Name_R: Queue Subscribers and Their Rules for Multi-consumer Queue

Column	Datatype	NULL
`QUEUE`	`VARCHAR2(30)`	`NOT NULL`
`NAME`	`VARCHAR2(30)`	-

Table 9-4 (Cont.) AQ$Queue_Name_R: Queue Subscribers and Their Rules for Multi-consumer Queue

Column	Datatype	NULL
ADDRESS	VARCHAR2(1024)	-
PROTOCOL	NUMBER	-
RULE	CLOB	-
RULE_SET	VARCHAR2(65)	-
TRANSFORMATION	VARCHAR2(65)	-

AQ$Queue_Name_S: Queue Subscribers and Their Rules for Multi-consumer Queue

This table shows queue subscribers and their rules for multi-consumer queue.

Table 9-5 AQ$Queue_Name_S: Queue Subscribers and Their Rules for Multi-consumer Queue

Column	Datatype	NULL
QUEUE	VARCHAR2(30)	NOT NULL
NAME	VARCHAR2(30)	-
ADDRESS	VARCHAR2(1024)	-
PROTOCOL	NUMBER	-
TRANSFORMATION	VARCHAR2(65)	-
QUEUE_TO_QUEUE	VARCHAR2(5)	-

DBA_QUEUE_SUBSCRIBERS: All Queue Subscribers in Database

The `DBA_QUEUE_SUBSCRIBERS` view returns a list of all subscribers on all queues in the database.

Its columns are the same as those in `ALL_QUEUE_SUBSCRIBERS`.

See Also:

Oracle Database Reference for more information about `DBA_QUEUE_SUBSCRIBERS`.

USER_QUEUE_SUBSCRIBERS: Queue Subscribers in User Schema

The `USER_QUEUE_SUBSCRIBERS` view returns a list of subscribers on queues in the schema of the current user.

Its columns are the same as those in `ALL_QUEUE_SUBSCRIBERS` except that it does not contain the `OWNER` column.

See Also:

Oracle Database Reference for more information about `USER_QUEUE_SUBSCRIBERS`.

ALL_QUEUE_SUBSCRIBERS: Subscribers for Queues Where User Has Queue Privileges

The `ALL_QUEUE_SUBSCRIBERS` view returns a list of subscribers to queues that the current user has privileges to dequeue from.

See Also:

Oracle Database Reference for more information about `ALL_QUEUE_SUBSCRIBERS`.

DBA_TRANSFORMATIONS: All Transformations

The `DBA_TRANSFORMATIONS` view displays all the transformations in the database.

These transformations can be specified with Advanced Queue operations like enqueue, dequeue and subscribe to automatically integrate transformations in messaging. This view is accessible only to users having DBA privileges.

See Also:

Oracle Database Reference for more information about `DBA_TRANSFORMATIONS`.

DBA_ATTRIBUTE_TRANSFORMATIONS: All Transformation Functions

The `DBA_ATTRIBUTE_TRANSFORMATIONS` view displays the transformation functions for all the transformations in the database.

See Also:

Oracle Database Reference for more information about `DBA_ATTRIBUTE_TRANSFORMATIONS`.

USER_TRANSFORMATIONS: User Transformations

The `USER_TRANSFORMATIONS` view displays all the transformations owned by the user.

To view the transformation definition, query `USER_ATTRIBUTE_TRANSFORMATIONS`.

See Also:

Oracle Database Reference for more information about `USER_TRANSFORMATIONS`.

USER_ATTRIBUTE_TRANSFORMATIONS: User Transformation Functions

The `USER_ATTRIBUTE_TRANSFORMATIONS` view displays the transformation functions for all the transformations of the user.

See Also:

Oracle Database Reference for more information about `USER_ATTRIBUTE_TRANSFORMATIONS`.

DBA_SUBSCR_REGISTRATIONS: All Subscription Registrations

The `DBA_SUBSCR_REGISTRATIONS` view lists all the subscription registrations in the database.

See Also:

Oracle Database Reference for more information about `DBA_SUBSCR_REGISTRATIONS`.

USER_SUBSCR_REGISTRATIONS: User Subscription Registrations

The `USER_SUBSCR_REGISTRATIONS` view lists the subscription registrations in the database for the current user.

Its columns are the same as those in `DBA_SUBSCR_REGISTRATIONS`.

See Also:

Oracle Database Reference for more information about `USER_SUBSCR_REGISTRATIONS`.

AQ$INTERNET_USERS: Oracle Database Advanced Queuing Agents Registered for Internet Access

The `AQ$INTERNET_USERS` view provides information about the agents registered for Internet access to Oracle Database Advanced Queuing. It also provides the list of database users that each Internet agent maps to.

Table 9-6 AQ$INTERNET_USERS View

Column	Datatype	NULL	Description
`AGENT_NAME`	`VARCHAR2(30)`	-	Name of the Oracle Database Advanced Queuing Internet agent
`DB_USERNAME`	`VARCHAR2(30)`	-	Name of database user that this Internet agent maps to
`HTTP_ENABLED`	`VARCHAR2(4)`	-	Indicates whether this agent is allowed to access Oracle Database Advanced Queuing through HTTP (`YES` or `NO`)
`FTP_ENABLED`	`VARCHAR2(4)`	-	Indicates whether this agent is allowed to access Oracle Database Advanced Queuing through FTP (always `NO` in current release)

V$AQ: Number of Messages in Different States in Database

The `V$AQ` view provides information about the number of messages in different states for the whole database.

In a Oracle Real Application Clusters environment, each instance keeps its own Oracle Database Advanced Queuing statistics information in its own System Global Area (SGA), and does not have knowledge of the statistics gathered by other instances. When a `GV$AQ` view is queried by an instance, all other instances funnel their Oracle Database Advanced Queuing statistics information to the instance issuing the query.

See Also:

Oracle Database Reference for more information about `V$AQ`.

V$BUFFERED_QUEUES: All Buffered Queues in the Instance

The `V$BUFFERED_QUEUES` view displays information about all buffered queues in the instance. There is one row per queue.

See Also:

Oracle Database Reference for more information about `V$BUFFERED_QUEUES`.

V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance

The `V$BUFFERED_SUBSCRIBERS` view displays information about the subscribers for all buffered queues in the instance. There is one row per subscriber per queue.

See Also:

Oracle Database Reference for more information about `V$BUFFERED_SUBSCRIBERS`.

V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance

The `V$BUFFERED_PUBLISHERS` view displays information about all buffered publishers in the instance.

There is one row per queue per sender. The values are reset to zero when the database (or instance in an Oracle RAC environment) restarts.

See Also:

Oracle Database Reference for more information about `V$BUFFERED_PUBLISHERS`.

V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance

The `V$PERSISTENT_QUEUES` view displays information about all active persistent queues in the database since the queues' first activity time.

There is one row per queue. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.

See Also:

Oracle Database Reference for more information about `V$PERSISTENT_QUEUES`.

V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues

The `V$PERSISTENT_QMN_CACHE` view displays detailed statistics about all background activities relating to all queue tables in the database.

There is one row per queue table. The values are reset when the database (or instance in an Oracle RAC environment) restarts.

See Also:

Oracle Database Reference for more information about `V$PERSISTENT_QMN_CACHE`.

V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance

The `V$PERSISTENT_SUBSCRIBERS` view displays information about all active subscribers of the persistent queues in the database.

There is one row per instance per queue per subscriber. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.

See Also:

Oracle Database Reference for more information about `V$PERSISTENT_SUBSCRIBERS`.

V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance

The `V$PERSISTENT_PUBLISHERS` view displays information about all active publishers of the persistent queues in the database.

There is one row per instance per queue per publisher. The rows are deleted when the database (or instance in an Oracle RAC environment) restarts.

See Also:

Oracle Database Reference for more information about V$PERSISTENT_PUBLISHERS.

V$PROPAGATION_SENDER: Buffer Queue Propagation Schedules on the Sending (Source) Side

The `V$PROPAGATION_SENDER` view displays information about buffer queue propagation schedules on the sending (source) side.

The values are reset to zero when the database (or instance in a Oracle Real Application Clusters (Oracle RAC) environment) restarts, when propagation migrates to another instance, or when an unscheduled propagation is attempted.

See Also:

Oracle Database Reference for more information about `V$PROPAGATION_SENDER` .

V$PROPAGATION_RECEIVER: Buffer Queue Propagation Schedules on the Receiving (Destination) Side

The `V$PROPAGATION_RECEIVER` view displays information about buffer queue propagation schedules on the receiving (destination) side.

The values are reset to zero when the database (or instance in a Oracle Real Application Clusters (Oracle RAC) environment) restarts, when propagation migrates to another instance, or when an unscheduled propagation is attempted.

See Also:

Oracle Database Reference for more information about `V$PROPAGATION_RECEIVER`.

V$SUBSCR_REGISTRATION_STATS: Diagnosability of Notifications

The `V$SUBSCR_REGISTRATION_STATS` view provides information for diagnosability of notifications.

See Also:

Oracle Database Reference for more information about `V$SUBSCR_REGISTRATION_STATS`.

V$METRICGROUP: Information About the Metric Group

This `V$METRICGROUP` view displays information about the metric group for each of the four major Streams components: capture, propagation, apply, and queue.

See Also:

Oracle Database Reference for more information about `V$METRICGROUP`.

V$AQ_MESSAGE_CACHE_STAT: Memory Management for Sharded Queues

The `V$AQ_MESSAGE_CACHE_STAT` view displays statistics about memory management for sharded queues in `streams_pool` within the System Global Area (SGA). Sharded queue uses `streams_pool` in units of subshards. Thus columns of this view shows statistics at subshard level irrespective of the queue. This view shows statistics across all sharded queues.

See Also:

Oracle Database Reference for more information about `V$AQ_MESSAGE_CACHE_STAT`.

Note:

Some of the above mentioned columns will be used by sharded queue memory advisor during analysis.

V$AQ_SHARDED_SUBSCRIBER_STAT: Sharded Queue Subscriber Statistics

The `V$AQ_SHARDED_SUBSCRIBER_STAT` view displays statistical information about the subscribers of sharded queues. This statistics is used by the memory advisor.

See Also:

Oracle Database Reference for more information about `V$AQ_SHARDED_SUBSCRIBER_STAT`.

V$AQ_MESSAGE_CACHE_ADVICE: Simulated Metrics

The `V$AQ_MESSAGE_CACHE_ADVICE` view shows simulated metrics for a range of potential message cache sizes. This view assists in cache sizing by providing information in form of metrics as described below.

See Also:

Oracle Database Reference for more information about `V$AQ_MESSAGE_CACHE_ADVICE`.

V$AQ_REMOTE_DEQUEUE_AFFINITY: Dequeue Affinity Instance List

The `V$AQ_REMOTE_DEQUEUE_AFFINITY` view lists the dequeue affinity instance of the subscribers not dequeuing locally from the shard's owner instance. Cross instance message forwarding is used for these subscribers.

See Also:

Oracle Database Reference for more information about `V$AQ_REMOTE_DEQUEUE_AFFINITY`.

V$AQ_BACKGROUND_COORDINATOR: Performance Statistics for AQ's Primary Background Coordinator Process (AQPC)

The `V$AQ_BACKGROUND_COORDINATOR` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards.

This view lists performance statistics for the Oracle Database Advanced Queuing primary background coordinator process (AQPC).

See Also:

Oracle Database Reference for more information about `V$AQ_BACKGROUND_COORDINATOR`.

V$AQ_JOB_COORDINATOR: Performance Statistics per Coordinator

The `V$AQ_JOB_COORDINATOR` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards.

This view lists performance statistics per coordinator, for every AQ coordinator controlled by the AQ's primary coordinator.

See Also::

Oracle Database Reference for more information about `V$AQ_JOB_COORDINATOR`.

V$AQ_SERVER_POOL: Performance Statistics for all Servers

The `V$AQ_SERVER_POOL` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. This view lists performance statistics for all the servers in the pool.

See Also::

Oracle Database Reference for more information about `V$AQ_SERVER_POOL`.

V$AQ_CROSS_INSTANCE_JOBS: Cross Process Jobs Description

The `V$AQ_CROSS_INSTANCE_JOBS` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. This view describes each of the cross process jobs.

Each job serves for forwarding messages for a shard from source instance to destination instance for a set of subscribers.

See Also::

Oracle Database Reference for more information about `V$AQ_CROSS_INSTANCE_JOBS`.

V$AQ_IPC_ACTIVE_MSGS

`V$AQ_IPC_ACTIVE_MSGS` displays information about long and priority messages being processed by secondary processes and the short message being processed by the primary process.

See Also:

Oracle Database Reference for more information about `V$AQ_IPC_ACTIVE_MSGS`.

V$AQ_IPC_MSG_STATS

V$AQ_IPC_MSG_STATS displays cumulative statistics for each message class, for example., cumulative calls, average pending/processing time, and last failure.

See Also:

Oracle Database Reference for more information about `V$AQ_IPC_MSG_STATS`.

V$AQ_IPC_PENDING_MSGS

V$AQ_IPC_PENDING_MSGS displays information about pending messages, present in the local primary context.

See Also:

Oracle Database Reference for more information about `V$AQ_IPC_PENDING_MSGS`.

V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations

The `V$AQ_NONDUR_REGISTRATIONS` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. This view provides information about non-durable subscriptions.

See Also::

Oracle Database Reference for more information about `V$AQ_NONDUR_REGISTRATIONS`.

V$AQ_NOTIFICATION_CLIENTS: Secure OCI Client Connections

The `V$AQ_NOTIFICATION_CLIENTS` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. This view displays performance statistics for secure OCI client connections.

See Also::

Oracle Database Reference for more information about `V$AQ_NOTIFICATION_CLIENTS`.

V$AQ_SUBSCRIBER_LOAD: Durable Subscribers

The `V$AQ_SUBSCRIBER_LOAD` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. This view describes the load of all subscribers of sharded queues in terms of latency at every instance in an Oracle RAC environment.

Latency denotes the predicted amount of time (in seconds) required from the current time to drain all the messages for that subscriber at each respective instance. The latency calculation considers past enqueue/dequeue rates and future enqueue/dequeue rates based on history.

See Also::

Oracle Database Reference for more information about `V$AQ_SUBSCRIBER_LOAD`.

V$AQ_NONDUR_SUBSCRIBER: Non-Durable Subscribers

The `V$AQ_NONDUR_SUBSCRIBER` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. `V$AQ_NONDUR_SUBSCRIBER` provides information about non-durable subscribers on sharded queues.

See Also::

Oracle Database Reference for more information about `V$AQ_NONDUR_SUBSCRIBER`.

V$AQ_NONDUR_SUBSCRIBER_LWM: LWM of Non Durable Subscriber

The `V$AQ_NONDUR_SUBSCRIBER_LWM` view is applicable for Oracle Database 12*c* Release 1 (12.1) onwards. The LWM of a non durable subscriber is a combination of shard, priority and LWM (sub-shard).

See Also::

Oracle Database Reference for more information about `V$AQ_NONDUR_SUBSCRIBER_LWM`.

V$AQ_MESSAGE_CACHE: Performance Statistics

The `V$AQ_MESSAGE_CACHE` view provides performance statistics of the message cache for sharded queues at the subshard level in the instance.

See Also::

Oracle Database Reference for more information about `V$AQ_MESSAGE_CACHE`.

10

Troubleshooting Oracle Database Advanced Queuing

These topics describe how to troubleshoot Oracle Database Advanced Queuing (AQ).

- Debugging Oracle Database Advanced Queuing Propagation Problems
- Oracle Database Advanced Queuing Error Messages

Debugging Oracle Database Advanced Queuing Propagation Problems

These tips should help with debugging propagation problems. This discussion assumes that you have created queue tables and queues in source and target databases and defined a database link for the destination database.

The notation assumes that you supply the actual name of the entity (without the brackets).

> **See Also:**
>
> "Optimizing Propagation"

To begin debugging, do the following:

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

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

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

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

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

   ```
   select count(*) from table_name@dblink_name;
   ```

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

4. Check for messages in the source queue with:

   ```
   select count (*) from AQ$<source_queue_table>
     where q_name = 'source_queue_name';
   ```

5. Check for messages in the destination queue with:

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

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

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

7. Check to see that the queue table `sys.aq$_prop_table_`*`instno`* exists in `DBA_QUEUE_TABLES`. The queue `sys.aq$_prop_notify_queue_`*`instno`* must also exist in `DBA_QUEUES` and must be enabled for enqueue and dequeue.

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

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

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

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

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

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

   Note:

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

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

   Debugging information is logged to job queue trace files as propagation takes place. You can check the trace file for errors and for statements indicating that messages have been sent.

Oracle Database Advanced Queuing Error Messages

The Oracle Database Advanced Queuing Error Messages are listed here.

ORA-1555

You might get this error when using the `NEXT_MESSAGE` navigation option for dequeue. `NEXT_MESSAGE` uses the snapshot created during the first dequeue call. After that, undo information may not be retained.

The workaround is to use the `FIRST_MESSAGE` option to dequeue the message. This reexecutes the cursor and gets a new snapshot. `FIRST_MESSAGE` does not perform as well as `NEXT_MESSAGE`, so Oracle recommends that you dequeue messages in batches: `FIRST_MESSAGE` for one, `NEXT_MESSAGE` for the next 1000 messages, then `FIRST_MESSAGE` again, and so on.

ORA-24033

This error is raised if a message is enqueued to a multiconsumer queue with no recipient and the queue has no subscribers (or rule-based subscribers that match this message). This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

ORA-25237

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

ORA-25307

Flow control has been enabled for the message sender. This means that the fastest subscriber of the sender's message is not able to keep pace with the rate at which messages are enqueued. The buffered messaging application must handle this error and attempt again to enqueue messages after waiting for some time.

11 Internet Access to Oracle Database Advanced Queuing

You can access Oracle Database Advanced Queuing (AQ) over the Internet by using SOAP with AQ queues. IDAP is the SOAP specification for Oracle Database Advanced Queuing operations.

IDAP defines XML message structure for the body of the Simple Object Access Protocol (SOAP) request. An Internet Data Access Presentation (IDAP)-structured message is transmitted over the Internet using HTTP.

Users can register for notifications using the IDAP interface.

Topics:

- Overview of Oracle Database Advanced Queuing Operations Over the Internet
- Deploying the Oracle Database Advanced Queuing XML Servlet
- Internet Data Access Presentation (IDAP)
- Request and Response IDAP Documents
- Notification of Messages by E-Mail

Overview of Oracle Database Advanced Queuing Operations Over the Internet

The section discusses these topics.

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

Oracle Database Advanced Queuing Internet Operations Architecture

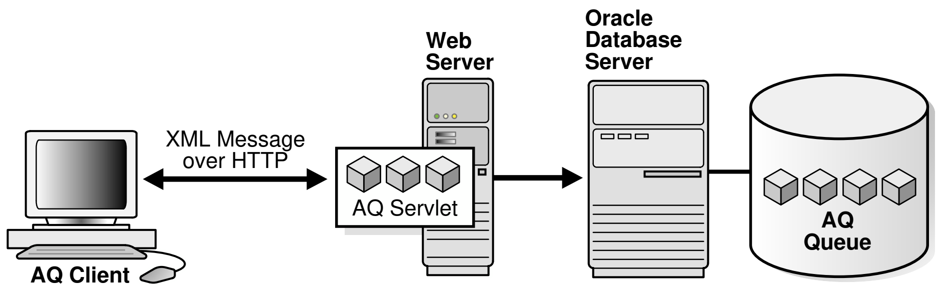
The figure shows the architecture for performing Oracle Database Advanced Queuing operations over HTTP.

The major components are:

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

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

Figure 11-1 Architecture for Performing Oracle Database Advanced Queuing Operations Using HTTP



Internet Message Payloads

Oracle Database Advanced Queuing supports messages of three types: RAW, Oracle object, and JMS. All these message types can be accessed using SOAP and Web services.

If the queue holds messages in RAW, Oracle object, or Java Message Service (JMS) format, then XML payloads are transformed to the appropriate internal format during enqueue and stored in the queue. During dequeue, when messages are obtained from queues containing messages in any of the preceding formats, they are converted to XML before being sent to the client.

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

RAW Queues

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

Oracle Object Type Queues

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

JMS Type Queues/Topics

For queues with JMS types (that is, those with payloads of type `AQ$_JMS_*`), there are four XML elements, depending on the JMS type. IDAP supports queues or topics with the following JMS types:

- `TextMessage`
- `MapMessage`
- `BytesMessage`

- `ObjectMessage`

JMS queues with payload type `StreamMessage` are not supported through IDAP.

Configuring the Web Server to Authenticate Users Sending POST Requests

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

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

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

Client Requests Using HTTP

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

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

See Also:

"Request and Response IDAP Documents"

User Sessions and Transactions

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

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

An explicit commit or rollback must be applied to end the transaction. The commit or rollback requests can also be included as part of other Oracle Database Advanced Queuing operations.

Oracle Database Advanced Queuing Servlet Responses Using HTTP

The server accepts the client HTTP(S) connection and authenticates the user (Oracle Database Advanced Queuing agent) specified by the client. The server receives the `POST` request and invokes the Oracle Database Advanced Queuing servlet.

If this is the first request from this client, then a new HTTP session is created. The XML message is parsed and its contents are validated. If a session ID is passed by the client in the HTTP headers, then this operation is performed in the context of that session.

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

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

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

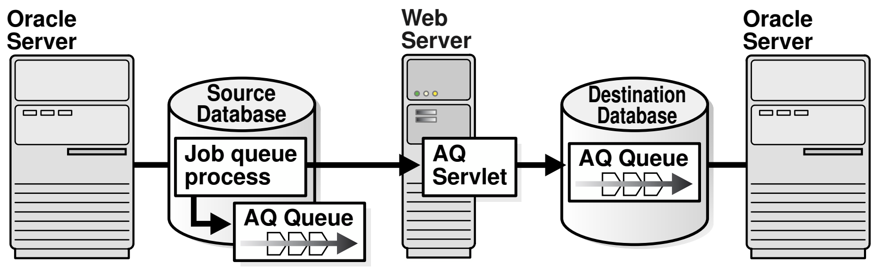
See Also:

"User Sessions and Transactions"

Oracle Database Advanced Queuing Propagation Using HTTP and HTTPS

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

Figure 11-2 HTTP Oracle Database Advanced Queuing Propagation



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

Deploying the Oracle Database Advanced Queuing XML Servlet

The AQ servlet can be deployed with any Web server, for example, Tomcat. Follow these steps to deploy the AQ XML servlet using Tomcat:

1. For JDK1.8.x, include the following in your `CLASSPATH`:

   ```
   ORACLE_HOME/jdbc/lib/ojdbc8.jar
   ORACLE_HOME/jlib/jndi.jar
   ORACLE_HOME/jlib/jta.jar
   ORACLE_HOME/jlib/orai18n.jar
   ORACLE_HOME/jlib/orai18n-collation.jar
   ORACLE_HOME/jlib/orai18n-mapping.jar
   ORACLE_HOME/jlib/orai18n-utility.jar
   ORACLE_HOME/lib/http_client.jar
   ORACLE_HOME/lib/lclasses12.zip
   ORACLE_HOME/lib/servlet.jar
   ORACLE_HOME/lib/xmlparserv2.jar
   ORACLE_HOME/lib/xschema.jar
   ORACLE_HOME/lib/xsu12.jar
   ORACLE_HOME/rdbms/jlib/aqapi.jar
   ORACLE_HOME/rdbms/jlib/aqxml.jar
   ORACLE_HOME/rdbms/jlib/jmscommon.jar
   ORACLE_HOME/rdbms/jlib/xdb.jar
   ```

2. Copy the following jar files into the `tomcat/lib` directory:

   ```
   ORACLE_HOME/jdbc/lib/ojdbc8.jar
   ORACLE_HOME/jlib/jndi.jar
   ORACLE_HOME/jlib/jta.jar
   ORACLE_HOME/lib/http_client.jar
   ORACLE_HOME/lib/lclasses12.zip
   ORACLE_HOME/lib/servlet.jar
   ORACLE_HOME/lib/xmlparserv2.jar
   ORACLE_HOME/lib/xschema.jar
   ORACLE_HOME/lib/xsu12.jar
   ORACLE_HOME/rdbms/jlib/aqapi.jar
   ORACLE_HOME/rdbms/jlib/aqxml.jar
   ORACLE_HOME/rdbms/jlib/jmscommon.jar
   ORACLE_HOME/rdbms/jlib/xdb.jar
   ```

3. Create or update `tomcat-users.xml` file appropriately for Web applications users accessing queues. For example:

   ```
   User            Password
   -----------------------------------
   john                  welcome
   ```

4. Set up queues in database and create AQ agents so that Tomcat users created in step 3 get authenticated before it can access AQ queues. DBA needs to make use of `DBMS_AQADM.CREATE_AQ_AGENT` and `DBMS_AQADM.ENABLE_DB_ACCESS` procedures. For example, if we assume `JOHN` is the user created in Tomcat and `AQXMLUSER` is the AQ agent created on the database, then in order to access AQ servlet using HTTP, run the following queries:

   ```
   EXECUTE dbms_aqadm.create_aq_agent(agent_name=>'JOHN', enable_http =>true);
   EXECUTE dbms_aqadm.enable_db_access('JOHN', 'AQXMLUSER');
   ```

   Here `AQXMLUSER` is the AQ user that is created in the database.

   DBA can check internet AQ users agents details using the following query :

   ```
   SELECT agent_name, db_username, http_enabled FROM aq$internet_users ;
   ```

5. Deploy the AQ XML servlet, which extends `oracle.AQ.xml.AQxmlServlet` class.

6. Start or stop the Tomcat instance as follows:

   a. Start the Tomcat instance using `sh tomcat/bin/startup.sh`

b. Shutdown the Tomcat instance using `sh tomcat/bin/shutdown.sh`
c. For logs in Tomcat check `tomcat/logs/catalina.out` file

Internet Data Access Presentation (IDAP)

Internet Data Access Presentation (IDAP) uses the Content-Type of `text/xml` to specify the body of the SOAP request.

XML provides the presentation for IDAP request and response messages as follows:

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

SOAP Message Structure

These topics shows how SOAP structures a message request or response.

- SOAP Envelope
- SOAP Header
- SOAP Body

SOAP Envelope

This is the root or top element in an XML tree. Its tag is `SOAP:Envelope`. SOAP defines a global attribute `SOAP:encodingStyle` that indicates serialization rules used instead of those described by the SOAP specification.

This attribute can appear on any element and is scoped to that element and all child elements not themselves containing such an attribute. Omitting this attribute means that type specification has been followed unless overridden by a parent element.

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

SOAP Header

This is the first element under the root. Its tag is `SOAP:Header`. A SOAP header passes necessary information, such as the transaction identifier.

The header is encoded as a child of the `SOAP:Envelope` XML element. Headers are identified by the name element and are namespace-qualified. A header entry is encoded as an embedded element.

SOAP Body

This is the Oracle Database Advanced Queuing XML document. Its tag is `SOAP:Body`, and it contains a first subelement whose name is the method name.

This method request element contains elements for each input and output parameter. The element names are the parameter names. The body also contains `SOAP:Fault`, indicating information about an error. The Oracle Database Advanced Queuing XML document has the namespace `http://ns.oracle.com/AQ/schemas/access`

SOAP Method Invocation

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

HTTP Headers

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

For example:

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

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

Method Invocation Body

SOAP method invocation consists of a method request and optionally a method response. The SOAP method request and method response are an HTTP request and response, respectively, whose contents are XML documents consisting of the root and mandatory body elements.

These XML documents are referred to as SOAP payloads in the rest of the sections.

A SOAP payload is defined as follows:

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

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

- The HTTP infrastructure on the receiving site can receive and process the request. In this case, the HTTP infrastructure passes the headers and body to the SOAP infrastructure.

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

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

Results from a Method Request

The results of the request are to be provided in the form of a request response. The HTTP response must be of Content-Type `text/xml`.

A SOAP result indicates success and an error indicates failure. The method response never contains both a result and an error.

Request and Response IDAP Documents

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

The body represents:

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

> **Note:**
>
> Oracle Database Advanced Queuing Internet access is supported only for 8.1 or higher style queues.
>
> Transactional Event Queues (TxEventQ) do not support internet access through SOAP.

This section contains these topics:

- IDAP Client Requests for Enqueue
- IDAP Client Requests for Dequeue
- IDAP Client Requests for Registration
- IDAP Client Requests to Commit a Transaction
- IDAP Client Requests to Roll Back a Transaction
- IDAP Server Response to an Enqueue Request

- IDAP Server Response to a Dequeue Request
- IDAP Server Response to a Register Request
- IDAP Commit Response
- IDAP Rollback Response
- IDAP Notification
- IDAP Response in Case of Error

IDAP Client Requests for Enqueue

Client send and publish requests use `AQXmlSend` to enqueue to a single-consumer queue and `AQXmlPublish` to enqueue to multiconsumer queues/topics.

`AQXmlSend` and `AQXmlPublish` contain the following elements:

- producer_options
- message_set
- message_header
- message_payload
- AQXmlCommit

producer_options

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

- `destination`

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

- `visibility`

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

- `transformation`

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

message_set

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

message_header

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

- `sender_id`

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

- message_id

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

- correlation

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

- delay

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

- expiration

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

- priority

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

- recipient_list

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

- message_state

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

- exception_queue

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

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

message_payload

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

AQXmlCommit

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

> **See Also:**
>
> "Internet Message Payloads" for an explanation of IDAP message payloads

IDAP Client Requests for Dequeue

Client requests for dequeue use `AQXmlReceive`, which contains these elements.

- consumer_options
- AQXmlCommit

consumer_options

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

- `destination`

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

- `consumer_name`

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

- `wait_time`

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

- `selector`

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

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

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

  A dequeue `condition` element cannot exceed 4000 characters.

Note:

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

- `visibility`

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

- `dequeue_mode`

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

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

- `navigation_mode`

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

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

- `transformation`

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

AQXmlCommit

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

IDAP Client Requests for Registration

Client requests for registration use `AQXmlRegister`, which must contain a `register_options` element. The `register_options` element contains these child elements.

- `destination`

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

- `consumer_name`

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

- `notify_url`

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

IDAP Client Requests to Commit a Transaction

A request to commit all actions performed by the user in a session uses `AQXmlCommit`.

A commit request has the following format:

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

IDAP Client Requests to Roll Back a Transaction

A request to roll back all actions performed by the user in a session uses `AQXmlRollback`. Actions performed with `IMMEDIATE` visibility are not rolled back.

An IDAP client rollback request has the following format:

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

IDAP Server Response to an Enqueue Request

The response to an enqueue request to a single-consumer queue uses `AQXmlSendResponse`.

It contains the following elements:

- `status_response`

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

- `send_result`

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

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

- `status_response`

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

- `publish_result`

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

IDAP Server Response to a Dequeue Request

The response to a dequeue request uses `AQXmlReceiveResponse`.

It contains the following elements:

- `status_response`

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

- `receive_result`

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

IDAP Server Response to a Register Request

The response to a register request uses `AQXmlRegisterResponse`.

It contains the `status_response` element described in "IDAP Server Response to a Dequeue Request".

IDAP Commit Response

The response to a commit request uses `AQXmlCommitResponse`.

It contains the `status_response` element described in "IDAP Server Response to a Dequeue Request". The response to a commit request has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <AQXmlCommitResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
         <status_response>
            <status_code>0</status_code>
         </status_response>
      </AQXmlCommitResponse>
   </Body>
</Envelope>
```

IDAP Rollback Response

The response to a rollback request uses `AQXmlRollbackResponse`.

It contains the `status_response` element described in "IDAP Server Response to a Dequeue Request".

IDAP Notification

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

It contains the following elements:

- `notification_options`

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

- `message_set`

  This element specifies the set of message properties.

IDAP Response in Case of Error

In case of an error in any of the preceding requests, a `FAULT` is generated.

The `FAULT` element contains the following elements:

- `faultcode`

  This element specifies the error code for the fault.

- `faultstring`

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

- `detail`

  This element contains the `status_response` element, which is described in "IDAP Server Response to a Dequeue Request".

Notification of Messages by E-Mail

These are the steps for setting up your database for e-mail notifications.

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

12

Oracle Database Advanced Queuing Administrative Interface

These topics describe the Oracle Database Advanced Queuing (AQ) administrative interface.

- Managing AQ Queue Tables
- Managing AQ Queues
- Managing Sharded Queues
- Managing Transformations
- Granting and Revoking Privileges
- Managing Subscribers
- Managing Propagations
- Managing Oracle Database Advanced Queuing Agents
- Adding an Alias to the LDAP Server
- Deleting an Alias from the LDAP Server

See Also:

- Oracle Transactional Event Queues and Advanced Queuing: Programmatic Interfaces for a list of available functions in each programmatic interface
- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_AQADM Package

Managing AQ Queue Tables

These topics describe how to manage AQ queue tables.

- Creating a Queue Table
- Altering a Queue Table
- Dropping a Queue Table
- Purging a Queue Table
- Migrating a Queue Table

Creating an AQ Queue Table

`DBMS_AQADM.CREATE_QUEUE_TABLE` creates an AQ queue table for messages of a predefined type.

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table          IN      VARCHAR2,
   queue_payload_type   IN      VARCHAR2,
   storage_clause       IN      VARCHAR2        DEFAULT NULL,
   sort_list            IN      VARCHAR2        DEFAULT NULL,
   multiple_consumers   IN      BOOLEAN         DEFAULT FALSE,
   message_grouping     IN      BINARY_INTEGER  DEFAULT NONE,
   comment              IN      VARCHAR2        DEFAULT NULL,
   auto_commit          IN      BOOLEAN         DEFAULT TRUE,
   primary_instance     IN      BINARY_INTEGER  DEFAULT 0,
   secondary_instance   IN      BINARY_INTEGER  DEFAULT 0,
   compatible           IN      VARCHAR2        DEFAULT NULL,
   secure               IN      BOOLEAN         DEFAULT FALSE
   replication_mode     IN      BINARY_INTEGER  DEFAULT NONE);
```

It has the following required and optional parameters:

Parameter	Description
`queue_table`	This required parameter specifies the queue table name. Mixed case (upper and lower case together) queue table names are supported if database compatibility is 10.0, but the names must be enclosed in double quote marks. So `abc.efg` means the schema is `ABC` and the name is `EFG`, but `"abc"."efg"` means the schema is `abc` and the name is `efg`. Starting from 12*c* Release 2 (12.2.), the maximum length of AQ queue table names is 122 bytes. If you attempt to create a queue table with a longer name, error ORA-24019 results.
`queue_payload_type`	This required parameter specifies the payload type as RAW or an object type. See "Payload Type" for more information.
`storage_clause`	This optional parameter specifies a tablespace for the queue table. See "Storage Clause" for more information.
`sort_list`	This optional parameter specifies one or two columns to be used as sort keys in ascending order. It has the format *`sort_column1,sort_column2`*. See "Sort Key" for more information.
`multiple_consumers`	This optional parameter specifies the queue table as single-consumer or multiconsumer. The default `FALSE` means queues created in the table can have only one consumer for each message. `TRUE` means queues created in the table can have multiple consumers for each message.

Parameter	Description
`message_grouping`	This optional parameter specifies whether messages are grouped or not. The default `NONE` means each message is treated individually. `TRANSACTIONAL` means all messages enqueued in one transaction are considered part of the same group and can be dequeued as a group of related messages.
`comment`	This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.
`auto_commit`	`TRUE` causes the current transaction, if any, to commit before the `CREATE_QUEUE_TABLE` operation is carried out. The `CREATE_QUEUE_TABLE` operation becomes persistent when the call returns. This is the default. `FALSE` means the operation is part of the current transaction and becomes persistent only when the caller enters a commit. Note: This parameter has been deprecated.
`primary_instance`	This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value `0` means queue monitor scheduling and propagation is done in any available instance. You can specify and modify this parameter only if `compatible` is `8.1` or higher.
`secondary_instance`	This optional parameter specifies the owner of the queue table if the primary instance is not available. The default value `0` means that the queue table will fail over to any available instance. You can specify and modify this parameter only if `primary_instance` is also specified and `compatible` is `8.1` or higher.
`compatible`	This optional parameter specifies the lowest database version with which the queue table is compatible. The possible values are `8.0`, `8.1`, and `10.0`. If the database is in 10.1-compatible mode, then the default value is `10.0`. If the database is in 8.1-compatible or 9.2-compatible mode, then the default value is `8.1`. If the database is in 8.0-compatible mode, then the default value is `8.0`. The `8.0` value is deprecated in Oracle Database Advanced Queuing 10*g* Release 2 (10.2). For more information on compatibility, see "Oracle Database Advanced Queuing Compatibility Parameters".

Parameter	Description
`secure`	This optional parameter must be set to `TRUE` if you want to use the queue table for secure queues. Secure queues are queues for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot unless they are configured as secure queue users
`replication_mode`	Reserved for future use. `DBMS_AQADM.REPLICATION_MODE` if queue is being created in the Replication Mode or else `DBMS_AQADM.NONE`. Default is `DBMS_AQADM.NONE`.

Payload Type

To specify the payload type as an object type, you must define the object type.

Note:

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

CLOB, BLOB, and BFILE objects are valid in an Oracle Database Advanced Queuing message. You can propagate these object types using Oracle Database Advanced Queuing propagation with Oracle software since Oracle8*i* release 8.1.x. To enqueue an object type that has a LOB, you must first set the `LOB_attribute` to `EMPTY_BLOB()` and perform the enqueue. You can then select the `LOB` locator that was generated from the queue table's view and use the standard `LOB` operations.

Note:

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

Storage Clause

The `storage_clause` argument can take any text that can be used in a standard `CREATE TABLE storage_clause` argument.

Once you pick the tablespace, any index-organized table (IOT) or index created for that queue table goes to the specified tablespace. You do not currently have a choice to split them between different tablespaces.

Note:

The qmon processes in the 11g Release 2 (11.2) perform auto-coalesce of the the dequeue IOT, history IOT, and the time manager IOT. It is not required to manually coalesce AQ IOTs. However, it can be performed as a workaround if a performance degradation is observed.

If you choose to create the queue table in a locally managed tablespace or with freelist groups > 1, then Queue Monitor Coordinator will skip the cleanup of those blocks. This can cause a decline in performance over time.

Coalesce the dequeue IOT by running

```
ALTER TABLE AQ$_queue_table_I COALESCE;
```

You can run this command while there are concurrent dequeuers and enqueuers of the queue, but these concurrent users might see a slight decline in performance while the command is running.

Sort Key

The sort_list parameter determines the order in which messages are dequeued. You cannot change the message sort order after you have created the queue table. Your choices are:

- `ENQ_TIME`
- `ENQ_TIME,PRIORITY`
- `PRIORITY`
- `PRIORITY,ENQ_TIME`
- `PRIORITY,COMMIT_TIME`
- `COMMIT_TIME`

If `COMMIT_TIME` is specified, then any queue that uses the queue table is a commit-time queue, and Oracle Database Advanced Queuing computes an approximate CSCN for each enqueued message when its transaction commits.

If you specify `COMMIT_TIME` as the sort key, then you must also specify the following:

- `multiple_consumers` = `TRUE`
- `message_grouping` = `TRANSACTIONAL`
- `compatible` = `8.1` or higher

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

Other Tables and Views

The following objects are created at table creation time:

- `AQ$_queue_table_name`, a read-only view which is used by Oracle Database Advanced Queuing applications for querying queue data
- `AQ$_queue_table_name_E`, the default exception queue associated with the queue table

- `AQ$_queue_table_name_I`, an index or an index-organized table (IOT) in the case of multiple consumer queues for dequeue operations
- `AQ$_queue_table_name_T`, an index for the queue monitor operations
- `AQ$_queue_table_name_L`, dequeue log table, used for storing message identifiers of committed dequeue operations on the queue

The following objects are created only for 8.1-compatible multiconsumer queue tables:

- `AQ$_queue_table_name_S`, a table for storing information about subscribers
- `AQ$_queue_table_name_H`, an index organized table (IOT) for storing dequeue history data

Note:

Oracle Database Advanced Queuing does not support the use of triggers on these internal AQ queue tables.

If you do not specify a schema, then you default to the user's schema.

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue table is created, then a corresponding Lightweight Directory Access Protocol (LDAP) entry is also created.

If the queue type is `ANYDATA`, then a buffered queue and two additional objects are created. The buffered queue stores logical change records created by a capture process. The logical change records are staged in a memory buffer associated with the queue; they are not ordinarily written to disk.

If they have been staged in the buffer for a period of time without being dequeued, or if there is not enough space in memory to hold all of the captured events, then they are spilled to:

- `AQ$_queue_table_name_P`, a table for storing the captured events that spill from memory
- `AQ$_queue_table_name_D`, a table for storing information about the propagations and apply processes that are eligible for processing each event

See Also:

- "Dequeue Modes"
- *Oracle Database SecureFiles and Large Objects Developer's Guide*

Examples

The following examples assume you are in a SQL*Plus testing environment. In Example 12-1, you create users in preparation for the other examples in this chapter. For this example, you must connect as a user with administrative privileges. For most of the other examples in this chapter, you can connect as user `test_adm`. A few examples must be run as `test` with `EXECUTE` privileges on `DBMS_AQADM`.

Example 12-1 Setting Up AQ Administrative Users

```
CREATE USER test_adm IDENTIFIED BY test_adm DEFAULT TABLESPACE example;
GRANT DBA, CREATE ANY TYPE TO test_adm;
GRANT EXECUTE ON DBMS_AQADM TO test_adm;
GRANT aq_administrator_role TO test_adm;
BEGIN
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
      privilege          =>    'MANAGE_ANY',
      grantee            =>    'test_adm',
      admin_option       =>     FALSE);
END;
/
CREATE USER test IDENTIFIED BY test;
GRANT EXECUTE ON dbms_aq TO test;
```

Example 12-2   Setting Up AQ Administrative Example Types

```
CREATE TYPE test.message_typ AS object(
   sender_id              NUMBER,
   subject                VARCHAR2(30),
   text                   VARCHAR2(1000));
/
CREATE TYPE test.msg_table AS TABLE OF test.message_typ;
/
CREATE TYPE test.order_typ AS object(
   custno                 NUMBER,
   item                   VARCHAR2(30),
   description            VARCHAR2(1000));
/
CREATE TYPE test.lob_typ AS object(
   id                     NUMBER,
   subject                VARCHAR2(100),
   data                   BLOB,
   trailer                NUMBER);
/
```

Example 12-3   Creating a Queue Table for Messages of Object Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.obj_qtab',
      queue_payload_type     => 'test.message_typ');
END;
/
```

Example 12-4   Creating a Queue Table for Messages of RAW Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.raw_qtab',
      queue_payload_type     => 'RAW');
END;
/
```

Example 12-5   Creating a Queue Table for Messages of LOB Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.lob_qtab',
      queue_payload_type     => 'test.lob_typ');
END;
/
```

Example 12-6   Creating a Queue Table for Messages of XMLType

```
BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
     queue_table         => 'test.xml_qtab',
     queue_payload_type  => 'SYS.XMLType',
     multiple_consumers  => TRUE,
     compatible          => '8.1',
     comment             => 'Overseas Shipping multiconsumer orders queue table');
END;
/
```

Example 12-7 Creating a Queue Table for Grouped Messages

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table          => 'test.group_qtab',
      queue_payload_type   => 'test.message_typ',
      message_grouping     => DBMS_AQADM.TRANSACTIONAL);
END;
/
```

Example 12-8 Creating Queue Tables for Prioritized Messages and Multiple Consumers

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.priority_qtab',
      queue_payload_type     => 'test.order_typ',
      sort_list              => 'PRIORITY,ENQ_TIME',
      multiple_consumers     => TRUE);
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.multiconsumer_qtab',
      queue_payload_type     => 'test.message_typ',
      sort_list              => 'PRIORITY,ENQ_TIME',
      multiple_consumers     => TRUE);
END;
/
```

Example 12-9 Creating a Queue Table with Commit-Time Ordering

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.commit_time_qtab',
      queue_payload_type     => 'test.message_typ',
      sort_list              => 'COMMIT_TIME',
      multiple_consumers     => TRUE,
      message_grouping       => DBMS_AQADM.TRANSACTIONAL,
      compatible             => '10.0');
END;
/
```

Example 12-10 Creating an 8.1-Compatible Queue Table for Multiple Consumers

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table            => 'test.multiconsumer_81_qtab',
      queue_payload_type     => 'test.message_typ',
      multiple_consumers     =>  TRUE,
      compatible             => '8.1');
END;
/
```

Example 12-11 Creating a Queue Table in a Specified Tablespace

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table        => 'test.example_qtab',
      queue_payload_type => 'test.message_typ',
      storage_clause     => 'tablespace example');
END;
/
```

Example 12-12 Creating a Queue Table with Freelists or Freelist Groups

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE(
      queue_table             => 'test.freelist_qtab',
      queue_payload_type      => 'RAW',
      storage_clause          => 'STORAGE (FREELISTS 4 FREELIST GROUPS 2)',
      compatible              => '8.1');
END;
/
```

Altering an AQ Queue Table

`DBMS_AQADM.ALTER_QUEUE_TABLE` alters the existing properties of an AQ queue table.

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
   queue_table          IN   VARCHAR2,
   comment              IN   VARCHAR2       DEFAULT NULL,
   primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
   secondary_instance   IN   BINARY_INTEGER DEFAULT NULL
   replication_mode     IN   BINARY_INTEGER DEFAULT NULL);
```

Parameter	Description
`queue_table`	This required parameter specifies the queue table name.
`comment`	This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.
`primary_instance`	This optional parameter specifies the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. You can specify and modify this parameter only if `compatible` is `8.1` or higher.
`secondary_instance`	This optional parameter specifies the owner of the queue table if the primary instance is not available. You can specify and modify this parameter only if `primary_instance` is also specified and `compatible` is `8.1` or higher.
`replication_mode`	Reserved for future use. `DBMS_AQADM.REPLICATION_MODE` if Queue is being altered to be in the Replication Mode or else `DBMS_AQADM.NONE`. Default value is `NULL`.

Note:

In general, DDL statements are not supported on queue tables and may even render them inoperable. For example, issuing an `ALTER TABLE` ... `SHRINK` statement against a queue table results in an internal error, and all subsequent attempts to use the queue table will also result in errors. Oracle recommends that you not use DDL statements on queue tables.

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue table is modified, then a corresponding LDAP entry is also altered.

Example 12-13 Altering a Queue Table by Changing the Primary and Secondary Instances

```
BEGIN
   DBMS_AQADM.ALTER_QUEUE_TABLE(
      queue_table          => 'test.obj_qtab',
      primary_instance     => 3,
      secondary_instance   => 2);
END;
/
```

Example 12-14 Altering a Queue Table by Changing the Comment

```
BEGIN
   DBMS_AQADM.ALTER_QUEUE_TABLE(
      queue_table          => 'test.obj_qtab',
      comment              => 'revised usage for queue table');
END;
/
```

Dropping an AQ Queue Table

`DBMS_AQADM.DROP_QUEUE_TABLE` drops an existing AQ queue table.

```
DBMS_AQADM.DROP_QUEUE_TABLE(
   queue_table       IN     VARCHAR2,
   force             IN     BOOLEAN DEFAULT FALSE,
```

You must stop and drop all the queues in a queue table before the queue table can be dropped. You must do this explicitly if `force` is set to `FALSE`. If `force` is set to `TRUE`, then all queues in the queue table and their associated propagation schedules are dropped automatically.

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue table is dropped, then a corresponding LDAP entry is also dropped.

Example 12-15 Dropping a Queue Table

```
BEGIN
   DBMS_AQADM.DROP_QUEUE_TABLE(
      queue_table        => 'test.obj_qtab');
END;
/
```

Example 12-16 Dropping a Queue Table with force Option

```
BEGIN
   DBMS_AQADM.DROP_QUEUE_TABLE(
      queue_table        => 'test.raw_qtab',
      force              => TRUE);
END;
/
```

Purging an AQ Queue Table

`DBMS_AQADM.PURGE_QUEUE_TABLE` purges messages from an AQ queue table.

```
DBMS_AQADM.PURGE_QUEUE_TABLE(
   queue_table         IN    VARCHAR2,
   purge_condition     IN    VARCHAR2,
   purge_options       IN    aq$_purge_options_t);
```

It has the following parameters:

Parameter	Description
`queue_table`	This required parameter specifies the queue table name.
`purge_condition`	The purge condition must be in the format of a SQL `WHERE` clause, and it is case-sensitive. The condition is based on the columns of `aq$`*`queue_table_name`* view. Each column name in the purge condition must be prefixed with "`qtview.`" All purge conditions supported for persistent messages are also supported for buffered messages. To purge all queues in a queue table, set `purge_condition` to either `NULL` (a bare null word, no quotes) or `''` (two single quotes).
`purge_options`	Type `aq$_purge_options_t` contains a `block` parameter. If `block` is `TRUE`, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueuers and dequeuers to block while the queue table is purged. The purge call always succeeds if `block` is `TRUE`. The default for `block` is `FALSE`. This will not block enqueuers and dequeuers, but it can cause the purge to fail with an error during high concurrency times. Type `aq$_purge_options_t` also contains a `delivery_mode` parameter. If it is the default `PERSISTENT`, then only persistent messages are purged. If it is set to `BUFFERED`, then only buffered messages are purged. If it is set to `PERSISTENT_OR_BUFFERED`, then both types are purged.

A trace file is generated in the udump destination when you run this procedure. It details what the procedure is doing. The procedure commits after it has processed all the messages.

See Also:

"DBMS_AQADM" in *Oracle Database PL/SQL Packages and Types Reference* for more information on `DBMS_AQADM.PURGE_QUEUE_TABLE`

Note:

Some purge conditions, such as `consumer_name` in Example 12-20 and `sender_name` in Example 12-21, are supported only in 8.1-compatible queue tables. For more information, see Table 9-1.

Example 12-17 Purging All Messages in a Queue Table

```
DECLARE
po dbms_aqadm.aq$_purge_options_t;
BEGIN
   po.block := FALSE;
   DBMS_AQADM.PURGE_QUEUE_TABLE(
     queue_table     => 'test.obj_qtab',
     purge_condition => NULL,
     purge_options   => po);
END;
/
```

Example 12-18 Purging All Messages in a Named Queue

```
DECLARE
po dbms_aqadm.aq$_purge_options_t;
BEGIN
   po.block := TRUE;
   DBMS_AQADM.PURGE_QUEUE_TABLE(
     queue_table     => 'test.obj_qtab',
     purge_condition => 'qtview.queue = ''TEST.OBJ_QUEUE''',
     purge_options   => po);
END;
/
```

Example 12-19 Purging All PROCESSED Messages in a Named Queue

```
DECLARE
po dbms_aqadm.aq$_purge_options_t;
BEGIN
   po.block := TRUE;
   DBMS_AQADM.PURGE_QUEUE_TABLE(
     queue_table     => 'test.obj_qtab',
     purge_condition => 'qtview.queue = ''TEST.OBJ_QUEUE''
                         and qtview.msg_state = ''PROCESSED''',
     purge_options   => po);
END;
/
```

Example 12-20 Purging All Messages in a Named Queue and for a Named Consumer

```
DECLARE
po dbms_aqadm.aq$_purge_options_t;
BEGIN
   po.block := TRUE;
   DBMS_AQADM.PURGE_QUEUE_TABLE(
     queue_table     => 'test.multiconsumer_81_qtab',
     purge_condition => 'qtview.queue = ''TEST.MULTICONSUMER_81_QUEUE''
                         and qtview.consumer_name = ''PAYROLL_APP''',
     purge_options   => po);
END;
/
```

Example 12-21 Purging All Messages from a Named Sender

```
DECLARE
po dbms_aqadm.aq$_purge_options_t;
BEGIN
   po.block := TRUE;
   DBMS_AQADM.PURGE_QUEUE_TABLE(
     queue_table     => 'test.multiconsumer_81_qtab',
     purge_condition => 'qtview.sender_name = ''TEST.OBJ_QUEUE''',
     purge_options    => po);
END;
/
```

Migrating an AQ Queue Table

`DBMS_AQADM.MIGRATE_QUEUE_TABLE` migrates an AQ queue table from 8.0, 8.1, or 10.0 to 8.0, 8.1, or 10.0. Only the owner of the queue table can migrate it.

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

Note:

This procedure requires that the `EXECUTE` privilege on `DBMS_AQADM` be granted to the queue table owner, who is probably an ordinary queue user. If you do not want ordinary queue users to be able to create and drop queues and queue tables, add and delete subscribers, and so forth, then you must revoke the `EXECUTE` privilege as soon as the migration is done.

Note:

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

If a schema was created by an import of an export dump from a lower release or has Oracle Database Advanced Queuing queues upgraded from a lower release, then attempts to drop it with `DROP USER CASCADE` will fail with ORA-24005. To drop such schemas:

1. Event 10851 should be set to level 1.
2. Drop all tables of the form `AQ$_`*`queue_table_name`*`_NR` from the schema.
3. Turn off event 10851.
4. Drop the schema.

Example 12-22   Upgrading a Queue Table from 8.1-Compatible to 10.0-Compatible

```
BEGIN
   DBMS_AQADM.MIGRATE_QUEUE_TABLE (
           queue_table     => 'test.xml_qtab',
           compatible      => '10.0');
END;
/
```

Managing AQ Queues

These topics describe how to manage AQ queues.

Note:

Starting and stopping a TxEventQ queue use the same APIs as AQ queues.

- Creating a Queue
- Altering a Queue
- Starting a Queue
- Stopping a Queue
- Dropping a Queue

Creating an AQ Queue

`DBMS_AQADM.CREATE_QUEUE` creates an AQ queue.

```
DBMS_AQADM.CREATE_QUEUE(
   queue_name          IN       VARCHAR2,
   queue_table         IN       VARCHAR2,
   queue_type          IN       BINARY_INTEGER DEFAULT NORMAL_QUEUE,
   max_retries         IN       NUMBER         DEFAULT NULL,
   retry_delay         IN       NUMBER         DEFAULT 0,
   retention_time      IN       NUMBER         DEFAULT 0,
   dependency_tracking IN       BOOLEAN        DEFAULT FALSE,
   comment             IN       VARCHAR2       DEFAULT NULL,
```

It has the following parameters:

Parameter	Description
`queue_name`	This required parameter specifies the name of the new queue. Mixed case (upper and lower case together) queue names are supported if database compatibility is 10.0, but the names must be enclosed in double quote marks. So `abc.efg` means the schema is `ABC` and the name is `EFG`, but `"abc"."efg"` means the schema is `abc` and the name is `efg`. Starting from 12*c* Release 2 (12.2.), the maximum length of user-generated queue names is 122 bytes. If you attempt to create a queue with a longer name, error ORA-24019 results. Queue names generated by Oracle Database Advanced Queuing, such as those listed in "Other Tables and Views", cannot be longer than 128 characters.
`queue_table`	This required parameter specifies the queue table in which the queue is created.
`queue_type`	This parameter specifies what type of queue to create. The default `NORMAL_QUEUE` produces a normal queue. `EXCEPTION_QUEUE` produces an exception queue.
`max_retries`	This parameter limits the number of times a dequeue with the `REMOVE` mode can be attempted on a message. The maximum value of `max_retries` is 2**31 -1.

Parameter	Description
`retry_delay`	This parameter specifies the number of seconds after which this message is scheduled for processing again after an application rollback. The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if `max_retries` is set to 0. This parameter is supported for single-consumer queues and 8.1-style or higher multiconsumer queues but not for 8.0-style multiconsumer queues, which are deprecated in Oracle Database Advanced Queuing 10*g* Release 2 (10.2).
`retention_time`	This parameter specifies the number of seconds a message is retained in the queue table after being dequeued from the queue. When `retention_time` expires, messages are removed by the time manager process. `INFINITE` means the message is retained forever. The default is 0, no retention.
`dependency_tracking`	This parameter is reserved for future use. `FALSE` is the default. `TRUE` is not permitted in this release.
`comment`	This optional parameter is a user-specified description of the queue. This user comment is added to the queue catalog.

All queue names must be unique within a schema. Once a queue is created with `CREATE_QUEUE`, it can be enabled by calling `START_QUEUE`. By default, the queue is created with both enqueue and dequeue disabled. To view retained messages, you can either dequeue by message ID or use SQL. If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue is created, then a corresponding LDAP entry is also created.

The following examples (Example 12-23 through Example 12-30) use data structures created in Example 12-1 through Example 12-12.

Example 12-23   Creating a Queue for Messages of Object Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name          =>  'test.obj_queue',
      queue_table         =>  'test.obj_qtab');
END;
/
```

Example 12-24   Creating a Queue for Messages of RAW Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name           => 'test.raw_queue',
      queue_table          => 'test.raw_qtab');
END;
/
```

Example 12-25   Creating a Queue for Messages of LOB Type

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name           => 'test.lob_queue',
      queue_table          => 'test.lob_qtab');
END;
/
```

Example 12-26 Creating a Queue for Grouped Messages

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name          => 'test.group_queue',
      queue_table         => 'test.group_qtab');
END;
/
```

Example 12-27 Creating a Queue for Prioritized Messages

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name         => 'test.priority_queue',
      queue_table        => 'test.priority_qtab');
END;
/
```

Example 12-28 Creating a Queue for Prioritized Messages and Multiple Consumers

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name         => 'test.multiconsumer_queue',
      queue_table        => 'test.multiconsumer_qtab');
END;
/
```

Example 12-29 Creating a Queue to Demonstrate Propagation

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name        => 'test.another_queue',
      queue_table       => 'test.multiconsumer_qtab');
END;
/
```

Example 12-30 Creating an 8.1-Style Queue for Multiple Consumers

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE(
      queue_name         => 'test.multiconsumer_81_queue',
      queue_table        => 'test.multiconsumer_81_qtab');
END;
/
```

Altering an AQ Queue

`DBMS_AQADM.ALTER_QUEUE` alters existing properties of an AQ queue.

```
DBMS_AQADM.ALTER_QUEUE(
   queue_name        IN    VARCHAR2,
   max_retries       IN    NUMBER   DEFAULT NULL,
   retry_delay       IN    NUMBER   DEFAULT NULL,
   retention_time    IN    NUMBER   DEFAULT NULL,
   comment           IN    VARCHAR2 DEFAULT NULL);
```

Only `max_retries`, `comment`, `retry_delay`, and `retention_time` can be altered. To view retained messages, you can either dequeue by message ID or use SQL. If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue is modified, then a corresponding LDAP entry is also altered.

The following example changes retention time, saving messages for 1 day after dequeuing:

Example 12-31 Altering a Queue by Changing Retention Time

```
BEGIN
   DBMS_AQADM.ALTER_QUEUE(
      queue_name        => 'test.another_queue',
      retention_time    => 86400);
END;
/
```

Starting an AQ Queue

`DBMS_AQADM.START_QUEUE` enables the specified AQ queue for enqueuing or dequeuing.

```
DBMS_AQADM.START_QUEUE(
   queue_name       IN     VARCHAR2,
   enqueue          IN     BOOLEAN DEFAULT TRUE,
   dequeue          IN     BOOLEAN DEFAULT TRUE);
```

After creating a queue, the administrator must use `START_QUEUE` to enable the queue. The default is to enable it for both enqueue and dequeue. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Example 12-32 Starting a Queue with Both Enqueue and Dequeue Enabled

```
BEGIN
   DBMS_AQADM.START_QUEUE (
      queue_name         => 'test.obj_queue');
END;
/
```

Example 12-33 Starting a Queue for Dequeue Only

```
BEGIN
   DBMS_AQADM.START_QUEUE(
      queue_name         => 'test.raw_queue',
      dequeue            => TRUE,
      enqueue            => FALSE);
END;
/
```

Stopping an AQ Queue

`DBMS_AQADM.STOP_QUEUE` disables enqueuing, dequeuing, or both on the specified AQ queue.

```
DBMS_AQADM.STOP_QUEUE(
   queue_name       IN   VARCHAR2,
   enqueue          IN   BOOLEAN DEFAULT TRUE,
   dequeue          IN   BOOLEAN DEFAULT TRUE,
   wait             IN   BOOLEAN DEFAULT TRUE);
```

By default, this call disables both enqueue and dequeue. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Example 12-34 Stopping a Queue

```
BEGIN
   DBMS_AQADM.STOP_QUEUE(
      queue_name        => 'test.obj_queue');
```

```
END;
/
```

Dropping an AQ Queue

This procedure drops an existing AQ 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.

```
DBMS_AQADM.DROP_QUEUE(
   queue_name        IN    VARCHAR2,
```

If `GLOBAL_TOPIC_ENABLED = TRUE` when a queue is dropped, then a corresponding LDAP entry is also dropped.

Example 12-35   Dropping a Standard Queue

```
BEGIN
   DBMS_AQADM.DROP_QUEUE(
      queue_name         => 'test.obj_queue');
END;
/
```

Managing Transformations

Transformations change the format of a message, so that a message created by one application can be understood by another application. You can use transformations on both persistent and buffered messages. These topics describe how to manage queue tables.

- Creating a Transformation
- Modifying a Transformation
- Dropping a Transformation

Note:

TxEventQ queues do not support transformations.

Creating a Transformation

`DBMS_TRANSFORM.CREATE_TRANSFORMATION` creates a message format transformation.

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema                 VARCHAR2(30),
    name                   VARCHAR2(30),
    from_schema            VARCHAR2(30),
    from_type              VARCHAR2(30),
    to_schema              VARCHAR2(30),
    to_type                VARCHAR2(30),
    transformation         VARCHAR2(4000));
```

The transformation must be a SQL function with input type `from_type`, returning an object of type `to_type`. It can also be a SQL expression of type `to_type`, referring to `from_type`. All references to `from_type` must be of the form `source.user_data`.

You must be granted EXECUTE privilege on dbms_transform to use this feature. This privilege is included in the AQ_ADMINISTRATOR_ROLE.

You must also have EXECUTE privilege on the user-defined types that are the source and destination types of the transformation, and have EXECUTE privileges on any PL/SQL function being used in the transformation function. The transformation cannot write the database state (that is, perform DML operations) or commit or rollback the current transaction.

Example 12-36 Creating a Transformation

```
BEGIN
   DBMS_TRANSFORM.CREATE_TRANSFORMATION(
      schema         => 'test',
      name           => 'message_order_transform',
      from_schema    => 'test',
      from_type      => 'message_typ',
      to_schema      => 'test',
      to_type        => 'order_typ',
      transformation => 'test.order_typ(
         source.user_data.sender_id,
         source.user_data.subject,
         source.user_data.text)');
END;
/
```

See Also:

"Oracle Database Advanced Queuing Security" for more information on administrator and user roles

Modifying a Transformation

DBMS_TRANSFORM.MODIFY_TRANSFORMATION changes the transformation function and specifies transformations for each attribute of the target type.

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION(
    schema            VARCHAR2(30),
    name              VARCHAR2(30),
    attribute_number  INTEGER,
    transformation    VARCHAR2(4000));
```

If the attribute number 0 is specified, then the transformation expression singularly defines the transformation from the source to target types.

All references to from_type must be of the form source.user_data. All references to the attributes of the source type must be prefixed by source.user_data.

You must be granted EXECUTE privileges on dbms_transform to use this feature. You must also have EXECUTE privileges on the user-defined types that are the source and destination types of the transformation, and have EXECUTE privileges on any PL/SQL function being used in the transformation function.

Dropping a Transformation

DBMS_TRANSFORM.DROP_TRANSFORMATION drops a transformation.

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
    schema      VARCHAR2(30),
    name        VARCHAR2(30));
```

You must be granted EXECUTE privileges on dbms_transform to use this feature. You must also have EXECUTE privileges on the user-defined types that are the source and destination types of the transformation, and have EXECUTE privileges on any PL/SQL function being used in the transformation function.

Granting and Revoking Privileges

These topics describe how to grant and revoke privileges.

- Granting Oracle Database Advanced Queuing System Privileges
- Revoking Oracle Database Advanced Queuing System Privileges
- Granting Queue Privileges
- Revoking Queue Privileges

Granting Oracle Database Advanced Queuing System Privileges

DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE grants Oracle Database Advanced Queuing 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.

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege         IN    VARCHAR2,
   grantee           IN    VARCHAR2,
   admin_option      IN    BOOLEAN := FALSE);
```

Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to dequeue messages from any queues in the database. Users granted the MANAGE_ANY privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

Note:

Starting from Oracle Database 12*c* Release 2, MANAGE_ANY, ENQUEUE_ANY, and DEQUEUE_ANY privileges will not allow access to SYS owned queues by users other than SYS.

Example 12-37 Granting AQ System Privileges

```
BEGIN
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
      privilege          =>    'ENQUEUE_ANY',
      grantee            =>    'test',
      admin_option       =>     FALSE);
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
      privilege          =>     'DEQUEUE_ANY',
      grantee            =>     'test',
      admin_option       =>      FALSE);
END;
/
```

Revoking Oracle Database Advanced Queuing System Privileges

DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE revokes Oracle Database Advanced Queuing system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY.

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(
   privilege         IN   VARCHAR2,
   grantee           IN   VARCHAR2);
```

The ADMIN option for a system privilege cannot be selectively revoked.

Users granted the ENQUEUE_ANY privilege are allowed to enqueue messages to any queues in the database. Users granted the DEQUEUE_ANY privilege are allowed to dequeue messages from any queues in the database. Users granted the MANAGE_ANY privilege are allowed to run DBMS_AQADM calls on any schemas in the database.

Note:

Starting from Oracle Database 12*c* Release 2, MANAGE_ANY, ENQUEUE_ANY, and DEQUEUE_ANY privileges will not allow access to SYS owned queues by users other than SYS.

Example 12-38   Revoking AQ System Privileges

```
BEGIN
   DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE(
      privilege          =>    'DEQUEUE_ANY',
      grantee            =>    'test');
END;
/
```

Granting Queue Privileges

DBMS_AQADM.GRANT_QUEUE_PRIVILEGE grants privileges on a queue to users and roles. The privileges are ENQUEUE, DEQUEUE, or ALL. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(
   privilege        IN    VARCHAR2,
   queue_name       IN    VARCHAR2,
   grantee          IN    VARCHAR2,
   grant_option     IN    BOOLEAN := FALSE);
```

Note:

This procedure requires that `EXECUTE` privileges on `DBMS_AQADM` be granted to the queue table owner, who is probably an ordinary queue user. If you do not want ordinary queue users to be able to create and drop queues and queue tables, add and delete subscribers, and so forth, then you must revoke the `EXECUTE` privilege as soon as the initial `GRANT_QUEUE_PRIVILEGE` is done.

Example 12-39   Granting Queue Privilege

```
BEGIN
   DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
      privilege     =>     'ALL',
      queue_name    =>     'test.multiconsumer_81_queue',
      grantee       =>     'test_adm',
      grant_option  =>      TRUE);
END;
/
```

Revoking Queue Privileges

`DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE` revokes privileges on a queue from users and roles. The privileges are `ENQUEUE` or `DEQUEUE`.

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege         IN      VARCHAR2,
   queue_name        IN      VARCHAR2,
   grantee           IN      VARCHAR2);
```

To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the `GRANT` option are revoked if the grantor's privileges are revoked.

You can revoke the dequeue right of a grantee on a specific queue, leaving the grantee with only the enqueue right as in Example 12-40.

Example 12-40   Revoking Dequeue Privilege

```
BEGIN
   DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE(
      privilege     =>     'DEQUEUE',
      queue_name    =>     'test.multiconsumer_81_queue',
      grantee       =>     'test_adm');
END;
```

Managing Subscribers

These topics describe how to manage subscribers.

- Adding a Subscriber
- Altering a Subscriber
- Removing a Subscriber

Adding a Subscriber

`DBMS_AQADM.ADD_SUBSCRIBER` adds a default subscriber to a queue.

```
DBMS_AQADM.ADD_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2 DEFAULT NULL,
   transformation IN    VARCHAR2 DEFAULT NULL,
   queue_to_queue IN    BOOLEAN DEFAULT FALSE,
   delivery_mode  IN    PLS_INTEGER DEFAULT PERSISTENT);
```

An application can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation succeeds only on queues that allow multiple consumers, and the total number of subscribers must be 1024 or less. This operation takes effect immediately and the containing transaction is committed. Enqueue requests that are executed after the completion of this call reflect the new action. Any string within the `rule` must be quoted (with single quotation marks) as follows:

```
rule   => 'PRIORITY <= 3 AND CORRID =  ''FROM JAPAN'''
```

User data properties or attributes apply only to object payloads and must be prefixed with `tab.userdata` in all cases.

If `GLOBAL_TOPIC_ENABLED` is set to true when a subscriber is created, then a corresponding LDAP entry is also created.

Specify the name of the transformation to be applied during dequeue or propagation. The transformation must be created using the `DBMS_TRANSFORM` package.

For queues that contain payloads with XMLType attributes, you can specify rules that contain operators such as `XMLType.existsNode()` and `XMLType.extract()`.

If parameter `queue_to_queue` is set to `TRUE`, then the added subscriber is a queue-to-queue subscriber. When queue-to-queue propagation is set up between a source queue and a destination queue, queue-to-queue subscribers receive messages through that propagation schedule.

If the `delivery_mode` parameter is the default `PERSISTENT`, then the subscriber receives only persistent messages. If it is set to `BUFFERED`, then the subscriber receives only buffered messages. If it is set to `PERSISTENT_OR_BUFFERED`, then the subscriber receives both types. You cannot alter this parameter with `ALTER_SUBSCRIBER`.

The agent name should be `NULL` if the destination queue is a single consumer queue.

Note:

`ADD_SUBSCRIBER` is an administrative operation on a queue. Although Oracle Database AQ does not prevent applications from issuing administrative and operational calls concurrently, they are executed serially. `ADD_SUBSCRIBER` blocks until pending calls that are enqueuing or dequeuing messages complete. It will not wait for the pending transactions to complete.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_TRANSFORM` package
- "Scheduling a Queue Propagation"

Example 12-41 Adding a Subscriber at a Designated Queue at a Database Link

```
DECLARE
   subscriber          sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber1', 'test2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name         => 'test.multiconsumer_81_queue',
      subscriber         =>  subscriber);
END;
/
```

Example 12-42 Adding a Single Consumer Queue at a Dababase Link as a Subscriber

```
DECLARE
   subscriber          sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber1', 'test2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name         => 'test.multiconsumer_81_queue',
      subscriber         =>  subscriber);
END;
/
```

Example 12-43 Adding a Subscriber with a Rule

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber2', 'test2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name =>  'test.multiconsumer_81_queue',
      subscriber =>   subscriber,
      rule       =>  'priority < 2');
END;
/
```

Example 12-44 Adding a Subscriber and Specifying a Transformation

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber3', 'test2.msg_queue2@london', null);
   DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name     => 'test.multiconsumer_81_queue',
      subscriber     =>  subscriber,
      transformation => 'test.message_order_transform');
END;
/
```

Example 12-45 Propagating from a Multiple-Consumer Queue to a Single Consumer Queue

```
DECLARE
   subscriber          SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT(NULL, 'test2.single_consumer__queue@london',
null);
  DBMS_AQADM.ADD_SUBSCRIBER(
      queue_name         => 'test.multiconsumer_81_queue',
      subscriber         =>  subscriber);
END;
```

Altering a Subscriber

`DBMS_AQADM.ALTER_SUBSCRIBER` alters existing properties of a subscriber to a specified queue.

```
DBMS_AQADM.ALTER_SUBSCRIBER (
   queue_name     IN    VARCHAR2,
   subscriber     IN    sys.aq$_agent,
   rule           IN    VARCHAR2
   transformation IN    VARCHAR2);
```

The rule, the transformation, or both can be altered. If you alter only one of these attributes, then specify the existing value of the other attribute to the alter call. If GLOBAL_TOPIC_ENABLED = TRUE when a subscriber is modified, then a corresponding LDAP entry is created.

Example 12-46 Altering a Subscriber Rule

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent('subscriber2', 'test2.msg_queue2@london', null);
   DBMS_AQADM.ALTER_SUBSCRIBER(
      queue_name => 'test.multiconsumer_81_queue',
      subscriber =>  subscriber,
      rule       => 'priority = 1');
END;
/
```

Removing a Subscriber

`DBMS_AQADM.REMOVE_SUBSCRIBER` removes a default subscriber from a queue.

```
DBMS_AQADM.REMOVE_SUBSCRIBER (
   queue_name         IN         VARCHAR2,
   subscriber         IN         sys.aq$_agent);
```

This operation takes effect immediately and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation. If `GLOBAL_TOPIC_ENABLED = TRUE` when a subscriber is dropped, then a corresponding LDAP entry is also dropped.

It is not an error to run the `REMOVE_SUBSCRIBER` procedure even when there are pending messages that are available for dequeue by the consumer. These messages are automatically made unavailable for dequeue when the `REMOVE_SUBSCRIBER` procedure finishes.

Note:

REMOVE_SUBSCRIBER is an administrative operation on a queue. Although Oracle Database AQ does not prevent applications from issuing administrative and operational calls concurrently, they are executed serially. REMOVE_SUBSCRIBER blocks until pending calls that are enqueuing or dequeuing messages complete. It will not wait for the pending transactions to complete.

Example 12-47 Removing a Subscriber

```
DECLARE
   subscriber       sys.aq$_agent;
BEGIN
   subscriber := sys.aq$_agent ('subscriber2', 'test2.msg_queue2@london', null);
   DBMS_AQADM.REMOVE_SUBSCRIBER(
      queue_name => 'test.multiconsumer_81_queue',
      subscriber => subscriber);
END;
/
```

Managing Propagations

The propagation schedules defined for a queue can be changed or dropped at any time during the life of the queue.

You can also temporarily disable a schedule instead of dropping it. All administrative calls can be made irrespective of whether the schedule is active or not. If a schedule is active, then it takes a few seconds for the calls to be processed.

These topics describe how to manage propagations.

- Scheduling a Queue Propagation
- Verifying Propagation Queue Type
- Altering a Propagation Schedule
- Enabling a Propagation Schedule
- Disabling a Propagation Schedule
- Unscheduling a Queue Propagation

Scheduling a Queue Propagation

DBMS_AQADM.SCHEDULE_PROPAGATION schedules propagation of messages.

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
   queue_name          IN  VARCHAR2,
   destination         IN  VARCHAR2 DEFAULT NULL,
   start_time          IN  DATE     DEFAULT SYSDATE,
   duration            IN  NUMBER   DEFAULT NULL,
   next_time           IN  VARCHAR2 DEFAULT NULL,
   latency             IN  NUMBER   DEFAULT 60,
   destination_queue   IN  VARCHAR2 DEFAULT NULL);
```

The destination can be identified by a database link in the destination parameter, a queue name in the destination_queue parameter, or both. Specifying only a database link results in queue-to-dblink propagation. If you propagate messages to several queues in another database, then all propagations have the same frequency.

If a private database link in the schema of the queue table owner has the same name as a public database link, AQ always uses the private database link.

Specifying the destination queue name results in queue-to-queue propagation. If you propagate messages to several queues in another database, queue-to-queue propagation enables you to configure each schedule independently of the others. You can enable or disable individual propagations.

> **Note:**
>
> If you want queue-to-queue propagation to a queue in another database, then you must specify parameters `destination` and `destination_queue`.

Queue-to-queue propagation mode supports transparent failover when propagating to a destination Oracle Real Application Clusters (Oracle RAC) system. With queue-to-queue propagation, it is not required to repoint a database link if the owner instance of the queue fails on Oracle RAC.

Messages can also be propagated to other queues in the same database by specifying a `NULL` destination. If a message has multiple recipients at the same destination in either the same or different queues, then the message is propagated to all of them at the same time.

The source queue must be in a queue table meant for multiple consumers. If you specify a single-consumer queue, than error ORA-24039 results. Oracle Database Advanced Queuing does not support the use of synonyms to refer to queues or database links.

If you specify a propagation `next_time` and `duration`, propagation will run periodically for the specified duration.If you specify a latency of zero with no `next_time` or `duration`, the resulting propagation will run forever, propagating messages as they appear in the queue, and idling otherwise. If a non-zero latency is specified, with no `next_time` or `duration` (default), the propagation schedule will be event-based. It will be scheduled to run when there are messages in the queue to be propagated. When there are no more messages for a system-defined period of time, the job will stop running until there are new messages to be propagated.The time at which the job runs depends on other factors, such as the number of ready jobs and the number of job queue processes.

Propagation uses a linear backoff scheme for retrying propagation from a schedule that encountered a failure. If a schedule continuously encounters failures, then the first retry happens after 30 seconds, the second after 60 seconds, the third after 120 seconds and so forth. If the retry time is beyond the expiration time of the current window, then the next retry is attempted at the start time of the next window. A maximum of 16 retry attempts are made after which the schedule is automatically disabled.

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

If you specify a value for `destination` that does not exist, then this procedure still runs without throwing an error. You can query runtime propagation errors in the `LAST_ERROR_MSG` column of the `USER_QUEUE_SCHEDULES` view.

See Also:

- "Managing Job Queues" in *Oracle Database Administrator's Guide* for more information on job queues and Jnnn background processes
- Internet Access to Oracle Database Advanced Queuing
- "USER_QUEUE_SCHEDULES: Propagation Schedules in User Schema"

Example 12-48 Scheduling a Propagation to Queues in the Same Database

```
BEGIN
   DBMS_AQADM.SCHEDULE_PROPAGATION(
      queue_name    =>    'test.multiconsumer_queue');
END;
/
```

Example 12-49 Scheduling a Propagation to Queues in Another Database

```
BEGIN
   DBMS_AQADM.SCHEDULE_PROPAGATION(
      queue_name    =>    'test.multiconsumer_queue',
      destination   =>    'another_db.world');
END;
/
```

Example 12-50 Scheduling Queue-to-Queue Propagation

```
BEGIN
   DBMS_AQADM.SCHEDULE_PROPAGATION(
      queue_name         =>    'test.multiconsumer_queue',
      destination        =>    'another_db.world'
      destination_queue  =>    'target_queue');
END;
/
```

Verifying Propagation Queue Type

`DBMS_AQADM.VERIFY_QUEUE_TYPES` verifies that the source and destination queues have identical types. The result of the verification is stored in the dictionary table `SYS.AQ$_MESSAGE_TYPES`, overwriting all previous output of this command.

```
DBMS_AQADM.VERIFY_QUEUE_TYPES(
   src_queue_name     IN    VARCHAR2,
   dest_queue_name    IN    VARCHAR2,
   destination        IN    VARCHAR2 DEFAULT NULL,
   rc                 OUT   BINARY_INTEGER);
```

If the source and destination queues do not have identical types and a transformation was specified, then the transformation must map the source queue type to the destination queue type.

Note:

- `SYS.AQ$_MESSAGE_TYPES` can have multiple entries for the same source queue, destination queue, and database link, but with different transformations.
- `VERIFY_QUEUE_TYPES` check happens once per AQ propagation schedule and *not* for every propagated message send
- In case the payload of the queue is modified then the existing propagation schedule between source and destination queue needs to be dropped and recreated.

Example 12-51 involves two queues of the same type. It returns:

```
VQT: new style queue
Compatible: 1
```

If the same example is run with `test.raw_queue` (a queue of type RAW) in place of `test.another_queue`, then it returns:

```
VQT: new style queue
Compatible: 0
```

Example 12-51 Verifying a Queue Type

```
SET SERVEROUTPUT ON
DECLARE
rc      BINARY_INTEGER;
BEGIN
   DBMS_AQADM.VERIFY_QUEUE_TYPES(
      src_queue_name  => 'test.multiconsumer_queue',
      dest_queue_name => 'test.another_queue',
      rc              =>  rc);
   DBMS_OUTPUT.PUT_LINE('Compatible: '||rc);
END;
/
```

Altering a Propagation Schedule

`DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE` alters parameters for a propagation schedule. The `destination_queue` parameter for queue-to-queue propagation cannot be altered.

```
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name        IN  VARCHAR2,
   destination       IN  VARCHAR2 DEFAULT NULL,
   duration          IN  NUMBER   DEFAULT NULL,
   next_time         IN  VARCHAR2 DEFAULT NULL,
   latency           IN  NUMBER   DEFAULT 60,
   destination_queue IN  VARCHAR2 DEFAULT NULL);
```

Example 12-52 Altering a Propagation Schedule to Queues in the Same Database

```
BEGIN
   DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
      queue_name    =>    'test.multiconsumer_queue',
      duration      =>    '2000',
      next_time     =>    'SYSDATE + 3600/86400',
      latency       =>    '32');
```

```
END;
/
```

Example 12-53   Altering a Propagation Schedule to Queues in Another Database

```
BEGIN
   DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
      queue_name     =>    'test.multiconsumer_queue',
      destination    =>    'another_db.world',
      duration       =>    '2000',
      next_time      =>    'SYSDATE + 3600/86400',
      latency        =>    '32');
END;
/
```

Enabling a Propagation Schedule

`DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE` enables a previously disabled propagation schedule.

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   queue_name        IN   VARCHAR2,
   destination       IN   VARCHAR2 DEFAULT NULL,
   destination_queue IN   VARCHAR2 DEFAULT NULL);
```

Example 12-54   Enabling a Propagation to Queues in the Same Database

```
BEGIN
   DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
      queue_name    =>   'test.multiconsumer_queue');
END;
/
```

Example 12-55   Enabling a Propagation to Queues in Another Database

```
BEGIN
   DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
      queue_name     =>    'test.multiconsumer_queue',
      destination    =>    'another_db.world');
END;
/
```

Disabling a Propagation Schedule

`DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE` disables a previously enabled propagation schedule.

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   queue_name        IN   VARCHAR2,
   destination       IN   VARCHAR2 DEFAULT NULL,
   destination_queue IN   VARCHAR2 DEFAULT NULL);
```

Example 12-56   Disabling a Propagation to Queues in the Same Database

```
BEGIN
   DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
      queue_name    =>   'test.multiconsumer_queue');
END;
/
```

Example 12-57 Disabling a Propagation to Queues in Another Database

```
BEGIN
   DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
      queue_name    =>    'test.multiconsumer_queue',
      destination   =>    'another_db.world');
END;
/
```

Unscheduling a Queue Propagation

`DBMS_AQADM.UNSCHEDULE_PROPAGATION` unschedules a previously scheduled propagation of messages from a queue to a destination. The destination is identified by a specific database link in the `destination` parameter or by name in the `destination_queue` parameter.

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
   queue_name        IN  VARCHAR2,
   destination       IN  VARCHAR2 DEFAULT NULL,
   destination_queue IN  VARCHAR2 DEFAULT NULL);
```

Example 12-58 Unscheduling a Propagation to Queues in the Same Database

```
BEGIN
   DBMS_AQADM.UNSCHEDULE_PROPAGATION(
      queue_name => 'test.multiconsumer_queue');
END;
/
```

Example 12-59 Unscheduling a Propagation to Queues in Another Database

```
BEGIN
   DBMS_AQADM.UNSCHEDULE_PROPAGATION(
      queue_name    =>   'test.multiconsumer_queue',
      destination   =>   'another_db.world');
END;
/
```

Managing Oracle Database Advanced Queuing Agents

These topics describe how to manage Oracle Database Advanced Queuing Agents.

- Creating an Oracle Database Advanced Queuing Agent
- Altering an Oracle Database Advanced Queuing Agent
- Dropping an Oracle Database Advanced Queuing Agent
- Enabling Database Access
- Disabling Database Access

Creating an Oracle Database Advanced Queuing Agent

`DBMS_AQADM.CREATE_AQ_AGENT` registers an agent for Oracle Database Advanced Queuing Internet access using HTTP protocols.

```
DBMS_AQADM.CREATE_AQ_AGENT (
  agent_name              IN VARCHAR2,
  certificate_location    IN VARCHAR2 DEFAULT NULL,
  enable_http             IN BOOLEAN DEFAULT FALSE,
  enable_anyp             IN BOOLEAN DEFAULT FALSE);
```

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Database Advanced Queuing Internet agents. When an agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- GLOBAL_TOPIC_ENABLED = TRUE
- certificate_location is specified

Altering an Oracle Database Advanced Queuing Agent

DBMS_AQADM.ALTER_AQ_AGENT alters an agent registered for Oracle Database Advanced Queuing Internet access.

```
DBMS_AQADM.ALTER_AQ_AGENT (
  agent_name                IN VARCHAR2,
  certificate_location      IN VARCHAR2 DEFAULT NULL,
  enable_http               IN BOOLEAN DEFAULT FALSE,
  enable_anyp               IN BOOLEAN DEFAULT FALSE);
```

When an Oracle Database Advanced Queuing agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- GLOBAL_TOPIC_ENABLED = TRUE
- certificate_location is specified

Dropping an Oracle Database Advanced Queuing Agent

DBMS_AQADM.DROP_AQ_AGENT drops an agent that was previously registered for Oracle Database Advanced Queuing Internet access.

```
DBMS_AQADM.DROP_AQ_AGENT (
  agent_name     IN VARCHAR2);
```

When an Oracle Database Advanced Queuing agent is created, altered, or dropped, an LDAP entry is created for the agent if the following are true:

- GLOBAL_TOPIC_ENABLED = TRUE
- certificate_location is specified

Enabling Database Access

DBMS_AQADM.ENABLE_DB_ACCESS grants an Oracle Database Advanced Queuing Internet agent the privileges of a specific database user. The agent should have been previously created using the CREATE_AQ_AGENT procedure.

```
DBMS_AQADM.ENABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

The SYS.AQ$INTERNET_USERS view has a list of all Oracle Database Advanced Queuing Internet agents and the names of the database users whose privileges are granted to them.

Disabling Database Access

`DBMS_AQADM.DISABLE_DB_ACCESS` revokes the privileges of a specific database user from an Oracle Database Advanced Queuing Internet agent. The agent should have been previously granted those privileges using the `ENABLE_DB_ACCESS` procedure.

```
DBMS_AQADM.DISABLE_DB_ACCESS (
  agent_name                IN VARCHAR2,
  db_username               IN VARCHAR2)
```

Adding an Alias to the LDAP Server

`DBMS_AQADM.ADD_ALIAS_TO_LDAP` adds an alias to the LDAP server.

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP(
   alias          IN VARCHAR2,
   obj_location   IN VARCHAR2);
```

This call takes the name of an alias and the distinguished name of an Oracle Database Advanced Queuing object in LDAP, and creates the alias that points to the Oracle Database Advanced Queuing object. The alias is placed immediately under the distinguished name of the database server. The object to which the alias points can be a queue, an agent, or a ConnectionFactory.

Deleting an Alias from the LDAP Server

`DBMS_AQADM.DEL_ALIAS_FROM_LDAP` removes an alias from the LDAP server.

```
DBMS_AQADM.DEL_ALIAS_FROM_LDAP(
   alias IN VARCHAR2);
```

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.

A

Nonpersistent Queues

Oracle Database 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). The messages are stored in a system-created queue table. Oracle Database Advanced Queuing supports persistent and nonpersistent messages with a common API.

Nonpersistent queues, which can be either single-consumer or multiconsumer, provide a mechanism for notification to all currently connected users. Subscribers can be added to multiconsumer nonpersistent queues, and nonpersistent queues can be destinations for propagation.

You use the enqueue interface to enqueue messages into a nonpersistent queue in the usual way. You can enqueue RAW and Oracle object type messages into a nonpersistent queue. OCI notifications are used to deliver such messages to users that are currently registered for notification.

The following topics describe nonpersistent queues, which are deprecated in Oracle Database Advanced Queuing 10*g* Release 2 (10.2). Oracle recommends that you use buffered messaging instead.

See Also:

"Buffered Messaging"

Topics:

- Creating Nonpersistent Queues
- Managing Nonpersistent Queues
- Compatibility of Nonpersistent Queues
- Nonpersistent Queue Notification
- Restrictions on Nonpersistent Queues

Creating Nonpersistent Queues

```
DBMS_AQADM.CREATE_NP_QUEUE (
   queue_name             IN        VARCHAR2,
   multiple_consumers     IN        BOOLEAN   DEFAULT FALSE,
   comment                IN        VARCHAR2  DEFAULT NULL);
```

This procedure creates a nonpersistent queue.

Only local recipients are supported for nonpersistent queues. The queue can be either single-consumer or multiconsumer. All queue names must be unique within a schema. The queues are created in an 8.1-compatible system-created queue table (`AQ$_MEM_SC` or `AQ$_MEM_MC`) in the same schema as that specified by the queue name. If the queue name does not specify a schema name, then the queue is created in the login user's schema.

Note:

Names of nonpersistent queues must not be longer than 24 characters. If you attempt to create a nonpersistent queue with a longer name, error ORA-24019 results.

Managing Nonpersistent Queues

Once a queue is created with `CREATE_NP_QUEUE`, it can be enabled by calling `START_QUEUE`. By default, the queue is created with both enqueue and dequeue disabled.

You can enqueue RAW and Oracle object type messages into a nonpersistent queue. You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the Oracle Call Interface (OCI) notification mechanism. You cannot invoke the `listen` call on a nonpersistent queue.

A nonpersistent queue can be dropped only by its owner.

Compatibility of Nonpersistent Queues

For 8.1-style or higher queues, the `compatible` parameter of `init.ora` and the `compatible` parameter of the queue table should be set to 8.1 or higher to use nonpersistent queues.

Nonpersistent Queue Notification

For nonpersistent queues, the message is delivered as part of the notification. Table A-1 shows the actions performed for nonpersistent queues for different notification mechanisms when RAW presentation is specified. Table A-2 shows the actions performed when XML presentation is specified.

Table A-1 Actions Performed for Nonpersistent Queues When RAW Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
RAW	OCI callback receives the RAW data in the payload.	Not supported	PL/SQL callback receives the RAW data in the payload.
Oracle object type	Not supported	Not supported	Not supported

Table A-2 Actions Performed for Nonpersistent Queues When XML Presentation Specified

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
RAW	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.
Oracle object type	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.

Restrictions on Nonpersistent Queues

You can create nonpersistent queues of `RAW` and Oracle object type.You are limited to sending messages only to subscribers and explicitly specified recipients who are local. Propagation is not supported from nonpersistent queues. When retrieving messages, you cannot use the dequeue call, but must instead employ the asynchronous notification mechanism, registering for the notification by mean of `OCISubscriptionRegister`.

The `visibility` attribute of `enqueue_options` must be set to `IMMEDIATE` for nonpersistent messages.

See Also:

"Enqueue Options"

B Oracle JMS and Oracle AQ XML Servlet Error Messages

A list of error messages is provided to aid you in troubleshooting problems.

- Oracle JMS Error Messages
- Oracle AQ XML Servlet Error Messages

Oracle 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-207 JMS Exception (string)
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.

Action: 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

JMS-222: Invalid Payload factory type
Cause: Payload factory should be of CustomDatumFactory or ORADataFactory type.

Action: Use one of CustomDatumFactory/ORADataFactory types.

JMS-223: Payload factory must be null for destinations with Sys.AnyData payload - use typemap instead
Cause: A CustomDatumFactory/ORADataFactory was specifed for consumers on destinations containing SYS.ANYDATA payloads

Action: This field must be set to null for destinations containing payloads of type SYS.AnyData. The ORADataFactory(s) must be registered in the session's typemap

JMS-224: Typemap is invalid - must be populated with SQLType/OraDataFactory mappings to receive messages from Sys.AnyData destinations
Cause: The typemap of the session is null or empty. For destinations of tpye Sys.AnyData, the typemap must contain OraDataFactory objects for all possible types that may be stored in the queue/topic

Action: Use the AQjmsSession.getTypeMap() method get the typemap. Then use the put() method to register SQLType names and their corresponding OraDataFactory objects

JMS-225: Invalid JDBC driver - OCI driver must be used for this operation
Cause: Operations on Sys.AnyData queues/topics are not supported using this JDBC driver

Action: Use the JDBC OCI driver to receive messages from Sys.AnyData queues/topics

JMS-226: Header-only Message does not have a body
Cause: Header-only message does not have a body; the body cannot be cleared, changed, or queried.

Action: Do not access or modify the body of a read-only message.

JMS-227: Illegal attempt to commit on a non transacted JMS Session
Cause: Illegal attempt to invoke the commit method on a non transacted JMS Session

Action: Remove invocation of the commit method on the current JMS Session

JMS-228: Illegal attempt to rollback on a non transacted JMS Session
Cause: Illegal attempt to invoke the rollback method on a non transacted JMS Session

Action: Remove invocation of the rollback method on the current JMS Session

JMS-229: (string) must be specified
Cause: The specified parameter was null

Action: Specify a non-null value for the parameter

JMS-230: Illegal operation on durable subscription with active TopicSubscriber
Cause: Try to unsubscribe, change or create durable subscription while it has an active TopicSubscriber.

Action: Close the active TopicSubscriber and try again.

JMS-231: Consumers on temporary destination must belong to the same
Cause: The specified parameter was null

Action: Specify a non-null value for the parameter

JMS-232: An invalid user/password was specified for the JMS connection
Cause: Invalid user/password specified for connection

Action: Specify valid user/password for connection

JMS-233: The required subscriber information is not available
Cause: Try to retrieve subscriber information from a subscriber which was obtained with getDurableSubscriber method and had no corresponding information in the in-memory map.

Action: Do not retrieve subscriber information from such a subscriber

JMS-234: This operation is not allowed in the current messaging domain
Cause: Perform an operation that is not allowed in the current messaging domain

Action: Do not perform the operation in the current messaging domain

JMS-235: Can not link the durable subscriber name with a topic in unsubscribe method.
Cause: OJMS is unable to link the supplied durable subscriber name with a topic. This causes unsubscribe method to fail.

Action: Use other unsubscribe methods which the user can provide topic name.

JMS-236: OJMS encountered invalid OCI handles.
Cause: OJMS encountered invalid OCI handles when using JDBC OCI driver .

Action: Make sure the underlying JDBC connection is not closed while OJMS is still operating.

JMS-237: Can not start thread for message listener.
Cause: OJMS is unable to start a new thread for message listener.

Action: Check the thread proxy code if the thread was started by a user provided thread proxy.

JMS-238: Illegal attempt to recover on a transacted JMS Session
Cause: Illegal attempt to invoke the recover method on a transacted JMS Session

Action: Use rollback method instead of recover method on transacted JMS session.

JMS-239: Illegal attempt to call (string) method on a XASession.
Cause: Illegal attempt to invoke the commit or rollback method on a XASession

Action: Use JTA to commit or rollback a distributed transaction

JMS-240: Illegal attempt to call setClientID after other actions.
Cause: Illegal attempt to call Connection.setClientID method after other actions has been taken on this Connection

Action: set client ID before any other action on the Connection

JMS-241: Illegal attempt to delete temporary destination when there are consumers using it.
Cause: Illegal attempt to delete temporary destination while there are existing consumers still using it.

Action: close the consumers before deleting the temporary destination

JMS-242: Illegal attempt to enqueue message with both immediate visibility and three phase enqueue process.
Cause: Illegal attempt to enqueue message with both immediate visibility and three phase enqueue process.

Action: turn on the system property oracle.jms.useTemplobsForEnqueue

JMS-243: Topic (string) not found}
Cause: The specified topic was not found

Action: Specify a valid topic

Oracle 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.

JMS-441: {0} cannot be null
Cause: The specified parameter was null

Action: Specify a non-null value

JMS-442: Name and Address for Agent cannot be null
Cause: Both the name and address parameters were specified as null

Action: Specify a non-null value for the name or address

JMS-443: IMMEDIATE visibility mode not supported for this queue/topic
Cause: IMMEDIATE visibility mode not supported for JMS type queue/topic

Action: Use ON_COMMIT or the default visibility mode for JMS type queue/topic

JMS-444: This feature is not supported yet
Cause: The requested feature is not yet supported

Action: wait for future releases that support the feature.

JMS-445: Destination alias must be specified
Cause: A null Destination alias was specified

Action: Specify a non-null destination alias

JMS-446: Agent alias must be specified
Cause: A null Agent alias was specified

Action: Specify a non-null agent alias

JMS-447: error in accessing LDAP server
Cause: error in accessing the LDAP server

Action: check the LDAP server is up and the environment parameters provided to the servlet are correct

JMS-448: Invalid Content-Type
Cause: Invalid Content-Type

Action: Content-Type must be "text/xml" or "application/x-www-form-urlencoded" with parameter name "aqxmldoc"

C

Oracle Messaging Gateway

Note:

Oracle Messaging Gateway is deprecated in 23ai and will be desupported in a future release.

- Introduction to Oracle Messaging Gateway
- Getting Started with Oracle Messaging Gateway
- Working with Oracle Messaging Gateway
- Oracle Messaging Gateway Message Conversion
- Monitoring Oracle Messaging Gateway
- Oracle Messaging Gateway Views

Introduction to Oracle Messaging Gateway

The Messaging Gateway administration package DBMS_MGWADM provides an interface for creating Messaging Gateway agents, managing agents, creating messaging system links, registering non-Oracle queues, and setting up propagation jobs.

Topics:

- Oracle Messaging Gateway Overview
- Oracle Messaging Gateway Features
- Oracle Messaging Gateway Architecture
- Propagation Processing Overview
- Oracle Database AQ Buffered Messages and Messaging Gateway

Oracle Messaging Gateway Overview

Messaging Gateway enables communication between applications based on non-Oracle messaging systems and Oracle Database Advanced Queuing.

Oracle Database Advanced Queuing provides propagation between two Oracle Database Advanced Queuing queues to enable e-business (HTTP through IDAP). Messaging Gateway extends this to applications based on non-Oracle messaging systems.

Because Messaging Gateway is integrated with Oracle Database Advanced Queuing and Oracle Database, it offers reliable message delivery. Messaging Gateway guarantees that messages are delivered once and only once between Oracle Database Advanced Queuing and non-Oracle messaging systems that support persistence. The PL/SQL interface provides an easy-to-learn administrative API, especially for developers already proficient in using Oracle Database Advanced Queuing.

This release of Messaging Gateway supports the integration of Oracle Database Advanced Queuing with applications based on WebSphere MQ 9.0 and TIB/Rendezvous 8.2.

Oracle Messaging Gateway Features

Messaging Gateway provides the following features:

- Extends Oracle Database Advanced Queuing message propagation

  Messaging Gateway propagates messages between Oracle Database Advanced Queuing and non-Oracle messaging systems. Messages sent by Oracle Database Advanced Queuing applications can be received by non-Oracle messaging system applications. Conversely, messages published by non-Oracle messaging system applications can be consumed by Oracle Database Advanced Queuing applications.

- Support for Java Message Service (JMS) messaging systems

  Messaging Gateway propagates messages between Oracle Java Message Service (Oracle JMS) and WebSphere MQ Java Message Service (WebSphere MQ JMS).

- Native message format support

  Messaging Gateway supports the native message formats of messaging systems. Oracle Database Advanced Queuing messages can have `RAW` or any Oracle object type payload. WebSphere MQ messages can be text or byte messages. TIB/Rendezvous messages can be any TIB/Rendezvous wire format datatype except the nested datatype `MSG` and those with unsigned integers.

- Message conversion

  Messaging Gateway facilitates message conversion between Oracle Database Advanced Queuing messages and non-Oracle messaging system messages. Messages are converted through either automatic routines provided by Messaging Gateway or customized message transformation functions that you provide.

  Note:

  – Messaging Gateway does not support message propagation between JMS and non-JMS messaging systems.

  – Oracle Database AQ Sharded Queues are not supported by MGW in 12*c* Release 2 (12.2)

- Integration with Oracle Database

  Messaging Gateway is managed through a PL/SQL interface similar to that of Oracle Database Advanced Queuing. Configuration information is stored in Oracle Database tables. Message propagation is carried out by an external process of the Oracle Database server.

- Guaranteed message delivery

  If the messaging systems at the propagation source and propagation destination both support transactions, then Messaging Gateway guarantees that persistent messages are propagated exactly once. If messages are not persistent or transactions are not supported by the messaging systems at the propagation source or propagation destination, then at-most-once propagation is guaranteed.

- Security support

  Messaging Gateway supports client authentication of Oracle Database and non-Oracle messaging systems.

  Messaging Gateway also allows Secure Socket Layer (SSL) support for IBM WebSphere MQ and WebSphere MQ JMS connections made by the Messaging Gateway agent.

- Multiple agent support

  Messaging Gateway supports multiple agents for a given database. Users can partition propagation jobs based on functionality, organizations, or workload and assign them to different Messaging Gateway agents. This allows Messaging Gateway to scale in an Oracle RAC environment and enables propagation job grouping and isolation.

See Also:

- "Propagation Processing Overview"
- Oracle Messaging Gateway Message Conversion
- "Converting Oracle Messaging Gateway Non-JMS Messages"

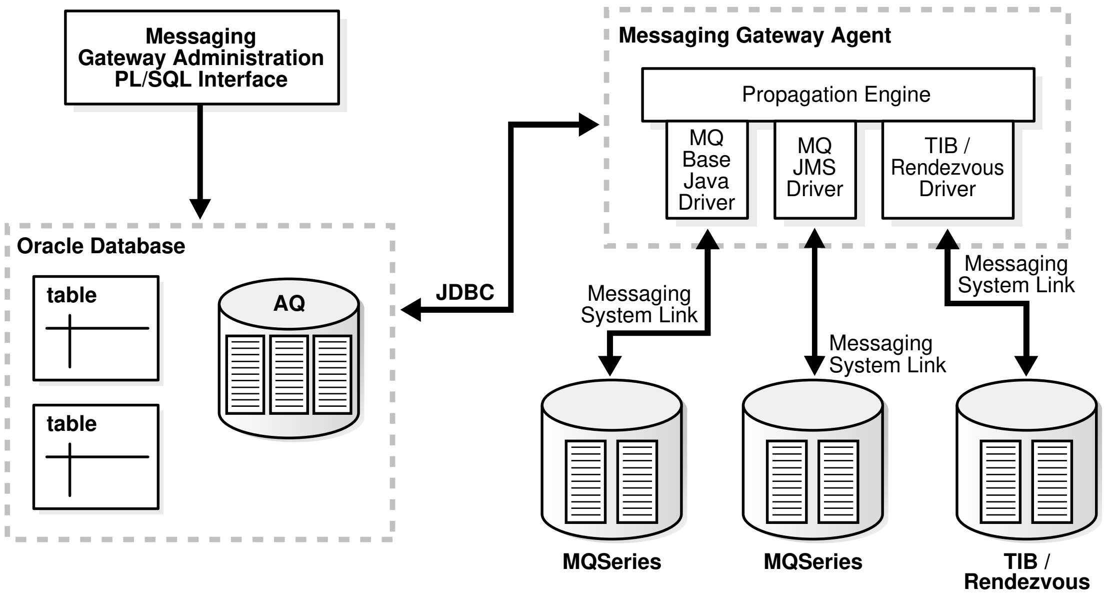
Oracle Messaging Gateway Architecture

Messaging Gateway has two main components:

- Administration Package DBMS_MGWADM
- Messaging Gateway Agent

Figure C-1 shows how these components work together with Oracle Database and non-Oracle messaging systems.

Figure C-1 Messaging Gateway Architecture



Administration Package DBMS_MGWADM

The Messaging Gateway administration package `DBMS_MGWADM` provides an interface for creating named Messaging Gateway agents, managing agents, creating messaging system links, registering non-Oracle queues, and setting up propagation jobs.

Users call the procedures in the package to make configuration changes regardless of whether the Messaging Gateway agent is running. If the Messaging Gateway agent is running, then the procedures in the package send notifications for configuration changes to the agent. The agent dynamically alters its configuration for most configuration changes, although some changes require that the agent be shut down and restarted before they take effect. All the procedures in the package are serialized to guarantee that the Messaging Gateway agent receives and processes notifications in the same order as they are made.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information on `DBMS_MGWADM`

Oracle Messaging Gateway Agent

The Messaging Gateway agent runs as an external process of the Oracle Database server and processes propagation jobs. It is started and shut down by calling the `STARTUP` and `SHUTDOWN` procedures in `DBMS_MGWADM` package.

The Messaging Gateway agent contains a multithreaded propagation engine and a set of drivers for messaging systems. The propagation engine fairly schedules propagation jobs and processes propagation jobs concurrently. The polling thread in the agent periodically polls the source queues of enabled propagation jobs and wakes up worker threads to process propagation jobs if messages are available. The drivers for non-Oracle messaging systems run as clients of the messaging systems for all messaging operations.

Oracle Database

As an Oracle Database feature, Messaging Gateway provides a mechanism of message propagation between Oracle Database Advanced Queuing and non-Oracle messaging systems. Oracle Database Advanced Queuing is involved in every propagation job as either propagation source or propagation destination.

Messaging Gateway is managed through the PL/SQL administration package `DBMS_MGWADM`. All configuration information and execution state information of Messaging Gateway are stored in Oracle Database and can be accessed through database views.

The Messaging Gateway agent runs as an external procedure of the Oracle Database server. Therefore, it runs only when its associated database server is running.

Non-Oracle Messaging Systems

The Messaging Gateway agent connects to non-Oracle messaging systems through messaging system links. Messaging system links are communication channels between the Messaging Gateway agent and non-Oracle messaging systems. Users can use the administration package `DBMS_MGWADM` to configure multiple links to the same or different non-Oracle messaging systems.

Queues in non-Oracle messaging systems, such as WebSphere MQ queues, TIB/Rendezvous subjects, and WebSphere MQ JMS destinations (queues and topics) can all serve as propagation sources and destinations for Messaging Gateway. They are referred to as foreign queues. All foreign queues involved in message propagation as source queues, destination queues, or exception queues must be registered through the administration package. The registration of a foreign queue does not create the physical queue in a non-Oracle messaging system, but merely records information about the queue, such as the messaging system link to access it, its native name, and its domain (queue or topic). The physical queue must be created through the administration interface of the non-Oracle messaging system.

See Also:

"Registering a Non-Oracle Queue"

Propagation Processing Overview

Propagation jobs must be defined in order for messages to be propagated from one messaging system to another. A propagation job defines the source queue, destination queue, and various other attributes that affect the processing of the propagation job.

If the propagation source is a queue (point-to-point), then the Messaging Gateway agent moves all messages in the queue to the destination. If the propagation source is a topic (publish/subscribe), then the Messaging Gateway agent creates a subscription on the propagation source topic. The agent moves all messages that are published to the topic after the subscription is created.

A propagation job is processed when it is enabled. Disabling a propagation job stops propagation processing but does not stop message subscription.

When the Messaging Gateway agent processes a propagation job, it dequeues messages from the source queue and enqueues the messages to the destination queue. As each message is propagated, it is converted from its native format in the source messaging system to its native format in the destination messaging system. Messaging Gateway provides automatic message conversions between simple and commonly used message formats. You can customize message conversions by providing your own message transformation functions.

When the Messaging Gateway agent fails to convert a message from the source format to the destination format, the agent moves the message from the source queue to an exception queue, if the exception queue exists, and continues to process the propagation job.

If the Messaging Gateway agent runs into failures when processing a propagation job, it retries up to sixteen times in an exponential backoff scheme (from two seconds up to thirty minutes) before it stops retrying.

To guarantee reliable message delivery, Messaging Gateway requires logging queues in messaging systems that support transactions and persistent messages. The Messaging Gateway agent uses the logging queues to store the processing states of propagation jobs so that it can restore propagation processing from failures.

See Also:

"Configuring Oracle Messaging Gateway Propagation Jobs"

Oracle Database AQ Buffered Messages and Messaging Gateway

Messaging Gateway does not support propagation of buffered messages. In outbound propagation, the Messaging Gateway agent dequeues only persistent messages from AQ queues. In inbound propagation, the Messaging Gateway agent always enqueues persistent messages into AQ queues.

Getting Started with Oracle Messaging Gateway

The following topics describe Oracle Messaging Gateway (MGW) prerequisites and how to load, set up, and unload Messaging Gateway. They also describe how to set up and modify the `mgw.ora` initialization file.

- Oracle Messaging Gateway Prerequisites
- Loading and Setting Up Oracle Messaging Gateway
- Setting Up Non-Oracle Messaging Systems
- Verifying the Oracle Messaging Gateway Setup
- Unloading Oracle Messaging Gateway
- Understanding the mgw.ora Initialization File

Oracle Messaging Gateway Prerequisites

Messaging Gateway uses one Oracle Scheduler job for each Messaging Gateway agent. If the value of the `JOB_QUEUE_PROCESSES` database initialization parameter is zero, then no Oracle

Scheduler jobs will run. If the value is non-zero, it effectively becomes the maximum number of Scheduler jobs and job queue jobs that can run concurrently.

Verify that a non-zero value is set, and that it is large enough to accommodate a Scheduler job for each Messaging Gateway agent to be started.

Loading and Setting Up Oracle Messaging Gateway

Perform the following procedures before running Messaging Gateway:

- Loading Database Objects into the Database
- Modifying listener.ora for the External Procedure
- Modifying tnsnames.ora for the External Procedure
- Setting Up an mgw.ora Initialization File
- Creating an Oracle Messaging Gateway Administrator User
- Creating an Oracle Messaging Gateway Agent User
- Configuring Oracle Messaging Gateway Connection Information
- Configuring Oracle Messaging Gateway in an Oracle RAC Environment

Note:

These setup instructions are specific to 32-bit and 64-bit versions of the Windows and 64-bit versions of the Unix-based operating systems. The tasks apply to both Windows and Unix-based operating systems, except where "Windows Operating System Only" or "Linux Operating System Only" is indicated. For other operating systems, see operating-system specific documentation.

Loading Database Objects into the Database

Using SQL*Plus, run `ORACLE_HOME/mgw/admin/catmgw.sql` as user `SYS` as `SYSDBA`. This script loads the database objects necessary for Messaging Gateway, including roles, tables, views, object types, and PL/SQL packages. It creates public synonyms for Messaging Gateway PL/SQL packages. It creates two roles, `MGW_ADMINISTRATOR_ROLE` and `MGW_AGENT_ROLE`, with certain privileges granted. All objects are owned by `SYS`.

Note:

In a CDB environment, run `ORACLE_HOME/mgw/admin/catmgw.sql` as PDB root.

Modifying listener.ora for the External Procedure

This procedure is for Unix-based operating systems only. Static service information for the listener is not necessary on the Windows operating system.

You must modify `listener.ora` so that the Messaging Gateway PL/SQL packages can call the external procedure.

1. Verify that the default Inter-process Communication (IPC) protocol address for the external procedures is set.

   ```
   LISTENER = (ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC))
   ```

2. Add static service information for the listener in step 1. This involves setting a `SID_DESC` for the listener. Within the `SID_DESC`, the parameters described in Table C-1 are important to Messaging Gateway and must be specified according to your own situation.

Example C-1 Adding Static Service Information for a Listener

```
# Add a SID_DESC
SID_LIST_LISTENER= (SID_LIST=
(SID_DESC =
   (SID_NAME= mgwextproc)
   (ENVS=
     "LD_LIBRARY_PATH=JRE_HOME/lib/amd64:JRE_HOME/lib/amd64/server:ORACLE_HOME/lib")
   (ORACLE_HOME=ORACLE_HOME)
   (PROGRAM = extproc))
```

Table C-1 SID_DESC Parameters

Parameter	Description
`SID_NAME`	The SID that is specified in the net service name in `tnsnames.ora`. In the following example, the `SID_NAME` is `mgwextproc`.

`ENVS`	Set up the `LD_LIBRARY_PATH` environment needed for the external procedure to run. The `LD_LIBRARY_PATH` must contain the following paths:
	`JRE_HOME``/lib/`*`PLATFORM_TYPE`*
	`JRE_HOME``/lib/`*`PLATFORM_TYPE`*`/server`
	`ORACLE_HOME``/lib`
	It should also contain any additional libraries required by third-party messaging systems. See "Setting Up Non-Oracle Messaging Systems".
	The `ENVS` value is `LD_LIBRARY_PATH_64` for Oracle Solaris on SPARC (64-Bit) and Oracle Solaris on x86-64 (64-Bit), `LIBPATH` for IBM AIX on POWER Systems, and `LD_LIBRARY_PATH` for the remaining platforms.

`ORACLE_HOME`	Your Oracle home directory. Using `$ORACLE_HOME` does not work.
`PROGRAM`	The name of the external procedure agent, which is `extproc`

Note:

`JRE_HOME` represents the root directory of a JRE installation, just as *`ORACLE_HOME`* represents the root directory of an Oracle installation. Oracle recommends that you use the JRE installed with Oracle Database.

Example C-1 adds `SID_NAME` `mgwextproc` to a listener.ora file for Linux x86.

Modifying tnsnames.ora for the External Procedure

For the external procedure, configure a net service name `MGW_AGENT` in `tnsnames.ora` whose connect descriptor matches the information configured in `listener.ora`, as shown in Example C-2. The net service name must be `MGW_AGENT` (this value is fixed). The `KEY` value must match the `KEY` value specified for the IPC protocol in `listener.ora`. The SID value must match the value specified for `SID_NAME` of the `SID_DESC` entry in `listener.ora`.

Note:

If the `names.default_domain` parameter for `sqlnet.ora` has been used to set a default domain, then that domain must be appended to the `MGW_AGENT` net service name in `tnsnames.ora`. For example, if `sqlnet.ora` contains the entry `names.default_domain=acme.com`, then the net service name in `tnsnames.ora` must be `MGW_AGENT.acme.com`.

Example C-2 Configuring MGW_AGENT

```
MGW_AGENT =
(DESCRIPTION=
   (ADDRESS_LIST= (ADDRESS= (PROTOCOL=IPC)(KEY=EXTPROC)))
   (CONNECT_DATA= (SID=mgwextproc)))
```

Setting Up an mgw.ora Initialization File

The Messaging Gateway default initialization file *`ORACLE_HOME`*`/mgw/admin/mgw.ora` is a text file. The Messaging Gateway external procedure uses it to get initialization parameters to start the Messaging Gateway agent. Copy *`ORACLE_HOME`*`/mgw/admin/sample_mgw.ora` to `mgw.ora` and modify it according to your situation.

The following procedure sets environment variables and other parameters required for all applications of Messaging Gateway:

1. **Windows Operating System Only:** Set the `MGW_PRE_PATH` variable. Its value is the path to the `jvm.dll` library:

   ```
   set MGW_PRE_PATH = JRE_HOME\bin\client
   ```

   This variable is prepended to the path inherited by the Messaging Gateway agent process.
2. Set `CLASSPATH` to include at least the following:
   - JRE runtime classes:

     `JRE_HOME``/lib/rt.jar`
   - Oracle JDBC classes:

     `ORACLE_HOME``/jdbc/lib/ojdbc6.jar`
   - Oracle internationalization classes:

     `ORACLE_HOME``/jlib/orai18n.jar`
   - SQLJ runtime:

     `ORACLE_HOME``/sqlj/lib/runtime12.jar`
   - Java Message Service (JMS) interface

     `ORACLE_HOME``/rdbms/jlib/jmscommon.jar`
   - Oracle JMS implementation classes

     `ORACLE_HOME``/rdbms/jlib/aqapi.jar`

- Java transaction API

  ORACLE_HOME/jlib/jta.jar
- Any additional classes needed for Messaging Gateway to access non-Oracle messaging systems

> **Note:**
>
> Replace *ORACLE_HOME* with the appropriate, spelled-out value. Using $ORACLE_HOME, for example, does not work.
>
> Users of the Windows operating system must set CLASSPATH using the Windows operating system path syntax.

> **See Also:**
>
> "Setting Up Non-Oracle Messaging Systems"

Creating an Oracle Messaging Gateway Administrator User

To perform Messaging Gateway administration work, a database user must be created with MGW_ADMINISTRATOR_ROLE privileges, as shown in Example C-3.

Example C-3 Creating a Messaging Gateway Administrator User

```
CREATE USER admin_user IDENTIFIED BY admin_password;
GRANT CREATE SESSION to admin_user;
GRANT MGW_ADMINISTRATOR_ROLE to admin_user;
```

Creating an Oracle Messaging Gateway Agent User

To establish the Messaging Gateway agent connection back to the database, a database user with MGW_AGENT_ROLE privileges must be created, as shown in Example C-4.

Example C-4 Creating a Messaging Gateway Agent User

```
CREATE USER agent_user IDENTIFIED BY agent_password;
GRANT CREATE SESSION to agent_user;
GRANT MGW_AGENT_ROLE to agent_user;
```

Configuring Oracle Messaging Gateway Connection Information

After the Messaging Gateway agent user is created, the administration user uses DBMS_MGWADM.ALTER_AGENT to configure Messaging Gateway with the user name, password, and database connect string used by the Messaging Gateway agent to connect back to the database, as shown in Example C-5. Use the Messaging Gateway user name and password that you created in "Creating an Oracle Messaging Gateway Agent User". The database connect string parameter can be set to either a net service name in tnsnames.ora (with IPC protocol for better performance) or NULL. If NULL, then the oracle_sid parameter must be set in mgw.ora.

For this release, always specify a not NULL value for the database connect string parameter when calling DBMS_MGWADM.ALTER_AGENT.

Example C-5 Configuring Messaging Gateway Connection Information

```
set echo off
set verify off
connect admin_user

ACCEPT password CHAR PROMPT 'Enter the password for AGENT_USER: ' HIDE

EXEC DBMS_MGWADM.ALTER_AGENT(
      agent_name  => 'default_agent',
      username    => 'agent_user',
      password    => '&password',
      database    => 'agent_database');
```

Configuring Oracle Messaging Gateway in an Oracle RAC Environment

This section contains these topics:

- Configuring Connection Information for the MGW Agent Connections
- Setting the Oracle RAC Instance for the Messaging Gateway Agent

Configuring Connection Information for the MGW Agent Connections

You must make all database connections made by the Messaging Gateway agent to the instance on which the Messaging Gateway agent process is running. This ensures correct failover behavior in an Oracle RAC environment. You can configure connections this way by having the instances use slightly different `tnsnames.ora` files. Each file contains an entry with the same net service name, but the connect data refers to only the instance associated with that `tnsnames.ora` file. The common net service name would then be used for the database parameter when `DBMS_MGWADM.ALTER_AGENT` is used to configure the Messaging Gateway agent database connection information.

For example, in a two-instance Oracle RAC environment with instances `OraDB1` and `OraDB2`, where the net service name `AGENT_DB` is to be used, the `tnsnames.ora` for instance `OraDB1` would look like this:

```
AGENT_DB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = myhost1.mycorp.com)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = OraDB10.mycorp.com)
      (INSTANCE_NAME = OraDB1)
    )
  )
```

The `tnsnames.ora` for `OraDB2` would look like this:

```
AGENT_DB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = myhost2.mycorp.com)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = OraDB10.mycorp.com)
      (INSTANCE_NAME = OraDB2)
    )
  )
```

You would then configure Messaging Gateway agent user connection information by running the following command:

```
EXEC DBMS_MGWADM.ALTER_AGENT(
   agent_name    => 'default_agent',
   username      => 'agent_user',
   password      => 'agent_password',
   database      => 'agent_db');
```

Setting the Oracle RAC Instance for the Messaging Gateway Agent

Messaging Gateway provides service affinity for the Messaging Gateway agent external process by leveraging the database service support of the Oracle Scheduler. By default, a Messaging Gateway agent will use the default database service that is mapped to all instances. If you want a Messaging Gateway agent to start on a select group of database instances, you must create a database service for those instances and then assign the database service to the Messaging Gateway agent using the `SERVICE` parameter of the `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT` procedures. The `DBMS_MGWADM.STARTUP` procedure submits an Oracle Scheduler job that starts the Messaging Gateway agent external process when the Scheduler job is executed. The Scheduler job will use the database service configured for the Messaging Gateway agent.

The database service specified by the `SERVICE` parameter is only used for the service affinity of the Oracle Scheduler job and thus the service affinity for the Messaging Gateway external process. It is not used for the database connections made by the Messaging Gateway agent user. Those JDBC client connections are based on the values specified for the `DATABASE` and `CONNTYPE` parameters.

See Also:

"Running the Oracle Messaging Gateway Agent on Oracle RAC"

Setting Up Non-Oracle Messaging Systems

This section contains these topics:

- Setting Up for TIB/Rendezvous
- Setting Up for WebSphere MQ Base Java or JMS

Setting Up for TIB/Rendezvous

Running as a TIB/Rendezvous Java client application, the Messaging Gateway agent requires TIB/Rendezvous software to be installed on the computer where the Messaging Gateway agent runs. In this section *`TIBRV_HOME`* refers to the installed TIB/Rendezvous software location.

Modifying listener.ora

On the Linux operating system, `LD_LIBRARY_PATH` in the entry for Messaging Gateway must include *`TIBRV_HOME`*`/lib` for the agent to access TIB/Rendezvous shared library files.

See Also:

"Modifying listener.ora for the External Procedure"

On the Windows operating system, you are not required to modify `listener.ora`. But the system environment variable `PATH` must include *`TIBRV_HOME`*`\bin`.

Modifying mgw.ora

`MGW_PRE_PATH` must include the directory that contains the TIB/Rendezvous license ticket file (`tibrv.tkt`), which usually is located in *`TIBRV_HOME/bin`*.

`CLASSPATH` must include the TIB/Rendezvous jar file *`TIBRV_HOME`*`/lib/tibrvj.jar`. If you use your own customized TIB/Rendezvous advisory message callback, then the location of the callback class must also be included.

You can set the following Java properties to change the default setting:

- `oracle.mgw.tibrv.encoding`
- `oracle.mgw.tibrv.intraProcAdvSubjects`
- `oracle.mgw.tibrv.advMsgCallback`

See Also:

"Understanding the mgw.ora Initialization File"

Example C-6 Setting Java Properties

```
setJavaProp oracle.mgw.tibrv.encoding=ISO8859_1
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=_RV.>
setJavaProp oracle.mgw.tibrv.advMsgCallback=MyadvCallback
```

Setting Up for WebSphere MQ Base Java or JMS

The WebSphere MQ client and WebSphere MQ classes for Java and JMS must be installed on the computer where the Messaging Gateway agent runs. In this section *`MQ_HOME`* refers to the location of the installed client. On the Linux operating system, this location is always `/opt/mqm`. On the Windows operating system, the installed location can vary.

Modifying listener.ora

No extra modification of `listener.ora` is necessary for Messaging Gateway to access WebSphere MQ.

Modifying mgw.ora

When using WebSphere MQ Base Java (non-JMS) interface, set `CLASSPATH` to include at least the following (in addition to those in "Setting Up an mgw.ora Initialization File"):

- *`MQ_HOME`*`/java/lib/com.ibm.mq.jar`
- *`MQ_HOME`*`/java/lib/connector.jar`

When using WebSphere MQ JMS interface, set `CLASSPATH` to include at least the following (in addition to those in "Setting Up an mgw.ora Initialization File"):

- *`MQ_HOME`*`/java/lib/com.ibm.mqjms.jar`
- *`MQ_HOME`*`/java/lib/com.ibm.mq.jar`
- *`MQ_HOME`*`/java/lib/connector.jar`

Verifying the Oracle Messaging Gateway Setup

The following procedure verifies the setup and includes a simple startup and shutdown of the Messaging Gateway agent:

1. Start the database listeners.

   Start the listener for the external procedure and other listeners for the regular database connection.
2. Test the database connect string for the Messaging Gateway agent user.

   Run `sqlplus` *`agent_user/agent_password@agent_database`*.

   If it is successful, then the Messaging Gateway agent can connect to the database.
3. **Linux Operating System Only:** Test the net service entry used to call the external procedure.

   Run `sqlplus` *`agent_user/agent_password`*`@MGW_AGENT`.

   This should fail with "ORA-28547: connection to server failed, probable Oracle Net admin error". Any other error indicates that the `tnsnames.ora`, `listener.ora`, or both are not correct.
4. Connect as *`admin_user`* and call `DBMS_MGWADM.STARTUP` to start the Messaging Gateway agent.
5. Using the `MGW_GATEWAY` view, wait for `AGENT_STATUS` to change to `RUNNING` and `AGENT_PING` to change to `REACHABLE`.
6. Connect as *`admin_user`* and call `DBMS_MGWADM.SHUTDOWN` to shut down the Messaging Gateway agent.
7. Using the `MGW_GATEWAY` view, wait for `AGENT_STATUS` to change to `NOT_STARTED`.

Unloading Oracle Messaging Gateway

Use this procedure to unload Messaging Gateway:

1. Shut down Messaging Gateway.
2. Remove any user-created queues whose payload is a Messaging Gateway canonical type (for example, `SYS.MGW_BASIC_MSG_T`).
3. Using SQL*Plus, run *`ORACLE_HOME`*`/mgw/admin/catnomgw.sql` as user `SYS` as `SYSDBA`.

   This drops the database objects used by Messaging Gateway, including roles, tables, views, packages, object types, and synonyms.
4. Remove entries for Messaging Gateway created in `listener.ora` and `tnsnames.ora`.

Understanding the mgw.ora Initialization File

Messaging Gateway reads initialization information from a text file when the Messaging Gateway agent starts. The initialization file contains lines for setting initialization parameters, environment variables, and Java properties. Each entity must be specified on one line. Leading whitespace is trimmed in all cases.

A Messaging Gateway administrator can specify the initialization file to be used for a Messaging Gateway agent via `DBMS_MGWADM.CREATE_AGENT` and `DBMS_MGWADM.ALTER_AGENT`. If an initialization file is not specified then a default initialization file will be used.

The default initialization file for the default agent is `ORACLE_HOME/mgw/admin/mgw.ora`.

The default initialization file for a named agent is `ORACLE_HOME/mgw/admin/mgw_AGENTNAME.ora` where `AGENTNAME` is the name in uppercase of the Messaging Gateway agent. For example, if the agent name is `my_agent` then the name of the agent's default initialization file is `ORACLE_HOME/mgw/admin/mgw_MY_AGENT.ora`. If the default initialization file for a named agent is not found then `ORACLE_HOME/mgw/admin/mgw.ora` will be used.

mgw.ora Initialization Parameters

The initialization parameters are typically specified by lines having a "*name*=*value*<NL>" format where *name* represents the parameter name, *value* represents its value and <NL> represents a new line.

log_directory

Usage:

Specifies the directory where the Messaging Gateway log/trace file is created.

Format:

```
log_directory = value
```

Default:

ORACLE_HOME/mgw/log

Example:

```
log_directory = /private/mgwlog
```

log_level

Usage:

Specifies the level of logging detail recorded by the Messaging Gateway agent. The logging level can be dynamically changed by calling `DBMS_MGWADM.SET_LOG_LEVEL` while the Messaging Gateway agent is running. Oracle recommends that log level `0` (the default value) be used at all times.

Format:

```
log_level = value
```

Values:

`0` for basic logging; equivalent to `DBMS_MGWADM.BASIC_LOGGING`

`1` for light tracing; equivalent to `DBMS_MGWADM.TRACE_LITE_LOGGING`

`2` for high tracing; equivalent to `DBMS_MGWADM.TRACE_HIGH_LOGGING`

`3` for debug tracing; equivalent to `DBMS_MGWADM.TRACE_DEBUG_LOGGING`

`4` for high debug tracing; equivalent to `DBMS_MGWADM.TRACE_DEBUG_HIGH_LOGGING`

Example:

```
log_level = 0
```

mgw.ora Environment Variables

Because the Messaging Gateway process environment is not under the direct control of the user, certain environment variables should be set using the initialization file. The environment variables currently used by the Messaging Gateway agent are `CLASSPATH`, `MGW_PRE_PATH`, and `ORACLE_SID`.

Environment variables such as `CLASSPATH` and `MGW_PRE_PATH` are set so the Messaging Gateway agent can find the required shared objects, Java classes, and so on. Environment variables are specified by lines having a "`set` *`env_var`*`=value<NL>`" or "`setenv` *`env_var`*`=`*`value`*`<NL>`" format where *`env_var`* represents the name of the environment variable to set, *`value`* represents the value of the environment variable, and `<NL>` represents a new line.

CLASSPATH

Usage:

Used by the Java Virtual Machine to find Java classes needed by the Messaging Gateway agent for propagation between Oracle Database Advanced Queuing and non-Oracle messaging systems.

Format:

```
set CLASSPATH=value
```

Example:

```
set CLASSPATH=ORACLE_HOME/jdbc/lib/ojdbc6.jar:JRE_HOME/lib/rt.jar:
ORACLE_HOME/sqlj/lib/runtime12.jar:ORACLE_HOME/jlib/orai18n.jar:ORACLE_HOME/rdbms/
jlib/jmscommon.jar:ORACLE_HOME/rdbms/jlib/aqapi.jar:ORACLE_HOME/jlib/jta.jar:
/opt/mqm/java/lib/com.ibm.mq.jar:/opt/mqm/java/lib/com.ibm.mqjms.jar:/opt/mqm/java
/lib/connector.jar
```

MGW_PRE_PATH

Usage:

Appended to the front of the path inherited by the Messaging Gateway process. For the Windows operating system, this variable must be set to indicate where the library `jvm.dll` is found.

Format:

```
set MGW_PRE_PATH=value
```

Example:

```
set MGW_PRE_PATH=JRE_HOME\bin\client
```

ORACLE_SID

Usage:

Can be used when a service name is not specified when configuring Messaging Gateway.

Format:

```
set ORACLE_SID=value
```

Example:

```
set ORACLE_SID=my_sid
```

mgw.ora Java Properties

You must specify Java system properties for the Messaging Gateway JVM when working with TIB/Rendezvous subjects. You can use the `setJavaProp` parameter of the Messaging Gateway initialization file for this. Java properties are specified by lines having a "`setJavaProp` *`prop_name`*`=`*`value`*`<NL>`" format, where *`prop_name`* represents the name of the Java property to set, *`value`* represents the value of the Java property, and `<NL>` represents a new line character.

oracle.mgw.batch_size

Usage:

This Java property represents the maximum number of messages propagated in one transaction. It serves as a default value if the Messaging Gateway job option, `MsgBatchSize`, is not specified. If altered from the default, then consideration should be given to the expected message size and the Messaging Gateway agent memory (see `max_memory` parameter of `DBMS_MGWADM.ALTER_AGENT`). The minimum value of this Java property is 1, the maximum is 100, and the default is 30.

See Also:

"DBMS_MGWADM" in *Oracle Database PL/SQL Packages and Types Reference*

Syntax:

```
setJavaProp oracle.mgw.batch_size=value
```

Example:

```
setJavaProp oracle.mgw.batch_size=10
```

oracle.mgw.polling_interval

Usage:

This parameter specifies the time (in milliseconds) that must elapse between polls for available messages of a propagation source queue. The default polling interval used by Messaging Gateway is 5000 milliseconds (5 seconds). The minimum value of this Java property is 100 millisecond, and the maximum value is 60 seconds.

Syntax:

```
setJavaProp oracle.mgw.polling_interval=value
```

Example:

```
setJavaProp oracle.mgw.polling_interval=1000
```

oracle.mgw.tibrv.encoding

Usage:

This parameter specifies the character encoding to be used by the TIB/Rendezvous messaging system links. Only one character set for all configured TIB/Rendezvous links is allowed due to TIB/Rendezvous restrictions. The default is ISO 8859-1 or the character set specified by the Java system property `file.encoding`.

Syntax:

```
setJavaProp oracle.mgw.tibrv.encoding=value
```

Example:

```
setJavaProp oracle.mgw.tibrv.encoding=ISO8859_1
```

oracle.mgw.tibrv.intraProcAdvSubjects

Usage

Used for all TIB/Rendezvous messaging system links, this parameter specifies the names of system advisory subjects that present on the intraprocess transport.

Syntax

```
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=
advisorySubjectName[:advisorySubjectName]
```

Example:

```
setJavaProp oracle.mgw.tibrv.intraProcAdvSubjects=_RV.>
```

oracle.mgw.tibrv.advMsgCallback

Usage:

Used for all TIB/Rendezvous messaging system links, this parameter specifies the name of the Java class that implements the `TibrvMsgCallback` interface to handle system advisory messages. If it is not specified, then the default system advisory message handler provided by Messaging Gateway is used, which writes system advisory messages into Messaging Gateway log files. If it is specified, then the directory where the class file is stored must be included in the `CLASSPATH` in `mgw.ora`.

Syntax:

```
setJavaProp oracle.mgw.tibrv.advMsgCallback=className
```

Example:

```
setJavaProp oracle.mgw.tibrv.advMsgCallback=MyAdvCallback
```

oracle.net.tns_admin

Usage:

This parameter specifies the directory of the tnsnames.ora file. It must be set if the Messaging Gateway agent is configured to use the JDBC Thin driver and the database specifier of the agent connection information is a TNSNames alias. This does not need to be set if the JDBC OCI driver is used or the database specifier is something other than a TNSNames alias.

Syntax:

```
setJavaProp oracle.net.tns_admin=value
```

Example:

```
setJavaProp oracle.net.tns_admin=/myoraclehome/network/admin
```

mgw.ora Comment Lines

Comment lines are designated with a `#` character as the first character of the line.

Working with Oracle Messaging Gateway

After Oracle Messaging Gateway (MGW) is loaded and set up, it is ready to be configured and run. You can use `DBMS_MGWADM.ALTER_AGENT` to set the user name, password, database specifier, and connection type the Messaging Gateway agent will use for creating database connections.

Topics:

- Configuring the Oracle Messaging Gateway Agent
- Starting and Shutting Down the Oracle Messaging Gateway Agent
- Configuring Messaging System Links
- Configuring Non-Oracle Messaging System Queues
- Configuring Oracle Messaging Gateway Propagation Jobs
- Propagation Jobs_ Subscribers_ and Schedules
- Configuration Properties

Note:

All commands in the examples must be run as a user granted `MGW_ADMINISTRATOR_ROLE`.

See Also:

"DBMS_MGWADM" and "DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference*

Configuring the Oracle Messaging Gateway Agent

Messages are propagated between Oracle Database Advanced Queuing and non-Oracle messaging systems by the Messaging Gateway agent. The Messaging Gateway agent runs as an external process of the Oracle Database server.

Messaging Gateway supports multiple agents for a given database. A default agent is automatically created that has the name of `DEFAULT_AGENT`. Additional named agents can be created to provide propagation job isolation and grouping, and scaling in an Oracle RAC environment. The default agent is usually sufficient for single instance, non-Oracle RAC, environments.

This section contains these topics:

- Creating a Messaging Gateway Agent
- Removing a Messaging Gateway Agent
- Setting Database Connection
- Setting the Resource Limits

Creating a Messaging Gateway Agent

You can use `DBMS_MGWADM.CREATE_AGENT` to create additional Messaging Gateway agents. The Messaging Gateway default agent, `DEFAULT_AGENT`, is automatically created when Messaging Gateway is installed and will always exist.

Agents can be configured with an agent user, connection information, database service, and resource limits when the agent is created, or at a later time using `DBMS_MGWADM.ALTER_AGENT`. A Messaging Gateway agent must be configured with a database user that has been granted the role `MGW_AGENT_ROLE` before the agent can be started.

Example C-7 creates the agent named `myagent` and specifies the database connection information for the agent user. Default values are used for all other parameters.

Example C-7 Creating a Messaging Gateway Agent

```
SQL> exec DBMS_MGWADM.CREATE_AGENT(
                         agent_name  => 'myagent',
                         username    => 'mgwagent',
                         password    => 'mgwagent_password',
                         database    => 'mydatabase');
```

Removing a Messaging Gateway Agent

A Messaging Gateway agent can be removed by calling `DBMS_MGWADM.REMOVE_AGENT`. Before an agent can be removed, all Messaging Gateway links associated with the agent must be removed and the agent shut down. The default agent, `DEFAULT_AGENT`, cannot be removed. Example C-8 removes the agent named `myagent`.

Example C-8 Removing a Messaging Gateway Agent

```
SQL> exec DBMS_MGWADM.REMOVE_AGENT(agent_name   => 'myagent');
```

Setting Database Connection

The Messaging Gateway agent runs as a process external to the database. To access Oracle Database Advanced Queuing and the Messaging Gateway packages, the Messaging Gateway agent needs to establish connections to the database. You can use `DBMS_MGWADM.ALTER_AGENT` to set the user name, password and the database connect string that the Messaging Gateway agent will use for creating database connections. The user must be granted the role `MGW_AGENT_ROLE` before the Messaging Gateway agent can be started.

Example C-9 shows the Messaging Gateway default agent being configured for user `mgwagent` with password `mgwagent_password` using net service name `mydatabase`.

Example C-9   Setting Database Connection Information

```
SQL> exec DBMS_MGWADM.ALTER_AGENT (
                             agent_name => 'default_agent',
                             username   => 'mgwagent',
                             password   => 'mgwagent_password',
                             database   => 'mydatabase');
```

Setting the Resource Limits

You can use `DBMS_MGWADM.ALTER_AGENT` to set resource limits for the Messaging Gateway agent. For example, you can set the heap size of the Messaging Gateway agent process and the number of propagation threads used by the agent process. The default values are 64 MB of memory heap and one propagation thread. For named agents, these values can also be specified when the agent is created by `DBMS_MGWADM.CREATE_AGENT`.

Example C-10 sets the heap size to 96 MB and two propagation threads for the agent myagent.

The memory heap size and the number of propagation threads cannot be altered when the Messaging Gateway agent is running.

Example C-10   Setting the Resource Limits

```
SQL> exec DBMS_MGWADM.ALTER_AGENT(
                        agent_name  => 'myagent',
                        max_memory  => 96,
                        max_threads => 2);
```

Starting and Shutting Down the Oracle Messaging Gateway Agent

This section contains these topics:

- Starting the Oracle Messaging Gateway Agent
- Shutting Down the Oracle Messaging Gateway Agent
- Oracle Messaging Gateway Agent Scheduler Job
- Running the Oracle Messaging Gateway Agent on Oracle RAC

Starting the Oracle Messaging Gateway Agent

After the Messaging Gateway agent is configured, you can start the agent with `DBMS_MGWADM.STARTUP`. Example C-11 shows how to start the default agent and agent `myagent`.

You can use the `MGW_GATEWAY` view to check the status of the Messaging Gateway agent, as described in Monitoring Oracle Messaging Gateway.

Example C-11 Starting the Messaging Gateway Agent

```
SQL> exec DBMS_MGWADM.STARTUP;
SQL> exec DBMS_MGWADM.STARTUP ('myagent');
```

Shutting Down the Oracle Messaging Gateway Agent

You can use `DBMS_MGWADM.SHUTDOWN` to shut down the Messaging Gateway agent. Example C-12 shows how to shut down the Messaging Gateway default agent and agent `myagent`.

You can use the `MGW_GATEWAY` view to check if the Messaging Gateway agent has shut down successfully, as described in Monitoring Oracle Messaging Gateway.

Example C-12 Shutting Down the Messaging Gateway Agent

```
SQL> exec DBMS_MGWADM.SHUTDOWN;
SQL> exec DBMS_MGWADM.SHUTDOWN ('myagent');
```

Oracle Messaging Gateway Agent Scheduler Job

Messaging Gateway uses a Scheduler job to start the Messaging Gateway agent. This job is created when procedure `DBMS_MGWADM.STARTUP` is called. When the job is run, it calls an external procedure that creates the Messaging Gateway agent in an external process. The job is removed after:

- The agent shuts down because `DBMS_MGWADM.SHUTDOWN` was called
- The agent terminates because a non-restartable error occurs

Messaging Gateway uses `DBMS_SCHEDULER` to create a repeatable Scheduler job with a repeat interval of one minute. The job is owned by `SYS`. A repeatable job enables the Messaging Gateway agent to restart automatically when a given job instance ends because of a database shutdown, database malfunction, or a restartable error. Only one instance of a Messaging Gateway agent job runs at a given time.

Each agent uses a Scheduler job class to specify the service affinity for the agent's Scheduler job. The job class will be configured with the database service specified by `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT`. A database administrator is responsible for setting up the database service. If no database service is specified, the default database service that maps to every instance is used.

The name of the Scheduler job class used by the Messaging Gateway default agent is `SYS.MGW_JOBCLS_DEFAULT_AGENT`. The Scheduler job used by the default agent is `SYS.MGW_JOB_DEFAULT_AGENT`.

The name of the Scheduler job class used by a Messaging Gateway named agent is `SYS.MGW_JOBCLS_<agent_name>`. The Scheduler job used by a named agent is `SYS.MGW_JOB_<agent_name>`.

If the agent job encounters an error, then the error is classified as either a restartable error or non-restartable error. A restartable error indicates a problem that might go away if the agent job were to be restarted. A non-restartable error indicates a problem that is likely to persist and be encountered again if the agent job restarts. ORA-01089 (immediate shutdown in progress) and ORA-28576 (lost RPC connection to external procedure) are examples of restartable errors. ORA-06520 (error loading external library) is an example of a non-restartable error.

Messaging Gateway uses a database shutdown trigger. If the Messaging Gateway agent is running on the instance being shut down, then the trigger notifies the agent of the shutdown, and upon receipt of the notification, the agent will terminate the current run. The job scheduler will automatically schedule the job to run again at a future time.

If a Messaging Gateway agent job instance ends because of a database malfunction or a restartable error detected by the agent job, then the job will not be removed and the job scheduler will automatically schedule the job to run again at a future time.

The `MGW_GATEWAY` view shows the agent status, database service, and the database instance on which the Messaging Gateway agent is current running. The Oracle Scheduler views provide information about Scheduler jobs, job classes, and job run details.

See Also:

- "DBMS_SCHEDULER" in *Oracle Database PL/SQL Packages and Types Reference*
- Monitoring Oracle Messaging Gateway

Running the Oracle Messaging Gateway Agent on Oracle RAC

While the Messaging Gateway job startup and shutdown principles are the same for Oracle Real Application Clusters (Oracle RAC) and non-Oracle RAC environments, there are some things to keep in mind for an Oracle RAC environment.

A single process of each configured Messaging Gateway agent can be running, even in an Oracle RAC environment. For example, if the default agent and two named agents have been configured with an agent user, then one instance of all three agents could be running at the same time. The database service associated with each agent determines the service affinity of the agent's Scheduler job, and thus, the database instance on which the agent process can run.

When a database instance is shut down in an Oracle RAC environment, the Messaging Gateway shutdown trigger will notify the agent to shut down only if the Messaging Gateway agent is running on the instance being shut down. The job scheduler will automatically schedule the job to be run again at a future time, either on another instance, or if the job can only run on the instance being shut down, when that instance is restarted.

Oracle recommends that all database connections made by the Messaging Gateway agent be made to the instance on which the Messaging Gateway agent process is running. This ensures correct failover behavior in an Oracle RAC environment.

If a Messaging Gateway agent has been associated with a database service, the agent's Scheduler job will not run unless that service is current enabled on a running instance. When you shut down a database Oracle stops all services to that database and you may need to manually restart the services when you start the database.

See Also:

- "Configuring Oracle Messaging Gateway in an Oracle RAC Environment"
- "`DBMS_MGWADM`" and "`DBMS_SCHEDULER`" in *Oracle Database PL/SQL Packages and Types Reference*

Configuring Messaging System Links

Running as a client of non-Oracle messaging systems, the Messaging Gateway agent communicates with non-Oracle messaging systems through messaging system links. A messaging system link is a set of connections between the Messaging Gateway agent and a non-Oracle messaging system.

To configure a messaging system link of a non-Oracle messaging system, users must provide information for the agent to make connections to the non-Oracle messaging system. Users can specify the maximum number of messaging connections.

An agent name will be associated with each messaging system link. This is done when the link is created and cannot be changed. The agent associated with the link is then responsible for processing all propagation jobs that use a registered queue associated with that link. The Messaging Gateway default agent will be used if an agent name is not specified when the messaging system link is created.

When configuring a messaging system link for a non-Oracle messaging system that supports transactions and persistent messages, the native name of log queues for inbound and outbound propagation must be specified in order to guarantee exactly-once message delivery. The log queues should be used only by the Messaging Gateway agent. No other programs should enqueue or dequeue messages of the log queues. The inbound log queue and outbound log queue can refer to the same physical queue, but better performance can be achieved if they refer to different physical queues.

One and only one Messaging Gateway agent should access a propagation log queue. This insures that a given log queue contains log records for only those propagation jobs processed by that agent and that the agent is free to discard any other log records it might encounter.

When configuring a messaging system link, users can also specify an `options` argument. An `options` argument is a set of {name, value} pairs of type `SYS.MGW_PROPERTY`.

This section contains these topics:

Creating a WebSphere MQ Base Java Link

A WebSphere MQ Base Java link is created by calling `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` with the following information provided:

- Interface type: `DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE`
- WebSphere MQ connection information:
  - Host name and port number of the WebSphere MQ server
  - Queue manager name
  - Channel name
  - User name and password
- Maximum number of messaging connections allowed
- Log queue names for inbound and outbound propagation
- Optional information such as:
  - Send, receive, and security exits
  - Character sets

Example C-13 configures a WebSphere MQ Base Java link `mqlink`. The link is configured to use the WebSphere MQ queue manager `my.queue.manager` on host `myhost.mydomain` and port 1414, using WebSphere MQ channel `mychannel`.

This example also sets the option to register a WebSphere MQ `SendExit` class. The class `mySendExit` must be in the `CLASSPATH` set in `mgw.ora`. The Messaging Gateway default agent (`DEFAULT_AGENT`) is responsible for the link and all propagation jobs using the link.

See Also:

- "Understanding the mgw.ora Initialization File" for information on setting the CLASSPATH of the Messaging Gateway agent
- "WebSphere MQ System Properties"

Example C-13 Configuring a WebSphere MQ Base Java Link

```
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
BEGIN
  v_prop := sys.mgw_mqseries_properties.construct();

  v_prop.interface_type := dbms_mgwadm.MQSERIES_BASE_JAVA_INTERFACE;
  v_prop.max_connections := 1;
  v_prop.username := 'mqm';
  v_prop.password := 'mqm';
  v_prop.hostname := 'myhost.mydomain';
  v_prop.port     := 1414;
  v_prop.channel  := 'mychannel';
  v_prop.queue_manager := 'my.queue.manager';
  v_prop.outbound_log_queue := 'mylogq';
  -- Specify a WebSphere MQ send exit class 'mySendExit' to be associated with
  -- the queue.
  -- Note that this is used as an example of how to use the options parameter,
  -- but is not an option that is usually set.
  v_options := sys.mgw_properties(sys.mgw_property('MQ_SendExit',
                                                   'mySendExit'));
  dbms_mgwadm.create_msgsystem_link(
      linkname => 'mqlink', agent_name=>'default_agent', properties => v_prop,
      options => v_options );
END;
```

Creating a WebSphere MQ JMS Link

A WebSphere MQ JMS link is created by calling `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` with the following information provided:

- Interface type

  Java Message Service (JMS) distinguishes between queue and topic connections. The Sun Microsystem JMS 1.1 standard supports domain unification that allows both JMS queues and topics to be accessed by a single JMS connection:

  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_CONNECTION` can be used to access both JMS queues and topics. This is the recommended interface for a WebSphere MQ JMS link.
  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_QUEUE_CONNECTION` can be used to access only JMS queues.
  - A WebSphere MQ JMS link created with interface type `DBMS_MGWADM.JMS_TOPIC_CONNECTION` can be used to access only JMS topics.

- WebSphere MQ connection information:
  - Host name and port number of the WebSphere MQ server
  - Queue manager name
  - Channel name
  - User name and password

- Maximum number of messaging connections allowed

  A messaging connection is mapped to a JMS session.

- Log destination (JMS queue or JMS topic) for inbound and outbound propagation

  The log destination type must be valid for the link type. JMS unified links and JMS queue links must use JMS queues for log destinations, and JMS topic links must use topics:

  - For a WebSphere MQ JMS unified or queue link, the log queue name must be the name of a physical WebSphere MQ JMS queue created using WebSphere MQ administration tools.
  - For a WebSphere MQ JMS topic link, the log topic name must be the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is `SYSTEM.JMS.D.SUBSCRIBER.QUEUE`. A link option can be used to specify a different physical queue.

- Optional information such as:
  - Send, receive, and security exits
  - Character sets

– WebSphere MQ publish/subscribe configuration used for JMS topics

Example C-14 configures a Messaging Gateway link to a WebSphere MQ queue manager using a JMS topic interface. The link is named `mqjmslink` and is configured to use the WebSphere MQ queue manager `my.queue.manager` on host `myhost.mydomain` and port 1414, using WebSphere MQ channel `mychannel`.

This example also uses the `options` parameter to specify a nondefault durable subscriber queue to be used with the log topic. The Messaging Gateway agent `myagent` is responsible for the link and all propagation jobs using the link.

See Also:

- "Registering a WebSphere MQ JMS Queue or Topic" for more information on JMS queues and topics
- "WebSphere MQ System Properties"

Example C-14 Configuring a WebSphere MQ JMS Link

```
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
BEGIN
  v_prop := sys.mgw_mqseries_properties.construct();
  v_prop.max_connections := 1;

  v_prop.interface_type := DBMS_MGWADM.JMS_TOPIC_CONNECTION;
  v_prop.username := 'mqm';
  v_prop.password := 'mqm';
  v_prop.hostname := 'myhost.mydomain';
  v_prop.port     := 1414;
  v_prop.channel  := 'mychannel';
  v_prop.queue_manager := 'my.queue.manager';

  v_prop.outbound_log_queue := 'mylogtopic'

  -- Specify a WebSphere MQ durable subscriber queue to be used with the
  -- log topic.
  v_options := sys.mgw_properties(
  sys.mgw_property('MQ_JMSDurSubQueue', 'myDSQueue'));

  DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
      linkname     =>  'mqjmslink',
      agent_name   =>  'myagent',
      properties   =>   v_prop,
      options      =>   v_options );
END;
```

Creating a WebSphere MQ Link to Use SSL

Messaging Gateway allows SSL support for IBM WebSphere MQ and WebSphere MQ JMS connections. This section describes how to configure Messaging Gateway to use SSL for a WebSphere MQ Base Java link and the same information applies to a WebSphere MQ JMS link. There are no differences in terms of the Messaging Gateway configuration.

The following are needed in order to use SSL for WebSphere MQ connections:

- A WebSphere MQ channel configured to use SSL.
- A truststore and optionally a keystore file that are in a location accessible to the Messaging Gateway agent process. In an Oracle RAC environment, these files must be accessible to all instances on which the Messaging Gateway agent process might run, using the same path specification.
- Use `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` to create a WebSphere MQ link with the desired SSL related link options. At minimum, the `MQ_SSLCIPHERSUITE` property should be set to specify the SSL ciphersuite used by the channel.
- Use `DBMS_MGWADM.SET_OPTION` to set certain JSSE Java properties for the Messaging Gateway agent assigned to the link.

JSEE related properties:

- `java.net.ssl.keyStore`

  This property is used to specify the location of the keystore. A keystore is a database of key material used for various purposes, including authentication and data integrity.
- `java.net.ssl.keyStorePassword`

  This property is used to specify the password of the keystore. This password is used to check the integrity of the data in the keystore before accessing it.
- `java.net.ssl.trustStore`

  This property is used to specify the location of the truststore. A truststore is a keystore that is used when making decisions about which clients and servers are trusted.
- `java.net.ssl.trustStorePassword`

  This property is used to specify the password of the truststore. This password is used to check the integrity of the data in the truststore before accessing it.

The `java.net.ssl.keyStore` and `java.net.ssl.keyStorePassword` properties are only needed if the WebSphere MQ channel is configured to use SSL client authentication.

Example C-15 configures a WebSphere MQ Base Java link mqssllink to use SSL connections using the `SSL_RSA_WITH_RC4_128_MD5` ciphersuite. It assumes the channel has been configured for SSL client authentication so the Messaging Gateway agent associated with the link, `DEFAULT_AGENT`, is configured with Java properties for both a keystore and a truststore.

This configuration should be done when the Messaging Gateway agent is shut down since the Java properties set by `DBMS_MGWADM.SET_OPTION` are set only when the agent first starts. If the agent is running when the configuration is done it will need to be shutdown and restarted before the SSL connections will be used.

Note:

"WebSphere MQ System Properties"

Example C-15 Configuring a WebSphere MQ Base Java Link for SSL

```
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
  v_agent varchar2(30) := 'default_agent';
BEGIN
     v_prop := sys.mgw_mqseries_properties.construct();
```

```
    v_prop.interface_type := DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE;
    v_prop.max_connections := 1;
    v_prop.username := 'mqm';
    v_prop.password := 'mqm';
    v_prop.hostname := 'myhost.mydomain';
    v_prop.port := 1414;
    v_prop.channel := 'mysslchannel';
    v_prop.queue_manager := 'my.queue.manager';
    v_prop.outbound_log_queue := 'mylogq';

  -- specify the SSL ciphersuite
    v_options := sys.mgw_properties(
       sys.mgw_property('MQ_SSLCIPHERSUITE','SSL_RSA_WITH_RC4_128_MD5') );

    -- create the MQSeries link
   DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(linkname => 'mqssllink',
                                     agent_name => v_agent,
                                     properties => v_prop,
                                     options => v_options);

  -- set Java properties for the agent that specify the JSSE security
  -- properties for the keystore and truststore; the paths will be
  -- saved as cleartext and the passwords encrypted

  DBMS_MGWADM.SET_OPTION(target_type  => DBMS_MGWADM.AGENT_JAVA_PROP,
                         target_name  => v_agent,
                         option_name  => 'javax.net.ssl.keyStore',
                         option_value => '/tmp/mq_ssl/key.jks',
                         encrypted    => false);

  DBMS_MGWADM.SET_OPTION(target_type  => DBMS_MGWADM.AGENT_JAVA_PROP,
                         target_name  => v_agent,
                         option_name  => 'javax.net.ssl.keyStorePassword',
                         option_value => 'welcome',
                         encrypted    => true);

  DBMS_MGWADM.SET_OPTION(target_type  => DBMS_MGWADM.AGENT_JAVA_PROP,
                         target_name  => v_agent,
                         option_name  => 'javax.net.ssl.trustStore',
                         option_value => '/tmp/mq_ssl/trust.jks',
                         encrypted    => false);

  DBMS_MGWADM.SET_OPTION(target_type  => DBMS_MGWADM.AGENT_JAVA_PROP,
                         target_name  => v_agent,
                         option_name  => 'javax.net.ssl.trustStorePassword',
                         option_value => 'welcome',
                         encrypted    => true);
END;
```

Creating a TIB/Rendezvous Link

A TIB/Rendezvous link is created by calling `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` with three parameters (`service`, `network` and `daemon`) for the agent to create a corresponding transport of `TibrvRvdTransport` type.

A TIB/Rendezvous message system link does not need propagation log queues. Logging information is stored in memory. Therefore, Messaging Gateway can only guarantee at-most-once message delivery.

Example C-16 configures a TIB/Rendezvous link named `rvlink` that connects to the `rvd` daemon on the local computer. An agent name is not specified for the link so the Messaging

Gateway default agent (`DEFAULT_AGENT`) is responsible for the link and all propagation jobs using the link.

See Also:

"TIB/Rendezvous System Properties"

Example C-16   Configuring a TIB/Rendezvous Link

```
DECLARE
  v_options sys.mgw_properties;
  v_prop     sys.mgw_tibrv_properties;
BEGIN
  v_prop := sys.mgw_tibrv_properties.construct();

  DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(linkname => 'rvlink', properties => v_prop);
END;
```

Altering a Messaging System Link

Using `DBMS_MGWADM.ALTER_MSGSYSTEM_LINK`, you can alter some link information after the link is created. You can alter link information with the Messaging Gateway agent running or shut down. Example C-17 alters the link `mqlink` to change the `max_connections` property.

See Also:

"Configuration Properties" for restrictions on changes when the Messaging Gateway agent is running

Example C-17   Altering a WebSphere MQ Link

```
DECLARE
  v_options sys.mgw_properties;
  v_prop sys.mgw_mqseries_properties;
BEGIN
  -- use alter_construct() for initialization
  v_prop := sys.mgw_mqseries_properties.alter_construct();
  v_prop.max_connections := 2;

  DBMS_MGWADM.ALTER_MSGSYSTEM_LINK(
    linkname => 'mqlink', properties => v_prop);
END;
```

Removing a Messaging System Link

You can remove a Messaging Gateway link to a non-Oracle messaging system with `DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK`, but only if all registered queues associated with this link have already been unregistered. The link can be removed with the Messaging Gateway agent running or shut down. Example C-18 removes the link `mqlink`.

Example C-18 Removing a Messaging Gateway Link

```
BEGIN
  dbms_mgwadm.remove_msgsystem_link(linkname =>'mqlink');
END;
```

Views for Messaging System Links

You can use the `MGW_LINKS` view to check links that have been created. It lists the name and link type, as shown in Example C-19.

You can use the `MGW_MQSERIES_LINKS` and `MGW_TIBRV_LINKS` views to check messaging system type-specific configuration information, as shown in Example C-20.

Example C-19 Listing All Messaging Gateway Links

```
SQL> select link_name, link_type from MGW_LINKS;

LINK_NAME     LINK_TYPE
------------------------
MQLINK        MQSERIES
RVLINK        TIBRV
```

Example C-20 Checking Messaging System Link Configuration Information

```
SQL> select link_name, queue_manager, channel, hostname from mgw_mqseries_links;

LINK_NAME   QUEUE_MANAGER      CHANNEL     HOSTNAME
----------------------------------------------------------
MQLINK      my.queue.manager  mychannel  myhost.mydomain

SQL> select link_name, service, network, daemon from mgw_tibrv_links;

LINK_NAME   SERVICE      NETWORK     DAEMON
-----------------------------------------------------
RVLINK
```

Configuring Non-Oracle Messaging System Queues

All non-Oracle messaging system queues involved in propagation as a source queue, destination queue, or exception queue must be registered through the Messaging Gateway administration interface. You do not need to register Oracle Database Advanced Queuing queues involved in propagation.

This section contains these topics:

- Registering a Non-Oracle Queue
- Unregistering a Non-Oracle Queue
- View for Registered Non-Oracle Queues

Registering a Non-Oracle Queue

You can register a non-Oracle queue using `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE`. Registering a non-Oracle queue provides information for the Messaging Gateway agent to access the queue. However, it does not create the physical queue in the non-Oracle messaging system. The physical queue must be created using the non-Oracle messaging system administration interfaces before the Messaging Gateway agent accesses the queue.

The following information is used to register a non-Oracle queue:

- Name of the messaging system link used to access the queue
- Native name of the queue (its name in the non-Oracle messaging system)
- Domain of the queue
  - `DBMS_MGWADM.DOMAIN_QUEUE` for a point-to-point queue
  - `DBMS_MGWADM.DOMAIN_TOPIC` for a publish/subscribe queue
- Options specific to the non-Oracle messaging system

  These options are a set of {name, value} pairs, both of which are strings.

See Also:

"Optional Foreign Queue Configuration Properties"

Example C-21 shows how to register the WebSphere MQ Base Java queue `my_mq_queue` as a Messaging Gateway queue `destq`.

Example C-21 Registering a WebSphere MQ Base Java Queue

```
BEGIN
  DBMS_MGWADM.REGISTER_FOREIGN_QUEUE(
    name            => 'destq',
    linkname        => 'mqlink',
    provider_queue  => 'my_mq_queue',
    domain          => dbms_mgwadm.DOMAIN_QUEUE);
END;
```

Registering a WebSphere MQ Base Java Queue

The domain must be `DBMS_MGWADM.DOMAIN_QUEUE` or NULL, because only point-to-point queues are supported for WebSphere MQ.

Registering a WebSphere MQ JMS Queue or Topic

When registering a WebSphere MQ JMS queue, the domain must be `DBMS_MGWADM.DOMAIN_QUEUE`, and the `linkname` parameter must refer to a WebSphere MQ JMS unified link or queue link.

When registering a WebSphere MQ JMS topic, the domain must be `DBMS_MGWADM.DOMAIN_TOPIC`, and the `linkname` parameter must refer to a WebSphere MQ JMS unified link or topic link. The `provider_queue` for a WebSphere MQ JMS topic used as a propagation source may include wildcards. See WebSphere MQ documentation for wildcard syntax.

Registering a TIB/Rendezvous Subject

When registering a TIB/Rendezvous subject with Messaging Gateway, the `provider_queue` parameter specifies a TIB/Rendezvous subject name. The domain of a registered TIB/Rendezvous queue must be `DBMS_MGWADM.DOMAIN_TOPIC` or NULL.

A registered TIB/Rendezvous queue with `provider_queue` set to a wildcard subject name can be used as a propagation source queue for inbound propagation. It is not recommended to use queues with wildcard subject names as propagation destination queues or exception queues. As documented in TIB/Rendezvous, sending messages to wildcard subjects can trigger unexpected behavior. However, neither Messaging Gateway nor TIB/Rendezvous prevents you from doing so.

Unregistering a Non-Oracle Queue

A non-Oracle queue can be unregistered with `DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE`, but only if there are no propagation jobs referencing it.

Example C-22 unregisters the queue `destq` of the link `mqlink`.

Example C-22   Unregistering a Non-Oracle Queue

```
BEGIN
  DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE(name =>'destq', linkname=>'mqlink');
END;
```

View for Registered Non-Oracle Queues

You can use the `MGW_FOREIGN_QUEUES` view to check which non-Oracle queues are registered and what link each uses, as shown in Example C-23.

Example C-23   Checking Which Queues Are Registered

```
SELECT name, link_name, provider_queue FROM MGW_FOREIGN_QUEUES;

NAME   LINK_NAME   PROVIDER_QUEUE
------------------------------------
DESTQ  MQLINK       my_mq_queue
```

Configuring Oracle Messaging Gateway Propagation Jobs

Propagating messages between an Oracle Database AQ queue and a non-Oracle messaging system queue requires a propagation job. Each propagation job will have a unique propagation type, source, and destination triplet.

You can create a propagation job to propagate messages between JMS destinations. You can also create a propagation job to propagate messages between non-JMS queues. Messaging Gateway does not support message propagation between a JMS destination and a non-JMS queue.

This section contains these topics:

- Propagation Job Overview
- Creating an Oracle Messaging Gateway Propagation Job
- Enabling and Disabling a Propagation Job
- Resetting a Propagation Job
- Altering a Propagation Job
- Removing a Propagation Job

Propagation Job Overview

A propagation job specifies what messages are propagated and how the messages are propagated.

Messaging Gateway allows bidirectional message propagation. An outbound propagation moves messages from Oracle Database Advanced Queuing to non-Oracle messaging systems. An inbound propagation moves messages from non-Oracle messaging systems to Oracle Database Advanced Queuing.

If the propagation source is a queue (point-to-point), then the Messaging Gateway agent moves all messages from the source queue to the destination queue. If the propagation source is a topic (publish/subscribe), then the Messaging Gateway agent creates a subscriber of the propagation source queue in the messaging system. The agent only moves messages that are published to the source queue after the subscriber is created.

When propagating a message, the Messaging Gateway agent converts the message from the format in the source messaging system to the format in the destination messaging system. Users can customize the message conversion by providing a message transformation. If message conversion fails, then the message will be moved to an exception queue, if one has been provided, so that the agent can continue to propagate messages for the subscriber.

A Messaging Gateway exception queue is different from an Oracle Database Advanced Queuing exception queue. Messaging Gateway moves a message to a Messaging Gateway exception queue when message conversion fails. Oracle Database Advanced Queuing moves a message to an Oracle Database Advanced Queuing exception queue after `MAX_RETRIES` dequeue attempts on the message.

Messages moved to an Oracle Database Advanced Queuing exception queue may result in irrecoverable failures on the associated Messaging Gateway propagation job. To avoid the problem, the `MAX_RETRIES` parameter of any Oracle Database Advanced Queuing queue that is used as the propagation source of a Messaging Gateway propagation job should be set to a value much larger than 16.

If the messaging system of the propagation source queue supports message selection, then a message selection rule can be specified for a propagation subscriber. Only messages that satisfy the message selector will be propagated.

Users can also specify propagation job options to control how messages are propagated, such as options for JMS message delivery mode and TIB/Rendezvous queue policies.

The `MGW_JOBS` view can be used to check the configuration and status of Messaging Gateway propagation jobs.

See Also:

Monitoring Oracle Messaging Gateway

Creating an Oracle Messaging Gateway Propagation Job

Messaging Gateway propagation jobs are created by `DBMS_MGWADM.CREATE_JOB`.

If the propagation source for non-JMS propagation is an Oracle Database AQ queue, then the queue can be either a single consumer queue or multiple consumer queue. If it is a multiple consumer queue, Messaging Gateway creates a corresponding Oracle Database AQ subscriber `MGW_job_name` for the propagation job `job_name` when `DBMS_MGWADM.CREATE_JOB` is called.

If the propagation source is a JMS topic, such as an Oracle Java Message Service (Oracle JMS) topic or a WebSphere MQ JMS topic, then a JMS subscriber `MGW_job_name` is created on the topic in the source messaging system by the Messaging Gateway agent. If the agent is not running, then the subscriber will not be created until the agent is restarted.

If the propagation source is a queue, then only one propagation job can be created using that queue as the propagation source. If the propagation source is a topic, then multiple propagation jobs can be set up using that topic as the propagation source with each propagation job having its own corresponding subscriber on the topic in the messaging system.

Example C-24 creates Messaging Gateway propagation job `job_aq2mq`.

Note:

If a WebSphere MQ JMS topic is involved in a propagation job and the interface type of the link is `DBMS_MGWADM.JMS_TOPIC_CONNECTION`, then a durable subscriber `MGL_`*`subscriber_id`* is created on the log topic. The durable subscriber is removed when the Messaging Gateway propagation job is successfully removed.

Example C-24 Creating a Messaging Gateway Propagation Job

```
BEGIN
   DBMS_MGWADM.CREATE_JOB(
     job_name            =>  'job_aq2mq',
     propagation_type    =>   DBMS_MGWADM.OUTBOUND_PROPAGATION,
     source              =>  'mquser.srcq',
     destination         =>  'deqq@mqlink');
END;
```

Enabling and Disabling a Propagation Job

A propagation job can be initially enabled or disabled when it is created by `DBMS_MGWADM.CREATE_JOB`. By default, a job is enabled when it is created. You can use `DBMS_MGWADM.ENABLE_JOB` to enable a propagation job and `DBMS_MGWADM.DISABLE_JOB` to disable a job. No propagation processing will occur when the job is disabled.

Example C-25 enables the propagation for propagation job `job_aq2mq`.

Example C-26 disables the propagation for propagation job `job_aq2mq`.

Example C-25 Enabling a Messaging Gateway Propagation Job

```
BEGIN
  DBMS_MGWADM.ENABLE_JOB(job_name => 'job_aq2mq');
END;
```

Example C-26 Disabling a Messaging Gateway Propagation Job

```
BEGIN
  DBMS_MGWADM.DISABLE_JOB(job_name => 'job_aq2mq');
END;
```

Resetting a Propagation Job

When a problem occurs with a propagation job, the Messaging Gateway agent retries the failed operation up to 16 times in an exponential backoff scheme before the propagation job stops. You can use `DBMS_MGWADM.RESET_JOB` to reset the failure count to zero to allow the agent to retry the failed operation immediately.

Example C-27 resets the failure count for propagation job `job_aq2mq`.

Example C-27   Resetting a Propagation Job

```
BEGIN
  DBMS_MGWADM.RESET_JOB (job_name => 'job_aq2mq');
END;
```

Altering a Propagation Job

After a propagation job is created you can alter the selection rule, transformation, exception queue, job options, and poll interval of the job using `DBMS_MGWADM.ALTER_JOB`. The job can be altered with the Messaging Gateway running or shut down.

Example C-28 adds an exception queue for a propagation job.

Example C-29 changes the polling interval for a propagation job. The polling interval determines how soon the agent can discover the available messages in the propagation source queue. The default polling interval is 5 seconds or the value set for `oracle.mgw.polling_interval` in the Messaging Gateway initialization file.

Example C-28   Altering Propagation Job by Adding an Exception Queue

```
BEGIN
    DBMS_MGWADM.ALTER_JOB(
       job_name         =>  'job_aq2mq',        exception_queue  =>
'mgwuser.my_ex_queue');
END;
```

Example C-29   Altering Propagation Job by Changing the Polling Interval

```
BEGIN
    DBMS_MGWADM.ALTER_JOB(
        job_name       =>  'job_aq2mq',         poll_interval  =>  2);
END;
```

Removing a Propagation Job

You can remove a Messaging Gateway propagation job by calling `DBMS_MGWADM.REMOVE_JOB`.

Before removing the propagation job from the Messaging Gateway configuration, Messaging Gateway does the following cleanup:

- Removes from the messaging system the associated subscriber that may have been created by Messaging Gateway
- Removes propagation log records from log queues for the job being removed

Messaging Gateway may fail to do the cleanup because:

- The Messaging Gateway agent is not running
- Non-Oracle messaging system is not running
- The Messaging Gateway agent cannot interact with the source or destination messaging system

If the Messaging Gateway cleanup fails for any reason, then the propagation job being removed is placed in a `DELETE_PENDING` state. The Messaging Gateway agent tries to clean up propagation jobs in a `DELETE_PENDING` state when:

- `DBMS_MGWADM.REMOVE_JOB` is called and the Messaging Gateway agent is running.
- The Messaging Gateway agent is starting and finds a propagation job in a `DELETE_PENDING` state.

DBMS_MGWADM.REMOVE_JOB has a force parameter that allows you to force the propagation job to be removed from the Messaging Gateway configuration without placing it in DELETE_PENDING state. This is useful in case of cleanup failures or if you want to remove a propagation job when the Messaging Gateway agent is not running.

Forcing a propagation job to be removed may result in obsolete log records being left in the log queues, and subscriptions in the messaging systems that may cause unnecessary message accumulation. Oracle recommends that the force option not be used for DBMS_MGWADM.REMOVE_JOB if possible.

Example C-30 removes a propagation job in a non-forced manner.

Example C-30 Removing a Propagation Job

```
BEGIN
    DBMS_MGWADM.REMOVE_JOB (job_name  =>  'job_aq2mq');
END;
```

Propagation Jobs, Subscribers, and Schedules

Subprograms are provided as part of the DBMS_MGWADM package that simplify the creation and management of propagation jobs. Those subprograms allow a user to configure a propagation job rather than a disjoint subscriber and schedule as was done in prior releases. Oracle recommends that you use the propagation job procedures but still supports the subscriber and schedule procedures for backward compatibility.

Table C-2 lists the Messaging Gateway propagation job procedures and shows which subscriber and/or schedule procedures it replaces. All procedures are from the DBMS_MGWADM package.

Table C-2 Messaging Gateway Propagation Job Subprograms

Job Procedure	Replaces Subscriber, Schedule Procedure
CREATE_JOB	ADD_SUBSCRIBER, SCHEDULE_PROPAGATION
ALTER_JOB	ALTER_SUBSCRIBER, ALTER_PROPAGATION_SCHEDULE
REMOVE_JOB	REMOVE_SUBSCRIBER, UNSCHEDULE_PROPAGATION
ENABLE_JOB	ENABLE_PROPAGATION_SCHEDULE
DISABLE_JOB	DISABLE_PROPAGATION_SCHEDULE
RESET_JOB	RESET_SUBSCRIBER

This section contains the following topics:

- Propagation Job_ Subscriber_ Schedule Interface Interoperability
- Propagation Job_ Subscriber_ Schedule Views
- Single Consumer Queue as Propagation Source

Propagation Job, Subscriber, Schedule Interface Interoperability

The user can create two types of propagation jobs, a new style job or an old style job. A new style job is created by DBMS_MGWADM.CREATE_JOB. An old style job is created by calling DBMS_MGWADM.ADD_SUBSCRIBER and DBMS_MGWADM.SCHEDULE_PROPAGATION using the same {propagation_type, source, destination} triplet. A subscriber that does not have a matching schedule, or a schedule that does not have a matching subscriber, is not considered to be a propagation job.

For new style job, the job name will serve as both the subscriber ID and the schedule ID. For an old style job, the subscriber ID is used as the job name.

Both the propagation job subprograms and the subscriber/schedule subprograms can be used for old style propagation jobs. Oracle recommends that you use the job subprograms to create and manage propagation jobs. The job subprograms cannot be used for an unmatched subscriber or schedule since those do not constitute a propagation job.

Only the new job subprograms can be used for new style propagation jobs. An error will occur if a user tries to call a subscriber or scheduler procedure on a new style job.

Other than `DBMS_MGWADM.REMOVE_JOB`, calling the job subprograms for an old style job is straightforward and the results are effectively the same as calling the corresponding subscriber/schedule subprograms. There may be certain restrictions in the future but there are none at this time.

The `DBMS_MGWADM.REMOVE_JOB` procedure can be used to remove both new style and old style jobs. A forced and non-forced remove is supported. If the Messaging Gateway agent is not running when a non-forced remove is done, the job will be flagged as delete pending and neither the underlying subscriber nor schedule will be removed at that time. The job (subscriber /schedule pair) will be removed once the agent is restarted and performs its cleanup work or a forced `DBMS_MGWADM.REMOVE_JOB` is performed. In order to insure that the subscriber/schedule pair is removed at the same time, an error will occur if you first call `DBMS_MGWADM.REMOVE_JOB` and subsequently attempt to call `DBMS_MGWADM.REMOVE_SUBSCRIBER` or `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` for an old style job.

Once `DBMS_MGWADM.REMOVE_JOB` as been called for a job and it has been flagged as delete pending, all job procedures, other than `DBMS_MGWADM.REMOVE_JOB`, will fail for both new style and old style jobs. In addition, all subscriber and schedule subprograms will fail if the propagation job happens to be an old style job.`DBMS_MGWADM.REMOVE_SUBSCRIBER` and `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` can be used for an old style job as long as `DBMS_MGWADM.REMOVE_JOB` has not been called for that job. If `DBMS_MGWADM.UNSCHEDULE_PROPAGATION` is called for an old style job, the schedule is immediately removed and it ceases to be a propagation job and `DBMS_MGWADM.REMOVE_SUSCRIBER` must be used to remove the subscriber. If `DBMS_MGWADM.REMOVE_SUBSCRIBER` is called for an old style job, the user can subsequently call `DBMS_MGWADM.REMOVE_JOB` as long as the subscriber exists.

Propagation Job, Subscriber, Schedule Views

The `MGW_JOBS` view shows information for the current propagation jobs, both new style jobs and old style jobs, and includes all the pertinent information shown by the `MGW_SUBSCRIBERS` and `MGW_SCHEDULES` views. The `MGW_SUBSCRIBERS` and `MGW_SCHEDULES` views are still useful for finding an unmatched subscriber or schedule since they don't constitute a propagation job and will not show up in the `MGW_JOBS` view.

Single Consumer Queue as Propagation Source

Messaging Gateway allows an Oracle Database AQ multiple consumer queue or a single consumer queue to be a propagation source for an outbound new style job created by `DBMS_MGWADM.CREATE_JOB`. A multiple consumer queue must be used for the propagation source for an outbound old style job. An error will occur if an administrator attempts to call `DBMS_MGWADM.ADD_SUBSCRIBER` and the source is a single consumer queue.

An Oracle Database AQ dequeue condition is not supported for native (non-JMS) outbound propagation when the propagation source is a single consumer queue.

Configuration Properties

This section summarizes basic and optional properties related to Messaging Gateway links, foreign queues, and propagation jobs.

This section contains these topics:

WebSphere MQ System Properties

Table C-3 summarizes the basic configuration properties for a WebSphere MQ messaging link. The table indicates which properties of `SYS.MGW_MQSERIES_PROPERTIES` are optional (`NULL` allowed), which can be altered, and if alterable, which values can be dynamically changed.

See Also:

"SYS.MGW_MQSERIES_PROPERTIES Type" in *Oracle Database PL/SQL Packages and Types Reference*

Table C-3 WebSphere MQ Link Properties

Attribute	NULL Allowed?	Alter Value?	Dynamic?
`queue_manager`	no	no	no
`hostname`	yes (1)	no	no
`port`	yes (1)	no	no
`channel`	yes (1), (6)	yes	no
`interface_type`	yes (2)	no	no
`max_connections`	yes (3)	yes	yes
`username`	yes	yes	yes
`password`	yes	yes	yes
`inbound_log_queue`	yes (4)	yes(4)	yes
`outbound_log_queue`	yes (5)	yes(5)	yes

Notes on Table C-3

- If `hostname` is `NULL`, then the port and channel must be `NULL`. If the hostname is not `NULL`, then the port must be not `NULL`. If the hostname is `NULL`, then a WebSphere MQ bindings connection is used; otherwise a client connection is used.

- If `interface_type` is `NULL`, then a default value of `DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE` is used.
- If `max_connections` is `NULL`, then a default value of `1` is used.
- Attribute `inbound_log_queue` can be `NULL` if the link is not used for inbound propagation. The log queue can be altered only when no inbound propagation job references the link.
- Attribute `outbound_log_queue` can be `NULL` if the link is not used for outbound propagation. The log queue can be altered only when no outbound propagation job references the link.
- The channel attribute must be `NULL` if a client channel definition table (CCDT) is used. The `MQ_ccdtURL` link option can be used to specify a CCDT.

Table C-4 summarizes the optional configuration properties supported when a WebSphere MQ Base Java interface is used to access the WebSphere MQ messaging system. Table C-5 summarizes the optional configuration properties supported when a WebSphere MQ JMS interface is used. Each table lists the property name, where that property applies, whether the property can be altered, and if alterable, whether the value can be dynamically changed. Only the properties listed in the tables are supported, and any extra properties are ignored.

Table C-4 Optional Configuration Properties for WebSphere MQ Base Java

Property Name	Used For	Alter Value?	Dynamic?
MQ_ccdtUrl	link	yes	no
MQ_ccsid	link	yes	no
MQ_ReceiveExit	link	yes	no
MQ_SecurityExit	link	yes	no
MQ_SendExit	link	yes	no
MQ_SSLCipherSuite	link	yes	no
MQ_SSLFipsRequired	link	yes	no
MQ_SSLPeerName	link	yes	no
MQ_SSLResetCount	link	yes	no
MQ_openOptions	foreign queue	no	no
MsgBatchSize	job	yes	yes
PreserveMessageID	job	yes	yes

Table C-5 Optional Configuration Properties for WebSphere MQ JMS

Property Name	Used For	Alter Value?	Dynamic?
MQ_BrokerControlQueue	link	yes	no
MQ_BrokerPubQueue	link	yes	no
MQ_BrokerQueueManager	link	yes	no
MQ_BrokerVersion	link	yes	no
MQ_ccdtUrl	link	yes	no
MQ_ccsid	link	yes	no
MQ_JmsDurSubQueue	link	no	no
MQ_PubAckInterval	link	yes	no
MQ_ReceiveExit	link	yes	no

Table C-5 (Cont.) Optional Configuration Properties for WebSphere MQ JMS

Property Name	Used For	Alter Value?	Dynamic?
MQ_ReceiveExitInit	link	yes	no
MQ_SecurityExit	link	yes	no
MQ_SecurityExitInit	link	yes	no
MQ_SendExit	link	yes	no
MQ_SendExitInit	link	yes	no
MQ_SSLCipherSuite	link	yes	no
MQ_SSLCrl	link	yes	no
MQ_SSLFipsRequired	link	yes	no
MQ_SSLPeerName	link	yes	no
MQ_SSLResetCount	link	yes	no
MQ_CharacterSet	foreign queue	no	no
MQ_JmsDurSubQueue	foreign queue	no	no
MQ_JmsTargetClient	foreign queue	no	no
JMS_DeliveryMode	job	yes	yes
JMS_NoLocal	job	no	no
MsgBatchSize	job	yes	yes
PreserveMessageID	job	yes	yes

TIB/Rendezvous System Properties

Table C-6 summarizes the basic configuration properties for a TIB/Rendezvous messaging link. It indicates which properties of `SYS.MGW_TIBRV_PROPERTIES` are optional (`NULL` allowed), which can be altered, and if alterable, which values can be dynamically changed.

See Also:

"SYS.MGW_TIBRV_PROPERTIES Type" in *Oracle Database PL/SQL Packages and Types Reference*

Table C-6 TIB/Rendezvous Link Properties

Attribute	NULL allowed?	Alter value?	Dynamic?
`service`	yes(1)	no	no
`daemon`	yes(1)	no	no
`network`	yes(1)	no	no
`cm_name`	yes(2)	no	no
`cm_ledger`	yes(2)	no	no

Notes on Table C-6:

- System default values will be used if `service`, `daemon`, or `network` are `NULL`.
- The `cm_name` and `cm_ledger` attributes are reserved for future use when TIB/Rendezvous certified messages are supported. At present, a `NULL` must be specified for these parameters when a TIB/Rendezvous link is configured.

Table C-7 summarizes the optional configuration properties supported when a TIB/Rendezvous messaging system is used. The table lists the property name, where that property applies, whether the property can be altered, and if alterable, whether the value can be dynamically changed. Only the properties listed in the table are supported, and any extra properties will be ignored.

Table C-7 Optional Properties for TIB/Rendezvous

Property Name	Used For	Alter Value?	Dynamic?
AQ_MsgProperties	job	yes	yes
MsgBatchSize	job	yes	yes
PreserveMessageID	job	yes	yes
RV_discardAmount	job	yes	no
RV_limitPolicy	job	yes	no
RV_maxEvents	job	yes	no

Optional Link Configuration Properties

This section describes optional link properties you can specify using the `options` parameter of `DBMS_MGWADM.CREATE_MSGSYSTEM_LINK` and `DBMS_MGWADM.ALTER_MSGSYSTEM_LINK`. Each listing also indicates which messaging system might use that property.

MQ_BrokerControlQueue
This property is used by WebSphere MQ JMS. It specifies the name of the broker control queue and corresponds to WebSphere MQ JMS administration tool property `BROKERCONQ`. The WebSphere MQ default is `SYSTEM.BROKER.CONTROL.QUEUE`.

MQ_BrokerPubQueue
This property is used by WebSphere MQ JMS. It specifies the name of the broker publish queue and corresponds to WebSphere MQ JMS administration tool property `BROKERPUBQ`. The WebSphere MQ default is `SYSTEM.BROKER.DEFAULT.STREAM`.

MQ_BrokerQueueManager
This property is used by WebSphere MQ JMS. It specifies the name of the broker queue manager and corresponds to WebSphere MQ administration tool property `BROKERQMGR`. If it is not set, then no default is used.

MQ_BrokerVersion
This property is used by WebSphere MQ JMS. It specifies the broker version number and corresponds to WebSphere MQ JMS administration tool property `BROKERVER`. The WebSphere MQ default is `0`.

MQ_ccdtUrl
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the URL string of a client channel definition table (CCDT) to be used. If not set, a CCDT is not used. If a CCDT is used, then the `SYS.MGW_MQSERIES_PROPERTIES.channel` link property must be `NULL`.

MQ_ccsid
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the character set identifier to be used to translate information in the WebSphere MQ message header. This should be the integer value of the character set (for example, `819`) rather than a descriptive string. If it is not set, then the WebSphere MQ default character set `819` is used.

MQ_JmsDurSubQueue
This property is used by WebSphere MQ JMS. It applies to WebSphere MQ JMS topic links only. The `SYS.MGW_MQSERIES_PROPERITES` attributes, `inbound_log_queue` and `outbound_log_queue`, specify the names of WebSphere MQ JMS topics used for propagation logging. This property specifies the name of the WebSphere MQ queue from which durable subscription messages are retrieved by the log topic subscribers. The WebSphere MQ default queue is `SYSTEM.JMS.D.SUBSCRIBER.QUEUE`.

MQ_PubAckInterval
This property is used by WebSphere MQ JMS. It specifies the interval, in number of messages, between publish requests that require acknowledgment from the broker and corresponds to WebSphere MQ JMS administration tool property `PUBACKINT`. The WebSphere MQ default is `25`.

MQ_ReceiveExit
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the `MQReceiveExit` interface. This class must be in the `CLASSPATH` of the Messaging Gateway agent. There is no default.

MQ_ReceiveExitInit
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by `MQ_ReceiveExit` and corresponds to WebSphere MQ JMS administration tool property `RECEXITINIT`. There is no default.

MQ_SecurityExit
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the `MQSecurityExit` interface. This class must be in the `CLASSPATH` of the Messaging Gateway agent. There is no default.

MQ_SecurityExitInit
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by `MQ_SecurityExit` and corresponds to WebSphere MQ JMS administration tool property `SECEXITINIT`. There is no default.

MQ_SendExit
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the fully qualified Java classname of a class implementing the `MQSendExit` interface. This class must be in the `CLASSPATH` of the Messaging Gateway agent. There is no default.

MQ_SendExitInit
This initialization string is used by WebSphere MQ JMS. It is passed by WebSphere MQ JMS to the constructor of the class specified by `MQ_SendExit`. It corresponds to WebSphere MQ JMS administration tool property `SENDEXITINIT`. There is no default.

MQ_SSLCipherSuite
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the CipherSuite to be used; for example, `SSL_RSA_WITH_RC4_128_MD5`. This corresponds to the WebSphere MQ `SSLCIPHERSUITE` administration property.

MQ_SSLCrl
This property is used by WebSphere MQ JMS. It specifies a space-delimited list of LDAP servers that can be used for certificate revocation list (CRL) checking. If not set, no CRL checking is done. This corresponds to the WebSphere MQ `SSLCRL` administration property. This option is not supported for WebSphere MQ Base Java, and instead, a client channel definition table (CCDT) must be used if CRL checking is needed.

MQ_SSLFipsRequired
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It indicates whether the CipherSuite being used is supported by the IBM Java JSSE FIPS provider (IBMSJSSEFIPS). The value should be `TRUE` or `FALSE`. The default value is `FALSE`. This corresponds to the WebSphere MQ `SSLFIPSREQUIRED` administration property.

MQ_SSLPeerName
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies a distinguished name (DN) pattern that the queue manager certificate must match in order for a connection to be established. If not set, no DN check is performed. This corresponds to the WebSphere MQ `SSLPEERNAME` administration property.

MQ_SSLResetCount
This property is used by WebSphere MQ Base Java and WebSphere MQ JMS. It specifies the total number of bytes sent and received before the secret key is renegotiated. If not set, the key is not renegotiated. This corresponds to the WebSphere MQ `SSLRESETCOUNT` administration property.

Optional Foreign Queue Configuration Properties

This section describes optional foreign queue properties that you can specify using the `options` parameter of `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE`. Each listing also indicates which messaging system might use that property.

MQ_CharacterSet
This property is used by WebSphere MQ JMS. It is used only for outbound propagation to a JMS queue or topic. It specifies the character set to be used to encode text strings sent to the destination. It should be the integer value of the character set (for example, `1208`) rather than a descriptive string. The default value used by Messaging Gateway is `1208` (UTF8).

MQ_JmsDurSubQueue
This property is used by WebSphere MQ JMS. It is a string representing the name of the WebSphere MQ queue from which durable subscription messages are retrieved by subscribers on this topic. It applies only to WebSphere MQ JMS topics. The WebSphere MQ default queue is `SYSTEM.JMS.D.SUBSCRIBER.QUEUE`.

MQ_JmsTargetClient
This property is used by WebSphere MQ JMS. It is used only for outbound propagation to a JMS queue or topic. Supported values are `TRUE` and `FALSE`. `TRUE` indicates that WebSphere MQ should store the message as a JMS message. `FALSE` indicates that WebSphere MQ should store the message in non-JMS format so that non-JMS applications can access it. Default is `TRUE`.

MQ_openOptions
This property is used by WebSphere MQ Base Java. It specifies the value used for the `openOptions` argument of the WebSphere MQ Base Java `MQQueueManager.accessQueue` method. No value is required. But if one is given, then the Messaging Gateway agent adds `MQOO_OUTPUT` to the specified value for an enqueue (`put`) operation. `MQOO_INPUT_SHARED` is added for a dequeue (`get`) operation. The default is `MQOO_OUTPUT` for an enqueue/put operation; `MQOO_INPUT_SHARED` for a dequeue/get operation.

Optional Job Configuration Properties

This section describes optional propagation job properties that you can specify using the options parameter of `DBMS_MGWADM.CREATE_JOB` and `DBMS_MGWADM.ALTER_JOB`.

AQ_MsgProperties
This property is used by TIB/Rendezvous. It specifies how Oracle Database AQ message properties will be used during message propagation. Supported values are `TRUE` and `FALSE`. The default value is `FALSE`.
For an outbound propagation job, if the value is `TRUE` (case insensitive), then the Messaging Gateway agent will add a field for most Oracle Database AQ message properties to the message propagated to the TIB/Rendezvous subject.
For an inbound propagation job, if the value is `TRUE` (case insensitive), then the Messaging Gateway agent will search the source message for a field with a reserved name, and if it exists, use its value to set the corresponding Oracle Database AQ message property. A default value will be used if the field does not exist or does not have an expected datatype.

JMS_DeliveryMode
This property is used by WebSphere MQ JMS and Oracle JMS. You can use this property when the propagation destination is a JMS messaging system. It sets the delivery mode of messages enqueued to the propagation destination queue by a JMS `MessageProducer`. The default is `PRESERVE_MSG`. Supported values and their associated delivery modes are:

- `PERSISTENT` (`DeliveryMode.PERSISTENT`)
- `NON_PERSISTENT` (`DeliveryMode.NON_PERSISTENT`)
- `PRESERVE_MSG` (delivery mode of the source JMS message is used)

JMS_NoLocal
This property is used by WebSphere MQ JMS and Oracle JMS. You can use it when the propagation source is a JMS messaging system. It sets the `noLocal` parameter of a JMS `TopicSubscriber`. `TRUE` indicates that messages that have been published to this topic through the same Messaging Gateway link will not be propagated. The default value `FALSE` indicates that such messages will be propagated from the topic.

MsgBatchSize
This property can be used by any supported messaging system. It specifies the maximum number of messages, if available, to be propagated in one transaction. The default is 30.

PreserveMessageID
This property is used by WebSphere MQ Base Java, WebSphere MQ JMS, TIB/Rendezvous, and Oracle JMS. It specifies whether Messaging Gateway should preserve the original message identifier when the message is propagated to the destination messaging system. The exact details depend on the capabilities of the messaging systems involved. Supported values are `TRUE` and `FALSE`. The default value is `FALSE`.

RV_discardAmount
This property is used by TIB/Rendezvous. It specifies the discard amount of a queue. It is meaningful only for an inbound propagation job. The default is 0.

RV_limitPolicy
This property is used by TIB/Rendezvous. It specifies the limit policy for resolving overflow of a queue limit. It is meaningful only for an inbound propagation job. The default is `DISCARD_NONE`. Supported values and their associated limit policies are: `DISCARD_NONE`, `DISCARD_FIRST`, `DISCARD_LAST` and `DISCARD_NEW`.

- `DISCARD_NONE` (`TibrvQueue.DISCARD_NONE`)
- `DISCARD_FIRST` (`TibrvQueue.DISCARD_FIRST`)
- `DISCARD_LAST` (`TibrvQueue.DISCARD_LAST`)
- `DISCARD_NEW` (`TibrvQueue.DISCARD_NEW`)

RV_maxEvents
This property is used by TIB/Rendezvous. It specifies the maximum event limit of a queue. It is meaningful only for an inbound propagation job. The default is 0.

Oracle Messaging Gateway Message Conversion

The following topics discuss how Oracle Messaging Gateway (MGW) converts message formats from one messaging system to another. A conversion is generally necessary when moving messages between Oracle Database Advanced Queuing and another system, because different messaging systems have different message formats. Java Message Service (JMS) messages are a special case. A JMS message can be propagated only to a JMS destination, making conversion a simple process.

- Converting Oracle Messaging Gateway Non-JMS Messages
- Message Conversion for WebSphere MQ
- Message Conversion for TIB/Rendezvous
- JMS Messages

Converting Oracle Messaging Gateway Non-JMS Messages

MGW converts the native message format of the source messaging system to the native message format of the destination messaging system during propagation. MGW uses canonical types and a model centering on Oracle Database Advanced Queuing for the conversion.

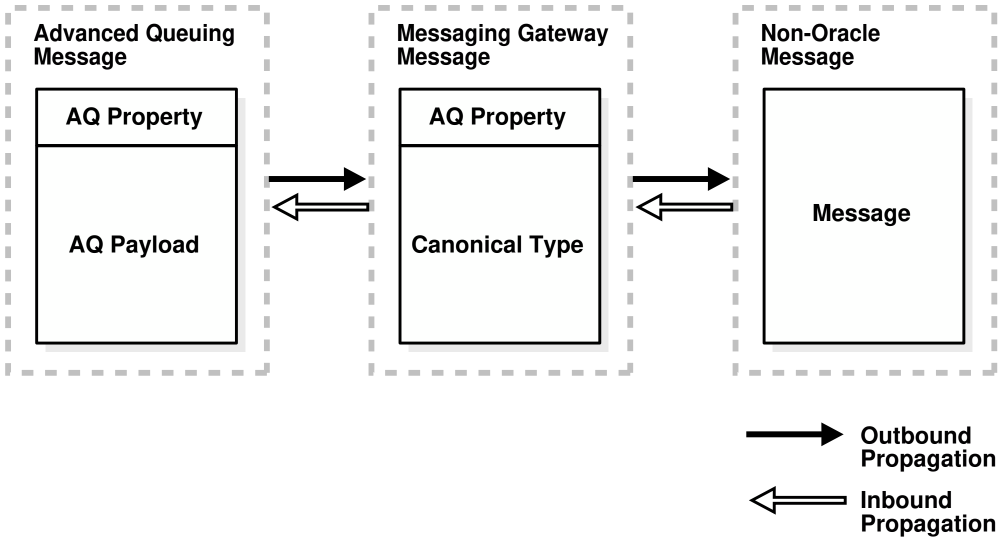
Overview of the Non-JMS Message Conversion Process

When a message is propagated by MGW, the message is converted from the native format of the source queue to the native format of the destination queue.

A native message usually contains a message header and a message body. The header contains the fixed header fields that all messages in that messaging system have, such as message properties in Oracle Database Advanced Queuing and the fixed header in WebSphere MQ. The body contains message contents, such as the Oracle Database Advanced Queuing payload, the WebSphere MQ message body, or the entire TIB/Rendezvous message. MGW converts both message header and message body components.

Figure C-2 shows how non-JMS messages are converted in two stages. A message is first converted from the native format of the source queue to the MGW internal message format, and then it is converted from the internal message format to the native format of the destination queue.

Figure C-2 Non-JMS Message Conversion



The MGW agent uses an internal message format consisting of a header that is similar to the Oracle Database Advanced Queuing message properties and a body that is a representation of an MGW canonical type.

Oracle Messaging Gateway Canonical Types

MGW defines canonical types to support message conversion between Oracle Database Advanced Queuing and non-Oracle messaging systems. A canonical type is a message type representation in the form of a PL/SQL Oracle type in Oracle Database. The canonical types are `RAW`, `SYS.MGW_BASIC_MSG_T`, and `SYS.MGW_TIBRV_MSG_T`.

WebSphere MQ propagation supports the canonical types `SYS.MGW_BASIC_MSG_T` and `RAW`. TIB/Rendezvous propagation supports the canonical types `SYS.MGW_TIBRV_MSG_T` and `RAW`.

See Also:

"DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference* for Syntax and attribute information for `SYS.MGW_BASIC_MSG_T` and `SYS.MGW_TIBRV_MSG_T`

Message Header Conversion

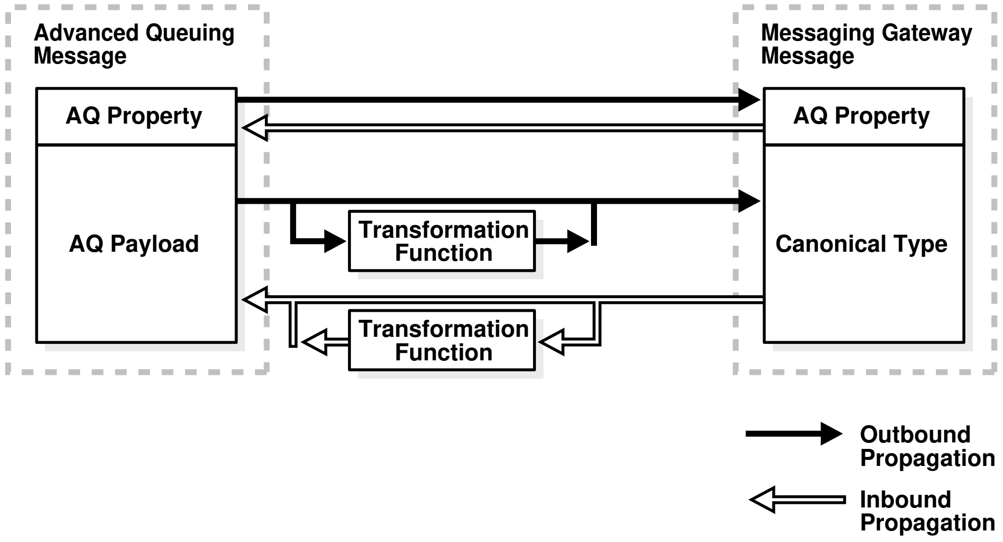
MGW provides default mappings between Oracle Database Advanced Queuing message properties and non-Oracle message header fields that have a counterpart in Oracle Database Advanced Queuing message properties with the same semantics. Where MGW does not provide a mapping, the message header fields are set to a default value, usually the default value defined by the messaging system.

Handling Arbitrary Payload Types Using Message Transformations

When converting to or from Oracle Database Advanced Queuing messages, the MGW agent uses only its canonical types. Arbitrary payload types are supported, however, with the assistance of user-defined Oracle Database Advanced Queuing message transformations to convert between an Oracle Database Advanced Queuing queue payload and an MGW canonical type.

For MGW to propagate messages from an Oracle Database Advanced Queuing queue with an arbitrary ADT payload (outbound propagation), you must provide a mapping to an MGW canonical ADT. The transformation is invoked when the MGW agent dequeues messages from the Oracle Database Advanced Queuing queue. Similarly, for MGW to propagate messages to an Oracle Database Advanced Queuing queue with an arbitrary ADT payload (inbound propagation), you must provide a mapping from an MGW canonical ADT. The transformation is invoked when the MGW agent enqueues messages to the Oracle Database Advanced Queuing queue.

Figure C-3   Oracle Database Advanced Queuing Message Conversion



The transformation is always executed in the context of the MGW agent, which means that the MGW agent user (the user specified using `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT`) must have `EXECUTE` privileges on the transformation function and the Oracle Database Advanced Queuing payload type. This can be accomplished by granting the `EXECUTE` privilege to `PUBLIC` or by granting the `EXECUTE` privilege directly to the MGW agent user.

To configure a MGW propagation job with a transformation:

1. Create the transformation function.
2. Grant `EXECUTE` to the MGW agent user or to `PUBLIC` on the function and the object types it references.
3. Call `DBMS_TRANSFORM.CREATE_TRANSFORMATION` to register the transformation.
4. Call `DBMS_MGWADM.CREATE_JOB` to create a MGW propagation job using the transformation, or `DBMS_MGWADM.ALTER_JOB` to alter an existing job.

Example C-31 Transformation Function Signature

```
FUNCTION trans_sampleadt_to_mgw_basic(in_msg IN mgwuser.sampleADT)
RETURN SYS.MGW_BASIC_MSG_T;
```

You can create a transformation using `DBMS_TRANSFORM.CREATE_TRANSFORMATION`, as shown in Example C-32.

Example C-32 Creating a Transformation

```
BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
        schema         => 'mgwuser',
        name           => 'sample_adt_to_mgw_basic',
        from_schema    => 'mgwuser',
        from_type      => 'sampleadt',
        to_schema      => 'sys',
        to_type        => 'MGW_BASIC_MSG_T',
        transformation => 'mgwuser.trans_sampleadt_to_mgw_basic(user_data)');
END;
```

Example C-33 Registering a Transformation

```
BEGIN
  DBMS_MGWADM.CREATE_JOB(
    job_name          =>    'job_aq2mq',
    propagation_type  =>     DBMS_MGWADM.OUTBOUND_PROPAGATION,
    source            =>    'mgwuser.srcq',
    destination       =>    'destq.mqlink',
    transformation    =>    'mgwuser.sample_adt_to_mgw_basic',
    exception_queue   =>    'mgwuser.excq');
END;
```

The value passed in the transformation parameter for these APIs must be the registered transformation name and not the function name. For example, `trans_sampleadt_to_mgw_basic` is a stored procedure representing a transformation function with the signature shown in Example C-31.

Note:

All commands in the examples must be run as a user granted `MGW_ADMINISTRATOR_ROLE`, except for the commands to create transformations.

Once created, this transformation can be registered with MGW when creating a propagation job. Example C-33 creates job `job_aq2mq`, for whom messages are propagated from Oracle Database Advanced Queuing queue `mgwuser.srcq` to non-Oracle messaging system queue `destq@mqlink` using transformation `mgwuser.sample_adt_to_mgw_basic`.

An error that occurs while attempting a user-defined transformation is usually considered a message conversion exception, and the message is moved to the exception queue if it exists.

See Also:

"DBMS_MGWADM", "DBMS_MGWMSG", and "DBMS_TRANSFORM" in *Oracle Database PL/SQL Packages and Types Reference*

Handling Logical Change Records

MGW provides facilities to propagate Logical Change Records (LCRs). Routines are provided to help in creating transformations to handle the propagation of both row LCRs and DDL LCRs stored in queues with payload type `ANYDATA`. An LCR is propagated as an XML string stored in the appropriate message type.

Note:

For LCR propagation, you must load the XDB package.

Because Oracle Streams uses `ANYDATA` queues to store LCRs, an `ANYDATA` queue is the source for outbound propagation. The transformation must first convert the `ANYDATA` object containing an LCR into an XMLType object using the MGW routine `DBMS_MGWMSG.LCR_TO_XML`. If the `ANYDATA` object does not contain an LCR, then this routine raises an error. The XML document string of the LCR is then extracted from the XMLType and placed in the appropriate MGW canonical type (`SYS.MGW_BASIC_MSG_T` or `SYS.MGW_TIBRV_MSG_T`).

Example C-34 illustrates a simplified transformation used for LCR outbound propagation. The transformation converts an `ANYDATA` payload containing an LCR to a `SYS.MGW_TIBRV_MSG_T` object. The string representing the LCR as an XML document is put in a field named `ORACLE_LCR`.

For LCR inbound propagation, an MGW canonical type (`SYS.MGW_BASIC_MSG_T` or `SYS.MGW_TIBRV_MSG_T`) is the transformation source type. A string in the format of an XML document representing an LCR must be contained in the canonical type. The transformation function must extract the string from the message, create an XMLType object from it, and convert it to an `ANYDATA` object containing an LCR with the MGW routine `DBMS_MGWMSG.XML_TO_LCR`. If the original XML document does not represent an LCR, then this routine raises an error.

Example C-35 illustrates a simplified transformation used for LCR inbound propagation. The transformation converts a `SYS.MGW_TIBRV_MSG_T` object with a field containing an XML string representing an LCR to an `ANYDATA` object. The string representing the LCR as an XML document is taken from a field named `ORACLE_LCR`.

See Also:

- "DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference*
- *`ORACLE_HOME`*`/mgw/samples/lcr` for complete examples of LCR transformations

Example C-34 Outbound LCR Transformation

```
create or replace function any2tibrv(adata in anydata)
return SYS.MGW_TIBRV_MSG_T is
    v_xml   XMLType;
    v_text  varchar2(2000);
    v_tibrv sys.mgw_tibrv_msg_t;
```

```
BEGIN
    v_xml   := dbms_mgwmsg.lcr_to_xml(adata);
    -- assume the lcr is smaller than 2000 characters long.
    v_text  := v_xml.getStringVal();
    v_tibrv := SYS.MGW_TIBRV_MSG_T.CONSTRUCT;
    v_tibrv.add_string('ORACLE_LCR', 0, v_text);
    return v_tibrv;
END any2tibrv;
```

Example C-35   Inbound LCR Transformation

```
create or replace function tibrv2any(tdata in sys.mgw_tibrv_msg_t)
return anydata is
    v_field   sys.mgw_tibrv_field_t;
    v_xml     XMLType;
    v_text    varchar2(2000);
    v_any     anydata;
BEGIN
    v_field := tdata.get_field_by_name('ORACLE_LCR');
    -- type checking
    v_text  := v_field.text_value;
    -- assume it is not null
    v_xml  := XMLType.createXML(v_text);
    v_any  := dbms_mgwmsg.xml_to_lcr(v_xml);
    return v_any;
END tibrv2any;
```

Message Conversion for WebSphere MQ

MGW converts between the MGW canonical types and the WebSphere MQ native message format. WebSphere MQ native messages consist of a fixed message header and a message body. The message body is treated as either a `TEXT` value or `RAW` (bytes) value. The canonical types supported for WebSphere MQ propagation are `SYS.MGW_BASIC_MSG_T` and `RAW`.

Figure C-4   Message Conversion for WebSphere MQ Using MGW_BASIC_MSG_T

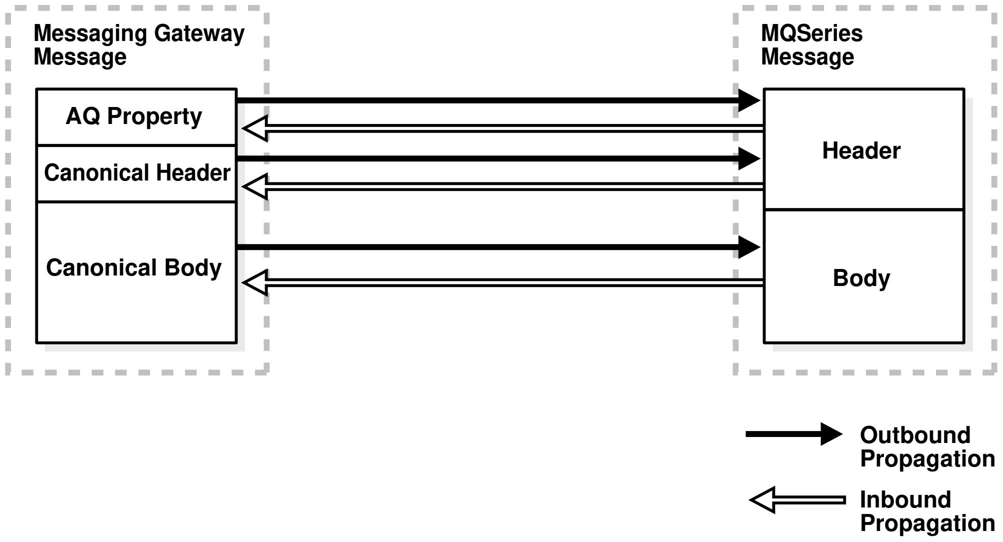


Figure C-4 illustrates the message conversion performed by the MGW WebSphere MQ driver when using the canonical type `SYS.MGW_BASIC_MSG_T`. For outbound propagation, the driver maps the Oracle Database Advanced Queuing message properties and canonical message to a WebSphere MQ message having a fixed header and a message body. For inbound propagation, the driver maps a native message to a set of Oracle Database Advanced Queuing message properties and a canonical message. When the canonical type is `RAW`, the mappings are the same, except no canonical headers exist.

WebSphere MQ Message Header Mappings

When the MGW canonical type used in an outbound propagation job is `RAW`, no WebSphere MQ header information is set from the `RAW` message body. Similarly, for inbound propagation no WebSphere MQ header information is preserved in the `RAW` message body. MGW canonical type `SYS.MGW_BASIC_MSG_T`, however, has a header that can be used to specify WebSphere MQ header fields for outbound propagation, and preserve WebSphere MQ header fields for inbound propagation.

This section describes the message properties supported for the WebSphere MQ messaging system when using `SYS.MGW_BASIC_MSG_T` as the canonical type. Table C-8 defines the MGW {name, value} pairs used to describe the WebSphere MQ header properties. The first column refers to valid string values for the `SYS.MGW_NAME_VALUE_T.NAME` field in the `SYS.MGW_BASIC_MSG_T` header. The second column refers to the `SYS.MGW_NAME_VALUE_T.TYPE` value corresponding to the name. (Refer to "Notes on Table C-8" for explanations of the numbers in parentheses.)

For inbound propagation, the WebSphere MQ driver generates {name,value} pairs based on the source message header and stores them in the header part of the canonical message of the `SYS.MGW_BASIC_MSG_T` type. For outbound propagation, the WebSphere MQ driver sets the message header and enqueue options from {name,value} pairs for these properties stored in the header part of the `SYS.MGW_BASIC_MSG_T` canonical message.

Table C-8 MGW Names for WebSphere MQ Header Values

MGW Name	MGW Type	WebSphere MQ Property Name	Used For
`MGW_MQ_accountingToken`	`RAW_VALUE (size 32)`	`accountingToken`	Outbound (1), Inbound
`MGW_MQ_applicationIdData`	`TEXT_VALUE (size 32)`	`applicationIdData`	Outbound (1), Inbound
`MGW_MQ_applicationOriginData`	`TEXT_VALUE (size 4)`	`applicationOriginData`	Outbound (1), Inbound
`MGW_MQ_backoutCount`	`INTEGER_VALUE`	`backoutCount`	Inbound
`MGW_MQ_characterSet`	`INTEGER_VALUE`	`characterSet`	Outbound, Inbound
`MGW_MQ_correlationId`	`RAW_VALUE (size 24)`	`correlationId`	Outbound (1), Inbound
`MGW_MQ_encoding`	`INTEGER_VALUE`	`encoding`	Outbound, Inbound
`MGW_MQ_expiry`	`INTEGER_VALUE`	`expiry`	Outbound, Inbound
`MGW_MQ_feedback`	`INTEGER_VALUE`	`feedback`	Outbound, Inbound

Table C-8 (Cont.) MGW Names for WebSphere MQ Header Values

MGW Name	MGW Type	WebSphere MQ Property Name	Used For
`MGW_MQ_format`	`TEXT_VALUE (size 8)`	`format`	Outbound (1), Inbound
`MGW_MQ_groupId`	`RAW_VALUE (size 24)`	`groupId`	Outbound (1), Inbound
`MGW_MQ_messageFlags`	`INTEGER_VALUE`	`messageFlags`	Outbound, Inbound
`MGW_MQ_messageId`	`RAW_VALUE (size 24)`	`messageId`	Outbound, Inbound
`MGW_MQ_messageSequenceNumber`	`INTEGER_VALUE`	`messageSequenceNumber`	Outbound, Inbound
`MGW_MQ_messageType`	`INTEGER_VALUE`	`messageType`	Outbound, Inbound
`MGW_MQ_offset`	`INTEGER_VALUE`	`offset`	Outbound, Inbound
`MGW_MQ_originalLength`	`INTEGER_VALUE`	`originalLength`	Outbound, Inbound
`MGW_MQ_persistence`	`INTEGER_VALUE`	`persistence`	Inbound
`MGW_MQ_priority`	`INTEGER_VALUE`	`priority`	Outbound, Inbound
`MGW_MQ_putApplicationName`	`TEXT_VALUE (size 28)`	`putApplicationName`	Outbound (1), Inbound
`MGW_MQ_putApplicationType`	`INTEGER_VALUE`	`putApplicationType`	Outbound (1), Inbound
`MGW_MQ_putDateTime`	`DATE_VALUE`	`putDateTime`	Inbound
`MGW_MQ_putMessageOptions`	`INTEGER_VALUE`	`putMessageOptions`	Outbound (1) (2)
`MGW_MQ_replyToQueueManagerName`	`TEXT_VALUE (size 48)`	`replyToQueueManagerName`	Outbound, Inbound
`MGW_MQ_replyToQueueName`	`TEXT_VALUE (size 48)`	`replyToQueueName`	Outbound, Inbound
`MGW_MQ_report`	`INTEGER_VALUE`	`report`	Outbound (1), Inbound
`MGW_MQ_userId`	`TEXT_VALUE (size 12)`	`userId`	Outbound, Inbound

See Also:

"DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference*

Notes on Table C-8

1. This use is subject to WebSphere MQ restrictions. For example, if `MGW_MQ_accountingToken` is set for an outgoing message, then WebSphere MQ overrides its value unless `MGW_MQ_putMessageOptions` is set to the WebSphere MQ constant `MQPMD_SET_ALL_CONTEXT`.
2. `MGW_MQ_putMessageOptions` is used as the `putMessageOptions` argument to the WebSphere MQ Base Java `Queue.put()` method. It is not part of the WebSphere MQ header information and is therefore not an actual message property.

   The value for the `openOptions` argument of the WebSphere MQ Base Java `MQQueueManager.accessQueue` method is specified when the WebSphere MQ queue is registered using the `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE` call. Dependencies can exist between the two. For instance, for `MGW_MQ_putMessageOptions` to include `MQPMD_SET_ALL_CONTEXT, the MQ_openMessageOptions` queue option must include `MQOO_SET_CONTEXT`.

   The MGW agent adds the value `MQPMO_SYNCPOINT` to any value that you can specify.

MGW sets default values for two WebSphere MQ message header fields: `messageType` defaults to `MQMT_DATAGRAM` and `putMessageOptions` defaults to `MQPMO_SYNCPOINT`.

MGW provides two default mappings between Oracle Database Advanced Queuing message properties and WebSphere MQ header fields.

One maps the Oracle Database Advanced Queuing message property `expiration`, representing the time-to-live of the message at the time the message becomes available in the queue, to the WebSphere MQ header field `expiry`, representing the time-to-live of the message. For outbound propagation, the value used for `expiry` is determined by subtracting the time the message was available in the queue from the `expiration`, converted to tenths of a second. Oracle Database Advanced Queuing value `NEVER` is mapped to `MQEI_UNLIMITED`. For inbound propagation, the value of `expiration` is simply `expiry` converted to seconds. WebSphere MQ value `MQEI_UNLIMITED` is mapped to `NEVER`.

The other default maps Oracle Database Advanced Queuing message property `priority` with the WebSphere MQ header field `priority`. It is described in Table C-9.

Table C-9 Default Priority Mappings for Propagation

Propagation Type	Message System	Priority Values									
Outbound	Oracle Database Advanced Queuing	0	1	2	3	4	5	6	7	8	9
Outbound	WebSphere MQ	9	8	7	6	5	4	3	2	1	0
Inbound	Oracle Database Advanced Queuing	9	8	7	6	5	4	3	2	1	0
Inbound	WebSphere MQ	0	1	2	3	4	5	6	7	8	9

Note:

For outbound propagation, Oracle Database Advanced Queuing priority values less than 0 are mapped to WebSphere MQ priority 9, and Oracle Database Advanced Queuing priority values greater than 9 are mapped to WebSphere MQ priority 0.

WebSphere MQ Outbound Propagation

If no message transformation is provided for outbound propagation, then the Oracle Database Advanced Queuing source queue payload type must be either `SYS.MGW_BASIC_MSG_T` or `RAW`. If a message transformation is specified, then the target ADT of the transformation must be `SYS.MGW_BASIC_MSG_T`, but the source ADT can be any ADT supported by Oracle Database Advanced Queuing.

If the Oracle Database Advanced Queuing queue payload is RAW, then the resulting WebSphere MQ message has the message body set to the value of the RAW bytes and, by default, the `format` field set to the value "`MGW_Byte`".

If the Oracle Database Advanced Queuing queue payload or transformation target ADT is `SYS.MGW_BASIC_MSG_T`, then the message is mapped to a WebSphere MQ native message as follows:

- The WebSphere MQ fixed header fields are based on the internal Oracle Database Advanced Queuing message properties and the `SYS.MGW_BASIC_MSG_T.header` attribute of the canonical message, as described in "WebSphere MQ Message Header Mappings".
- If the canonical message has a `TEXT` body, then the WebSphere MQ format header field is set to `MQFMT_STRING` unless overridden by the header property `MGW_MQ_format`. The message body is treated as text.
- If the canonical message has a `RAW` body, then the WebSphere MQ format header field is set to "`MGW_Byte`" unless overridden by the header property `MGW_MQ_format`. The message body is treated as raw bytes.
- If the canonical message has both a `TEXT` and `RAW` body, then message conversion fails.
- If the canonical message has neither a `TEXT` nor `RAW` body, then no message body is set, and the WebSphere MQ format header field is `MQFMT_NONE`.
- If the canonical message has a `TEXT` body with both small and large values set (`SYS.MGW_BASIC_MSG_T.TEXT_BODY.small_value` and `SYS.MGW_BASIC_MSG_T.TEXT_BODY.large_value` not empty), then message conversion fails.
- If the canonical message has a `RAW` body with both small and large values set (`SYS.MGW_BASIC_MSG_T.RAW_BODY.small_value` and `SYS.MGW_BASIC_MSG_T.RAW_BODY.large_value` not empty), then message conversion fails.

If the job option `PreserveMessageID` is specified with a value of `TRUE`, then the `correlationId` field of the WebSphere message header will be set to the AQ source message identifier. The `correlationId` value will be a 24-byte value of the form `"AQMSGID:"+`*`AQ_msgid`* where *`AQ_msgid`* represents the 16-byte Database AQ message identifier.

WebSphere MQ Inbound Propagation

If no message transformation is provided for inbound propagation, then the Oracle Database Advanced Queuing destination queue payload type must be either `SYS.MGW_BASIC_MSG_T` or `RAW`. If a message transformation is specified, then the source ADT of the transformation must be `SYS.MGW_BASIC_MSG_T`, but the destination ADT can be any ADT supported by Oracle Database Advanced Queuing.

If the Oracle Database Advanced Queuing queue payload is `RAW` and the incoming WebSphere MQ message has a `format` of `MQFMT_STRING`, then message conversion fails. Otherwise the message body is considered as raw bytes and enqueued directly to the destination queue. If the number of bytes is greater than 32KB, then message conversion fails. The actual limit is 32512 bytes rather than 32767 bytes.

If the Oracle Database Advanced Queuing queue payload or transformation source ADT is `SYS.MGW_BASIC_MSG_T`, then the WebSphere MQ message is mapped to a `SYS.MGW_BASIC_MSG_T` message as follows:

- Specific WebSphere MQ header fields are mapped to Oracle Database Advanced Queuing message properties as previously described.
- The `SYS.MGW_BASIC_MSG_T.header` attribute of the canonical message is set to {name, value} pairs based on the WebSphere MQ header fields, as described in Table C-8. These values preserve the original content of the WebSphere MQ message header.
- If the WebSphere MQ `format` header field is `MQFMT_STRING`, then the WebSphere MQ message body is treated as text, and its value is mapped to `SYS.MGW_BASIC_MSG_T.text_body`. For any other `format` value, the message body is treated as raw bytes, and its value is mapped to `SYS.MGW_BASIC_MSG_T.raw_body`.

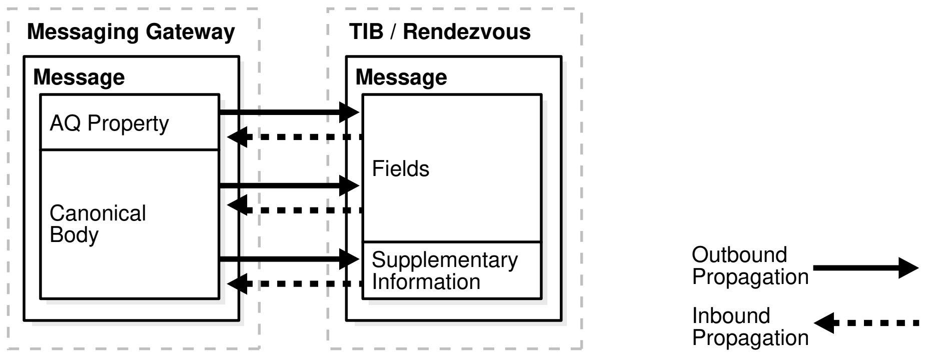
See Also:

"WebSphere MQ Message Header Mappings"

Message Conversion for TIB/Rendezvous

MGW regards a TIB/Rendezvous message as a set of fields and supplementary information. Figure C-5 shows how messages are converted between MGW and TIB/Rendezvous.

Figure C-5 Message Conversion for TIB/Rendezvous



When a message conversion failure occurs, messages are moved to an exception queue (if one has been provided), so that MGW can continue propagation of the remaining messages in the source queue. In inbound propagation from TIB/Rendezvous, an exception queue is a registered subject.

All TIB/Rendezvous wire format datatypes for TIB/Rendezvous fields are supported, except for the datatypes with unsigned integers and the nested message type. User-defined custom datatypes are not supported in this release. If a message contains data of the unsupported datatypes, then a message conversion failure occurs when the message is processed. A message conversion failure results in moving the failed message from the source queue to the exception queue, if an exception queue is provided.

Table C-10 shows the datatype mapping used when MGW converts between a native TIB/Rendezvous message and the canonical ADT. For each supported TIB/Rendezvous wire format type, it shows the Oracle type used to store the data and the `DBMS_MGWMSG` constant that represents that type.

Table C-10 TIB/Rendezvous Datatype Mapping

TIB/Rendezvous Wire Format	Oracle Type	ADT Field Type
`Bool`	`NUMBER`	`TIBRVMSG_BOOL`
`F32`	`NUMBER`	`TIBRVMSG_F32`
`F64`	`NUMBER`	`TIBRVMSG_F64`
`I8`	`NUMBER`	`TIBRVMSG_I8`
`I16`	`NUMBER`	`TIBRVMSG_I16`
`I32`	`NUMBER`	`TIBRVMSG_I32`
`I64`	`NUMBER`	`TIBRVMSG_I64`
`U8`	not supported	not supported
`U16`	not supported	not supported
`U32`	not supported	not supported
`U64`	not supported	not supported
`IPADDR32`	`VARCHAR2`	`TIBRVMSG_IPADDR32`
`IPPORT16`	`NUMBER`	`TIBRVMSG_IPPORT16`
`DATETIME`	`DATE`	`TIBRVMSG_DATETIME`
`F32ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_F32ARRAY`
`F64ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_F64ARRAY`
`I8ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_I8ARRAY`
`I16ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_I16ARRAY`
`I32ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_I32ARRAY`
`I64ARRAY`	`SYS.MGW_NUMBER_ARRAY_T`	`TIBRVMSG_I64ARRAY`
`U8ARRAY`	not supported	not supported
`U16ARRAY`	not supported	not supported
`U32ARRAY`	not supported	not supported
`U64ARRAY`	not supported	not supported
`MSG`	not supported	not supported

Table C-10 (Cont.) TIB/Rendezvous Datatype Mapping

TIB/Rendezvous Wire Format	Oracle Type	ADT Field Type
`OPAQUE`	`RAW` or `BLOB`	`TIBRVMSG_OPAQUE`
`STRING`	`VARCHAR2` or `CLOB`	`TIBRVMSG_STRING`
`XML`	`RAW` or `BLOB`	`TIBRVMSG_XML`

For propagation between Oracle Database Advanced Queuing and TIB/Rendezvous, MGW provides direct support for the Oracle Database Advanced Queuing payload types `RAW` and `SYS.MGW_TIBRV_MSG_T`. To support any other Oracle Database Advanced Queuing payload type, you must supply a transformation.

AQ Message Property Mapping for TIB/Rendezvous

This section describes the mapping between Oracle Database AQ message properties and TIB/Rendezvous fields. This mapping is used to preserve Database AQ message properties during outbound propagation, and set Database AQ message properties during inbound propagation.

Table C-11 describes the Database AQ message properties supported using TIB/Rendezvous fields. The first column indicates the `DBMS_AQ.MESSAGE_PROPERTIES_T` field for the Database AQ message property. The second and third columns indicate the name and datatype used for the TIB/Rendezvous field. The last column indicates if the message property is supported for inbound and outbound propagation.

Table C-11 TIB/Rendezvous and MGW Names for Oracle Database Advanced Queuing Message Properties

Oracle Database Advanced Queuing Message Property	MGW Name	TIB/Rendezvous Wire Format Datatype	Used For
`priority`	`MGW_AQ_priority`	`TibrvMsg.I32`	Outbound, Inbound
`expiration`	`MGW_AQ_expiration`	`TibrvMsg.I32`	Outbound, Inbound
`delay`	`MGW_AQ_delay`	`TibrvMsg.I32`	Outbound, Inbound
`correlation`	`MGW_AQ_correlation`	`TibrvMsg.STRING`	Outbound, Inbound
`exception_queue`	`MGW_AQ_exception_queue`	`TibrvMsg.STRING`	Outbound, Inbound
`enqueue_time`	`MGW_AQ_enqueue_time`	`TibrvMsg.DATETIME`	Outbound
`original_msgid`	`MGW_AQ_original_msgid`	`TibrvMsg.OPAQUE`	Outbound
`msgid` (1)	`MGW_AQ_messageID`	`TibrvMsg.OPAQUE`	Outbound

Notes on Table C-11:

1. The `msgid` Database AQ property represents the Database AQ message identifier, rather than a particular field of the `DBMS_AQ.MESSAGE_PROPERTIES_T` record.

TIB/Rendezvous Outbound Propagation

If no propagation transformation is provided for outbound propagation, then the Oracle Database Advanced Queuing source queue payload type must be either `SYS.MGW_TIBRV_MSG_T` or `RAW`. If a propagation transformation is specified, then the target ADT of the transformation must be `SYS.MGW_TIBRV_MSG_T`, but the source ADT can be any ADT supported by Oracle Database Advanced Queuing.

If the Oracle Database Advanced Queuing queue payload or transformation target ADT is `SYS.MGW_TIBRV_MSG_T`, then:

- Every field in the source message is converted to a TIB/Rendezvous message field of the resulting TIB/Rendezvous message.
- If the `reply_subject` attribute is not `NULL`, then the reply subject supplementary information is set.
- The `send_subject` field is ignored.

If the Oracle Database Advanced Queuing queue payload is `RAW`, then:

- The resulting message contains a field named `MGW_RAW_MSG` with value `TibrvMsg.OPAQUE`. The field ID is set to 0.

If the job option `AQ_MsgProperties` is specified with a value of `TRUE`, then the MGW agent generates fields to preserve the Database AQ message properties in the TIB/Rendezvous message according to Table C-11.

If the `PreserveMessageID` job option is specified with a value of `TRUE`, then the Database AQ message identifier (`msgid`) is preserved in the TIB/Rendezvous message according to Table C-11.

TIB/Rendezvous Inbound Propagation

If no propagation transformation is provided for inbound propagation, then the Oracle Database Advanced Queuing destination queue payload type must be either `RAW` or `SYS.MGW_TIBRV_MSG_T`. If a propagation transformation is specified, then the target ADT of the transformation can be any ADT supported by Oracle Database Advanced Queuing, but the source ADT of the transformation must be `SYS.MGW_TIBRV_MSG_T`.

If the Oracle Database Advanced Queuing queue payload or transformation source ADT is `SYS.MGW_TIBRV_MSG_T`, then:

- Every field in the source TIB/Rendezvous message is converted to a field of the resulting message of the `SYS.MGW_TIBRV_MSG_T` type.
- The MGW agent extracts the send subject name from the source TIB/Rendezvous message and sets the `send_subject` attribute in `SYS.MGW_TIBRV_MSG_T`. The send subject name is usually the same as the subject name of the registered propagation source queue, but it might be different when wildcards are used.
- The MGW agent extracts the reply subject name from the source TIB/Rendezvous message, if it exists, and sets the `reply_subject` attribute in `SYS.MGW_TIBRV_MSG_T`.
- If the source TIB/Rendezvous message contains more than three large text fields (greater than 4000 bytes of text) or more than three large bytes fields (greater than 2000 bytes), then message conversion fails.

If the Oracle Database Advanced Queuing queue payload is `RAW`, then:

- The Oracle Database Advanced Queuing message payload is the field data if the source TIB/Rendezvous message has a field named `MGW_RAW_MSG` of type `TibrvMsg.OPAQUE` or `TibrvMsg.XML`. The field name and ID are ignored. If no such field exists or has an unexpected type, then a message conversion failure occurs.
- A message conversion failure occurs if the `RAW` data size is greater than 32KB. This is due to a restriction on the data size allowed for a bind variable. Also, the actual limit is 32512 rather than 32767.

If the job option `AQ_MsgProperties` is specified with a value of `TRUE`, then the MGW agent searches for fields in the original TIB/Rendezvous messages with reserved field names. Table C-11 shows the field name strings and the corresponding values used in the TIB/ Rendezvous message.

If such fields exist, then the MGW agent uses the field value to set the corresponding Oracle Database Advanced Queuing message properties, instead of using the default values. If there is more than one such field with the same name, then only the first one is used. Such fields are removed from the resulting payload only if the Oracle Database Advanced Queuing queue payload is `RAW`. If a field with the reserved name does not have the expected datatype, then it causes a message conversion failure.

See Also:

"DBMS_MGWMSG" in *Oracle Database PL/SQL Packages and Types Reference* for the value datatypes

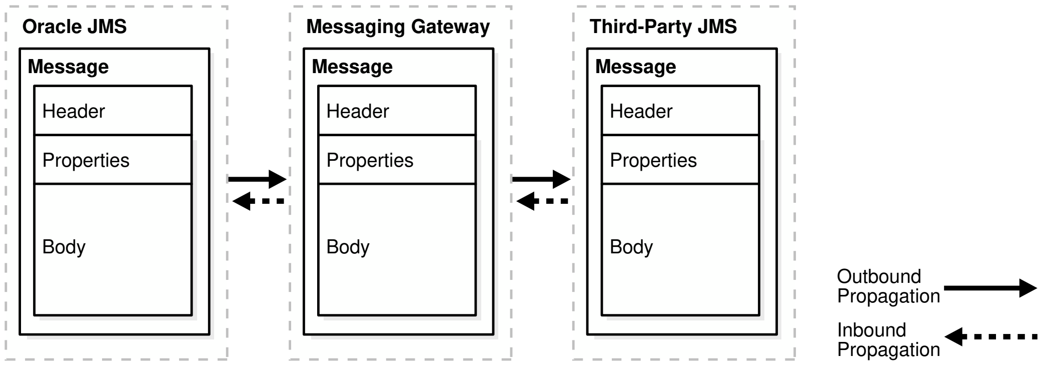
JMS Messages

MGW propagates only JMS messages between Oracle JMS and non-Oracle JMS systems, without changing the message content. Figure C-6 shows JMS message propagation.

MGW supports only the standard JMS message types. It does not support:

- JMS provider extensions, because any such extensions would not be recognized by the destination JMS system. An attempt to propagate any such non-JMS message results in an error.
- User transformations for JMS propagation.
- Propagation of Logical Change Records (LCRs).

Figure C-6 JMS Message Propagation



For the purposes of this discussion, a JMS message is a Java object of a class that implements one of the five JMS message interfaces. Table C-12 shows the JMS message interfaces and the corresponding Oracle JMS ADTs. The table also shows the interface, `javax.jms.Message`, which can be any one of the five specific types, and the corresponding generic Oracle JMS type `SYS.AQ$_JMS_MESSAGE`.

Table C-12 Oracle JMS Message Conversion

JMS Message	ADT
`javax.jms.TextMessage`	`SYS.AQ$_JMS_TEXT_MESSAGE`
`javax.jms.BytesMessage`	`SYS.AQ$_JMS_BYTES_MESSAGE`
`javax.jms.MapMessage`	`SYS.AQ$_JMS_MAP_MESSAGE`
`javax.jms.StreamMessage`	`SYS.AQ$_JMS_STREAM_MESSAGE`
`javax.jms.ObjectMessage`	`SYS.AQ$_JMS_OBJECT_MESSAGE`
`javax.jms.Message`	`SYS.AQ$_JMS_MESSAGE`

When a propagation job is activated, the MGW agent checks the Oracle Database Advanced Queuing payload type for the propagation source or destination. If the type is one of those listed in Table C-12 or `ANYDATA`, then message propagation is attempted. Otherwise an exception is logged and propagation is not attempted.

The MGW agent may add a JMS `String` property named `OracleMGW_OriginalMessageID` to the JMS message sent to the destination queue in order to preserve the original message identifier of the source message. This property is added if the `PreserveMessageID` job option is specified with a value of `TRUE`. It will also be added for any message moved to an exception queue upon a message conversion failure.

JMS Outbound Propagation

When dequeuing a message from an Oracle Database Advanced Queuing queue, Oracle JMS converts instances of the ADTs shown in Table C-12 into JMS messages. In addition it can convert instances of `ANYDATA` into JMS messages, depending on the content.

A queue with payload type `ANYDATA` can hold messages that do not map to a JMS message. MGW fails to dequeue such a message. An error is logged and propagation of messages from that queue does not continue until the message is removed.

JMS Inbound Propagation

Every message successfully dequeued using WebSphere MQ JMS is a JMS message. No message conversion is necessary prior to enqueuing using Oracle JMS. However, if the payload ADT of the propagation destination does not accept the type of the inbound message, then an exception is logged and an attempt is made to place the message in an exception queue. An example of such type mismatches is a JMS `TextMessage` and a queue payload type `SYS.AQ$_JMS_BYTES_MESSAGE`.

Monitoring Oracle Messaging Gateway

The following topics discuss means of monitoring the Oracle Messaging Gateway (MGW) agent, abnormal situations you may experience, several sources of information about Messaging Gateway errors and exceptions, and suggested remedies.

- Oracle Messaging Gateway Log Files
- Monitoring the Oracle Messaging Gateway Agent Status
- Monitoring Oracle Messaging Gateway Propagation
- Oracle Messaging Gateway Agent Error Messages

Oracle Messaging Gateway Log Files

Messaging Gateway agent status, history, and errors are recorded in Messaging Gateway log files. A different log file is created each time the Messaging Gateway agent is started. You should monitor the log file because any errors, configuration information read at startup time, or dynamic configuration information is written to the log.

The format of the log file name for the default agent is:

```
oramgw-hostname-timestamp-processid.log
```

The format of the log file name for a named agent is:

```
oramgw-AGENTNAME-hostname-timestamp-processid.log
```

By default the Messaging Gateway log file is in `ORACLE_HOME/mgw/log`. This location can overridden by the parameter `log_directory` in the Messaging Gateway initialization file used by the agent, usually `mgw.ora`.

This section contains these topics:

- Sample Oracle Messaging Gateway Log File
- Interpreting Exception Messages in an Oracle Messaging Gateway Log File

Sample Oracle Messaging Gateway Log File

The following sample log file shows the Messaging Gateway agent starting. The sample log file shows that a messaging link, a registered foreign queue, a propagation job, and a schedule associated with the job have been added. The log file shows that the propagation job has been activated. The last line indicates that the Messaging Gateway is up and running and ready to propagate messages.

Example C-36 Sample Messaging Gateway Log File

```
>>2007-01-16 15:04:49 MGW C-Bootstrap 0 LOG process-id=11080
Bootstrap program starting
>>2007-01-16 15:04:50 MGW C-Bootstrap 0 LOG process-id=11080
JVM created -- heapsize = 64
>>2007-01-16 15:04:53 MGW Engine 0 200 main
MGW Agent version: 11.1.0.0
>>2007-01-16 15:04:53 MGW AdminMgr 0 LOG main
Connecting to database using connect string = jdbc:oracle:oci:@INST1
>>2007-01-16 15:05:00 MGW Engine 0 200 main
MGW Component version: 11.1.0.3.0
>>2007-01-16 15:05:01 MGW Engine 0 200 main
MGW agent name: DEFAULT_AGENT, MGW job instance id: 273006EC6ED255F1E040578C6D021A8C,
MGW database instance: 1
>>2007-01-16 15:05:09 MGW Engine 0 1 main
Agent is initializing.
>>2007-01-16 15:05:09 MGW Engine 0 23 main
The number of worker threads is set to 1.
>>2007-01-16 15:05:09 MGW Engine 0 22 main
The default polling interval is set to 5000ms.
```

```
>>2007-01-16 15:05:09 MGW MQD 0 LOG main
Creating MQSeries messaging link:
link : MQLINK
link type : Base Java interface
queue manager : my.queue.manager
channel : channel1
host : my.machine
port : 1414
user :
ccdt url :
ssl cipherSuite :
connections : 1
inbound logQ : logq1
outbound logQ : logq2
>>2007-01-16 15:05:09 MGW Engine 0 4 main
Link MQLINK has been added.
>>2007-01-16 15:05:09 MGW Engine 0 7 main
Queue DESTQ@MQLINK has been registered; provider queue: MGWUSER.MYQUEUE.
>>2007-01-16 15:05:09 MGW Engine 0 9 main
Propagation Schedule JOB_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been
added.
>>2007-01-16 15:05:09 MGW AQN 0 LOG main
Creating AQ messaging link:
link : oracleMgwAq
link type : native
database : INST1
user : MGWAGENT
connection type : JDBC OCI
connections : 1
inbound logQ : SYS.MGW_RECV_LOG
outbound logQ : SYS.MGW_SEND_LOG
>>2007-01-16 15:05:10 MGW Engine 0 19 main
MGW propagation job JOB_AQ2MQ has been activated.
>>2007-01-16 15:05:10 MGW Engine 0 14 main
MGW propagation job JOB_AQ2MQ (MGWUSER.MGW_BASIC_SRC --> DESTQ@MQLINK) has been added.
>>2007-01-16 15:05:11 MGW Engine 0 2 main
Agent is up and running.
```

Interpreting Exception Messages in an Oracle Messaging Gateway Log File

Exception messages logged to the Messaging Gateway log file may include one or more linked exceptions, identified by `[Linked-exception]` in the log file. These are often the most useful means of determining the cause of a problem. For instance, a linked exception could be a `java.sql.SQLException`, possibly including an Oracle error message, a PL/SQL stack trace, or both.

The following example shows entries from a Messaging Gateway log file when an invalid value (`bad_service_name`) was specified for the `database` parameter of `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT`. This resulted in the Messaging Gateway agent being unable to establish database connections.

Example C-37 Sample Exception Message

```
>>2003-07-22 15:27:26  MGW  AdminMgr  0  LOG  main
Connecting to database using connect string = jdbc:oracle:oci8:@BAD_SERVICE_NAME
>>2003-07-22 15:27:29  MGW  Engine  0  EXCEPTION  main
oracle.mgw.admin.MgwAdminException: [241]  Failed to connect to database. SQL
error: 12154, connect string: jdbc:oracle:oci8:@BAD_SERVICE_NAME
[ …Java stack trace here…]
[Linked-exception]
java.sql.SQLException: ORA-12154: TNS:could not resolve the connect identifier
```

```
specified
[ …Java stack trace here…]
>>2003-07-22 15:27:29  MGW  Engine  0  25  main
Agent is shutting down.
```

Monitoring the Oracle Messaging Gateway Agent Status

This section contains these topics:

- MGW_GATEWAY View
- Oracle Messaging Gateway Irrecoverable Error Messages
- Other Oracle Messaging Gateway Error Conditions

MGW_GATEWAY View

The `MGW_GATEWAY` view monitors the progress of the Messaging Gateway agent. Among the fields that can be used to monitor the agent are:

- `AGENT_NAME`
- `AGENT_INSTANCE`
- `AGENT_PING`
- `AGENT_STATUS`
- `LAST_ERROR_MSG`
- `SERVICE`

The `AGENT_STATUS` field shows the status of the agent. This column has the following possible values:

NOT_STARTED
Indicates that the agent is neither running nor scheduled to be run.

START_SCHEDULED
Indicates that the agent job is waiting to be run by the job scheduler.

STARTING
Indicates that the agent is in the process of starting.

INITIALIZING
Indicates that the agent has started and is reading configuration data.

RUNNING
Indicates that the agent is ready to propagate any available messages or process dynamic configuration changes.

SHUTTING_DOWN
Indicates that the agent is in the process of shutting down.

BROKEN
Indicates that, while attempting to start an agent process, Messaging Gateway has detected another agent already running. This situation should never occur under normal usage.

Querying the `AGENT_PING` field pings the Messaging Gateway agent. Its value is either `REACHABLE` or `UNREACHABLE`. An agent with status of `RUNNING` should almost always be `REACHABLE`.

The columns `LAST_ERROR_MSG`, `LAST_ERROR_DATE`, and `LAST_ERROR_TIME` give valuable information if an error in starting or running the Messaging Gateway agent occurs. `AGENT_INSTANCE` indicates the Oracle Database instance on which the Messaging Gateway instance was started.

See Also:

"DBMS_MGWADM" in *Oracle Database PL/SQL Packages and Types Reference* for more information on the `MGW_GATEWAY` view

Oracle Messaging Gateway Irrecoverable Error Messages

A status of `NOT_STARTED` in the `AGENT_STATUS` field of the `MGW_GATEWAY` view indicates that the Messaging Gateway agent is not running. If the `AGENT_STATUS` is `NOT_STARTED` and the `LAST_ERROR_MSG` field is not NULL, then the Messaging Gateway agent has encountered an irrecoverable error while starting or running. Check if a Messaging Gateway log file has been generated and whether it indicates any errors. If a log file is not present, then the Messaging Gateway agent process was probably not started.

This section describes the causes and solutions for some error messages that may appear in the `LAST_ERROR_MSG` field of the `MGW_GATEWAY` view. Unless indicated otherwise, the Messaging Gateway agent will not attempt to restart itself when one of these errors occurs.

ORA-01089: Immediate shutdown in progress - no operations are permitted

The Messaging Gateway agent has shut down because the `SHUTDOWN IMMEDIATE` command was used to shut down a running Oracle Database instance on which the agent was running. The agent will restart itself on the next available database instance on which it is set up to run.

ORA-06520: PL/SQL: Error loading external library

The Messaging Gateway agent process was unable to start because the shared library was not loaded. This may be because the Java shared library was not in the library path. Verify that the library path in `listener.ora` has been set correctly.

ORA-28575: Unable to open RPC connection to external procedure agent

The Messaging Gateway agent was unable to start. It will attempt to start again automatically.

Possible causes include:

- The listener is not running. If you have modified `listener.ora`, then you must stop and restart the listener before the changes will take effect.
- Values in `tnsnames.ora`, `listener.ora`, or both are not correct.

  In particular, `tnsnames.ora` must have a net service name entry of `MGW_AGENT`. This entry is not needed for Messaging Gateway on Windows. The SID value specified for `CONNECT_DATA` of the `MGW_AGENT` net service name in `tnsnames.ora` must match the `SID_NAME` value of the `SID_DESC` entry in `listener.ora`. If the `MGW_AGENT` net service name is set up for an Inter-process Communication (IPC) connection, then the `KEY` values for `ADDRESS` in `tnsnames.ora` and `listener.ora` must match. If the `names.default_domain` parameter for `sqlnet.ora` has been used to set a default domain, then that domain must be appended to the `MGW_AGENT` net service name in `tnsnames.ora`.

ORA-28576: Lost RPC connection to external procedure agent

The Messaging Gateway agent process ended prematurely. This may be because the process was stopped by an outside entity or because an internal error caused a malfunction. The agent will attempt to start again automatically. Check the Messaging Gateway log file to determine if further information is available. If the problem persists, then contact Oracle Support Services for assistance.

ORA-32830: Result code -2 returned by Messaging Gateway agent

An error occurred when the Messaging Gateway agent tried to read its initialization file, usually `mgw.ora`. Verify that the file is readable.

ORA-32830: Result code -3 returned by Messaging Gateway agent

An error occurred creating the Messaging Gateway log file. Verify that the log directory can be written to. The default location is *`ORACLE_HOME`*`/mgw/log`.

ORA-32830: Result code -8 returned by Messaging Gateway agent

An error occurred starting the Java Virtual Machine (JVM). Verify that:

- You are using the correct Java version
- Your operating system version and patch level are sufficient for the JDK version
- You are using a reasonable value for the JVM heap size

  The heap size is specified by the `max_memory` parameter of `DBMS_MGWADM.ALTER_AGENT`
- On Windows platforms, verify the `MGW_PRE_PATH` set in `mgw.ora` contains the path to the correct JVM library (`jvm.dll`).

ORA-32830: Result code -12 returned by Messaging Gateway agent

An error occurred writing to the Messaging Gateway log file. Check the free disk space or any other issues that might result in file I/O problems.

ORA-32830: Result code -17 returned by Messaging Gateway agent

The JVM was successfully created but an error occurred trying to call the MGW Java agent program. Verify that the CLASSPATH set in `mgw.ora` is correct.

ORA-32830: Result code -19 returned by Messaging Gateway agent

The Messaging Gateway agent was configured to use a particular initialization file but that file does not exist. The `INITFILE` field of the `MGW_GATEWAY` view shows the full pathname of the file specified by the administrator. Either create that initialization file, or use `DBMS_MGWADM.ALTER_AGENT` to set `INITFILE` to another file or `NULL` to use the default initialization file.

ORA-32830: Result code -100 returned by Messaging Gateway agent

The Messaging Gateway agent JVM encountered a runtime exception or error on startup before it could write to the log file.

ORA-32830: Result code -101 returned by Messaging Gateway agent

An irrecoverable error caused the Messaging Gateway agent to shut down. Check the Messaging Gateway log file for further information. Verify that the values specified in `mgw.ora` are correct. Incorrect values can cause the Messaging Gateway agent to terminate due to unusual error conditions.

ORA-32830: Result code -102 returned by Messaging Gateway agent

The Messaging Gateway agent shut down because the version of file `ORACLE_HOME/mgw/jlib/mgw.jar` does not match the version of the Messaging Gateway PL/SQL packages. Verify that all Messaging Gateway components are from the same release.

ORA-32830: Result code -103 returned by Messaging Gateway agent

The Messaging Gateway agent shut down because the database instance on which it was running was shutting down. The agent should restart automatically, either on another instance if set up to do so, or when the instance that shut down is restarted.

ORA-32830: Result code -104 returned by Messaging Gateway agent

See previous error.

ORA-32830: Result code -105 returned by Messaging Gateway agent

The Messaging Gateway agent detected that it was running when it should not be. This should not happen. If it does, `AGENT_STATUS` will be `BROKEN` and the agent will shut down automatically. If you encounter this error:

- Terminate any Messaging Gateway agent process that may still be running. The process is usually named `extprocmgwextproc`.
- Run `DBMS_MGWADM.CLEANUP_GATEWAY(DBMS_MGWADM.CLEAN_STARTUP_STATE)`.
- Start the Messaging Gateway agent using `DBMS_MGWADM.STARTUP`.

ORA-32830: Result code -106 returned by Messaging Gateway agent

See previous error.

See Also:

"DBMS_MGWADM" in *Oracle Database PL/SQL Packages and Types Reference*

Other Oracle Messaging Gateway Error Conditions

This section discusses possible causes for `AGENT_STATUS` remaining `START_SCHEDULED` in `MGW_GATEWAY` view for an extended period.

Database Service Not Started

Messaging Gateway uses an Oracle Scheduler job to start the Messaging Gateway agent. Oracle Scheduler allows you to specify a database service under which a job should be run (service affinity). Messaging Gateway allows an administrator to configure the Messaging Gateway agent with a database service that will be used to configure the Scheduler job class associated with that agent.

When you shutdown a database Oracle stops all services to that database. You may need to manually restart the services when you start the database. If a Scheduler job is associated with a service then the job will not run until the service is started. If `AGENT_STATUS` for a Messaging

Gateway agent remains `START_SCHEDULED` for an extended period that might indicate that the database service is disabled or no database instances associated with the service are running. Use the `MGW_GATEWAY` view, Oracle Scheduler views, and service views to determine how the agent was configured and the current state of the Scheduler job and database service.

> **Note:**
>
> Oracle Messaging Gateway Agent Scheduler Job for information about Oracle Scheduler objects used by Messaging Gateway.

Too Few Job Queue Processes

Messaging Gateway uses Oracle Scheduler to start the Messaging Gateway external process. When `AGENT_STATUS` is `START_SCHEDULED`, the Messaging Gateway agent Scheduler job is waiting to be run by the Scheduler. The Messaging Gateway job will not run until there is an available job process. Messaging Gateway holds its Scheduler job process for the lifetime of the Messaging Gateway agent session. If multiple Messaging Gateway agents have been started, each agent uses its own Scheduler job and require its own job process.

If the value of the database initialization parameter `JOB_QUEUE_PROCESSES` is zero, then no Oracle Scheduler jobs will run. If the value is non-zero, it effectively becomes the maximum number of Scheduler jobs and job queue jobs than can concurrently run.

If Messaging Gateway status remains `START_SCHEDULED` for an extended period of time, then it may indicate that the database has been started with a value for `JOB_QUEUE_PROCESSES` that is zero or is too low and that all secondary job processes are busy. Verify that the value is non-zero and that the database instance has been started with enough job queue processes so that one is available for each Messaging Gateway agent.

Scheduler Job Broken or Disabled

The Messaging Gateway agent status will remain `START_SCHEDULED` if the Oracle Scheduler job associated with a Messaging Gateway agent has become disabled or broken for some reason. To determine if this is the case, use the `DBA_SCHEDULER_JOBS` view to look at `STATE` field for the agent's Scheduler job. Normally the Scheduler job state will be `SCHEDULED` when the Messaging Gateway agent's Scheduler job is waiting to be run, or `RUNNING` when the Messaging Gateway agent is running. The agent's Scheduler job should not exist if the Messaging Gateway agent status is `NOT_STARTED`.

Check other Scheduler views, such as `DBA_SCHEDULER_JOB_RUN_DETAILS`, for additional information about the Messaging Gateway Scheduler jobs. Also check the `MGW_GATEWAY` view and the Messaging Gateway log file for any error messages that may indicate a problem.

> **Note:**
>
> Oracle Messaging Gateway Agent Scheduler Job for information about Oracle Scheduler objects used by Messaging Gateway

Oracle Real Application Clusters (Oracle RAC) Environment

If Messaging Gateway is being used in an Oracle RAC environment and the agent has been configured with a database service but no database instances are running that have the service enabled, then the Messaging Gateway `AGENT_STATUS` will remain `START_SCHEDULED` until the service is started on a running database instance.

Monitoring Oracle Messaging Gateway Propagation

Messaging Gateway propagation can be monitored using the `MGW_JOBS` view and the Messaging Gateway log file. The view provides information on propagated messages and errors that may have occurred during propagation attempts. The log file can be used to determine the cause of the errors.

Besides showing configuration information, the `MGW_JOBS` view also has dynamic information that can be used to monitor message propagation. Applicable fields include `STATUS`, `ENABLED`, `PROPAGATED_MSGS`, `EXCEPTIONQ_MSGS`, `FAILURES`, `LAST_ERROR_MSG`, `LAST_ERROR_DATE`, and `LAST_ERROR_TIME`.

The `STATUS` field indicates current status of the job. `READY` means that the job is ready for propagation (but only if the `ENABLED` field is `TRUE`). `RETRY` means that a propagation failure occurred but that propagation will be retried. `FAILED` means that the agent has stopped propagation for the job due to an unrecoverable error or the maximum number of consecutive propagation failures has been reached. `DELETE_PENDING` means job removal is pending due to `DBMS_MGWADM.REMOVE_JOB` being called but certain cleanup tasks pertaining to the job are still outstanding. `SUBSCRIBER_DELETE_PENDING` means that `DBMS_MGWADM.REMOVE_SUBSCRIBER` has been called on an old style propagation job but certain cleanup tasks pertaining to the job are still outstanding.

The `ENABLED` field indicates whether the propagation job is currently enabled. `TRUE` indicates the job is enabled while `FALSE` indicates the job is disabled. No propagation will occur unless the job is enabled.

The `PROPAGATED_MSGS` field of the `MGW_JOBS` view indicates how many messages have been successfully propagated. This field is reset to zero when the Messaging Gateway agent is started.

If a Messaging Gateway propagation job has been configured with an exception queue, then the Messaging Gateway agent will move messages to that exception queue the first time the Messaging Gateway agent encounters a propagation failure caused by a message conversion failure. A message conversion failure is indicated by `oracle.mgw.common.MessageException` in the Messaging Gateway log file. The `EXCEPTIONQ_MSGS` field indicates how many messages have been moved to the exception queue. This field is reset to zero when the Messaging Gateway agent is started.

If an error occurs during message propagation for a propagation job, a count is incremented in the `FAILURES` field. This field indicates the number of failures encountered since the last successful propagation of messages. Each time a failure occurs, an error message and the time it occurred will be shown by `LAST_ERROR_MSG`, `LAST_ERROR_DATE`, and `LAST_ERROR_TIME`. When the number of failures reaches sixteen, Messaging Gateway halts propagation attempts for this propagation job. To resume propagation attempts you must call `DBMS_MGWADM.RESET_JOB` for the propagation job.

If an error occurs, then examine the Messaging Gateway log file for further information.

See Also:

"DBMS_MGWADM" in *Oracle Database PL/SQL Packages and Types Reference*

Oracle Messaging Gateway Agent Error Messages

This section lists some of the most commonly occurring errors that are shown in the `LAST_ERROR_MSG` column of the `MGW_JOBS` view and logged to the Messaging Gateway agent log file. Also shown are some errors that require special action. When you notice that a failure has occurred, look at the linked exceptions in the log file to determine the root cause of the problem.

Two primary types of errors are logged to the Messaging Gateway agent log file:

- `oracle.mgw.common.MessageException`

  This error type is logged when a message conversion failure occurs. The Messaging Gateway agent probably cannot propagate the message causing the failure, and the propagation job will eventually be stopped.
- `oracle.mgw.common.GatewayException`

  This error type is logged when some failure other than message conversion occurs. Depending on the cause, the problem may fix itself or require user action.

[221] Failed to access <*messaging_system*> queue: <*queue*>

An error occurred while trying to access either an Oracle Database Advanced Queuing queue or a non-Oracle queue. Check the linked exception error code and message in the log file.

[241] Failed to connect to database. SQL error: <*error*>, connect string: <*connect_string*>

This is probably caused by incorrect MGW agent connection information specified for `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT`. Either the Messaging Gateway agent user or password is incorrect or the database specifier (database parameter) is incorrect. Verify that the connection information is correct for the connection type used by the agent, JDBC OCI or JDBC Thin.

If the `database` parameter is `NULL`, then check the Messaging Gateway log file for the following Oracle linked errors:

```
ORA-01034: ORACLE not available
ORA-27101: shared memory realm does not exist
```

These two errors together indicate that the Messaging Gateway agent is attempting to connect to the database using a local IPC connection, but the `ORACLE_SID` value is not correct.

A local connection is used when the database parameter is set to `NULL`. If a local connection is desired, the correct `ORACLE_SID` value must be set in the Messaging Gateway agent process. This can be done by adding the following line to the MGW initialization file, usually `mgw.ora`:

```
set ORACLE_SID = sid_value
```

`ORACLE_SID` need not be set in the MGW initialization file if the database parameter is not `NULL`.

If setting `ORACLE_SID` in the MGW initialization file does not work, then the database parameter must be set to a value that is not `NULL`.

If the JDBC Thin connection is used, then the database parameter must be not `NULL`. If the JDBC Thin connection is used and the database parameter is a TNSNames alias, make sure that the `oracle.net.tns_names` Java property is set in the MGW initialization file. The property can be set by adding the following line to the MGW initialization file:

```
setJavaProp oracle.net.tns_admin=<directory containing tnsnames.ora>
```

Note:

"oracle.net.tns_admin" for more information

[415] Missing messages from source queue of job <*job_name*>

Possible causes include:

- The agent partially processed persistent messages that were dequeued by someone other than the Messaging Gateway agent.
- The propagation source queue was purged or re-created.
- A message was moved to the Oracle Database Advanced Queuing exception queue.

If this error occurs, then call procedure `CLEANUP_GATEWAY` in the `DBMS_MGWADM` package:

```
DBMS_MGWADM.CLEANUP_GATEWAY (
       action => DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE,
       sarg => <job_name>);
```

The call takes effect only if the propagation job has encountered the missing message problem and the agent is running. The agent treats the missing messages as nonpersistent messages and continues processing the propagation job.

See Also:

"Propagation Job Overview" for more information on Messaging Gateway exception queues

[416] Missing log records in receiving log queue for job <*job_name*>

Possible causes include:

- Log records were dequeued from the log queues by someone other than the Messaging Gateway agent.
- The log queues were purged or re-created.

If this error occurs, then call procedure `CLEANUP_GATEWAY` in the `DBMS_MGWADM` package:

```
DBMS_MGWADM.CLEANUP_GATEWAY (
       action => DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC,
       sarg => <job_name>);
```

The call takes effect only if the propagation job has encountered the missing log records problem and the agent is running.

Note:

Calling procedure `DBMS_MGWADM.CLEANUP_GATEWAY` may result in duplicated messages if the missing messages have already been propagated to the destination queue. Users should check the source and destination queues for any messages that exist in both places. If such messages exist, then they should be removed from either the source or destination queue before calling this procedure.

[417] Missing log records in sending log queue for job <*job_name*>

See previous error.

[421] WARNING: Unable to get connections to recover job <*job_name*>

This message is a warning message indicating that the Messaging Gateway agent failed to get a connection to recover the propagation job, because other propagation jobs are using them all. The agent will keep trying to get a connection until it succeeds.

If this message is repeated many times for a WebSphere MQ link, then increase the maximum number of connections used by the Messaging Gateway link associated with the propagation job.

See Also:

"Altering a Messaging System Link"

[434] Failed to access queue <*queue*>; provider queue <*queue*>

This message indicates that a messaging system native queue cannot be accessed. The queue may have been registered by `DBMS_MGWADM.REGISTER_FOREIGN_QUEUE`, or it may be an Oracle Database Advanced Queuing queue. The linked exceptions should give more information.

Possible causes include:

- The foreign queue was registered incorrectly, or the Messaging Gateway link was configured incorrectly.

  Verify configuration information. If possible, use the same configuration information to run a sample application of the non-Oracle messaging system.

- The non-Oracle messaging system is not accessible.

  Check that the non-Oracle messaging system is running and can be accessed using the information supplied in the Messaging Gateway link.

- The Oracle Database Advanced Queuing queue does not exist. Perhaps the queue was removed after the Messaging Gateway propagation job was created.

  Check that the Oracle Database Advanced Queuing queue still exists.

[436] LOW MEMORY WARNING: total memory = < >, free_mem = < >

The Messaging Gateway agent JVM is running low on memory. Java garbage collection will be invoked, but this may represent a JVM heap size that is too small. Use the `max_memory` parameter of `DBMS_MGWADM.ALTER_AGENT` to increase the JVM heap size. If the Messaging Gateway agent is running, then it must be restarted for this change to take effect.

[703] Failed to retrieve information for transformation <*transformation_id*>

The Messaging Gateway agent could not obtain all the information it needs about the transformation. The transformation parameter of `DBMS_MGWADM.CREATE_JOB` must specify the name of the registered transformation and not the name of the transformation function.

Possible causes include:

- The transformation does not exist. Verify that the transformation has been created. You can see this from the following query performed as user SYS:

  ```
  SELECT TRANSFORMATION_ID, OWNER FROM DBA_TRANSFORMATIONS;
  ```

- The wrong transformation is registered with Messaging Gateway. Verify that the transformation registered is the one intended.
- The Messaging Gateway agent user does not have `EXECUTE` privilege on the object type used for the `from_type` or the `to_type` of the transformation indicated in the exception.

  It is not sufficient to grant `EXECUTE` to `MGW_AGENT_ROLE` and then grant `MGW_AGENT_ROLE` to the agent user. You must grant `EXECUTE` privilege on the object type directly to the agent user or to `PUBLIC`.

  The following example shows such a case for the `from_type`. It also shows the use of linked exceptions for determining the precise cause of the error.

  No EXECUTE Privilege on Object Type

  ```
  Errors occurred during processing of job JOB_AQ2MQ_2
  oracle.mgw.common.GatewayException: [703] Failed to retrieve information for
  transformation mgwuser.SAMPLEADT_TO_MGW_BASIC_MSG
  […Java stack trace here…]
  [Linked-exception]
  java.sql.SQLException: "from_type" is null
  […Java stack trace here…]
  ```

[720] AQ payload type <*type*> not supported; queue: <*queue*>

The payload type of the Oracle Database Advanced Queuing queue used by a Messaging Gateway propagation job is not directly supported by Messaging Gateway. For non-JMS propagation, Messaging Gateway directly supports the payload types `RAW`, `SYS.MGW_BASIC_MSG_T` and `SYS.MGW_TIBRV_MSG_T`.

Possible actions include:

- Configure the Messaging Gateway propagation job to use a transformation that converts the queue payload type to a supported type.
- Remove the Messaging Gateway propagation job and create a new job that uses an Oracle Database Advanced Queuing queue with a supported payload type.

  For Java Message Service (JMS) propagation, the Messaging Gateway propagation job must be removed and a new job created whose Oracle Database Advanced Queuing payload type is supported by Oracle Java Message Service (Oracle JMS). Transformations are not supported for JMS propagation.

[721] Transformation type <*type*> not supported; queue: <*queue_name*>, transform: <*transformation*>

A Messaging Gateway propagation job was configured with a transformation that uses an object type that is not one of the Messaging Gateway canonical types.

For an outbound job, the transformation `from_type` must be the Oracle Database Advanced Queuing payload type, and the `to_type` must be a Messaging Gateway canonical type. For an inbound job, the transformation `from_type` must be a Messaging Gateway canonical type and the `to_type` must be the Oracle Database Advanced Queuing payload type.

[722] Message transformation failed; queue: <*queue_name*>, transform: <*transformation*>

An error occurred while attempting execution of the transformation. ORA-25229 is typically thrown by Oracle Database Advanced Queuing when the transformation function raises a PL/SQL exception or some other Oracle error occurs when attempting to use the transformation.

Possible causes include:

- The Messaging Gateway agent user does not have `EXECUTE` privilege on the transformation function. This is illustrated in the following example.

  It is not sufficient to grant `EXECUTE` to `MGW_AGENT_ROLE` and then grant `MGW_AGENT_ROLE` to the Messaging Gateway agent user. You must grant `EXECUTE` privilege on the transformation function directly to the Messaging Gateway agent user or to `PUBLIC`.

  No EXECUTE Privilege on Transformation Function

  ```
  Errors occurred during processing of job JOB_MQ2AQ_2
  oracle.mgw.common.GatewayException: [722] Message transformation failed queue:
  MGWUSER.DESTQ_SIMPLEADT, transform: MGWUSER.MGW_BASIC_MSG_TO_SIMPLEADT
  […Java stack trace here…]
  [Linked-exception]
  oracle.mgw.common.MessageException: [722] Message transformation failed;
  queue: MGWUSER.DESTQ_SIMPLEADT, transform:
  MGWUSER.MGW_BASIC_MSG_TO_SIMPLEADT
  […Java stack trace here…]
  [Linked-exception]
  java.sql.SQLException: ORA-25229: error on transformation of message msgid:
  9749DB80C85B0BD4E03408002086745E
  ORA-00604: error occurred at recursive SQL level 1
  ORA-00904: invalid column name
  […Java stack trace here…]
  ```

- The transformation function does not exist, even though the registered transformation does. If the transformation function does not exist, it must be re-created.
- The Messaging Gateway agent user does not have `EXECUTE` privilege on the payload object type for the queue indicated in the exception.

  It is not sufficient to grant `EXECUTE` to `MGW_AGENT_ROLE` and then grant `MGW_AGENT_ROLE` to the Messaging Gateway agent user. You must grant `EXECUTE` privilege on the object type directly to the Messaging Gateway agent user or to `PUBLIC`.
- The transformation function raised the error. Verify that the transformation function can handle all messages it receives.

[724] Message conversion not supported; to AQ payload type: <*type*>, from type: <*type*>

A Messaging Gateway propagation job is configured for inbound propagation where the canonical message type generated by the non-Oracle messaging system link is not compatible with the Oracle Database Advanced Queuing queue payload type. For example, propagation from a TIB/Rendezvous messaging system to an Oracle Database Advanced Queuing queue with a `SYS.MGW_BASIC_MSG_T` payload type, or propagation from WebSphere MQ to an Oracle Database Advanced Queuing queue with a `SYS.MGW_TIBRV_MSG_T` payload type.

Possible actions include:

- Configure the Messaging Gateway propagation job with a transformation that maps the canonical message type generated by the non-Oracle messaging link to the Oracle Database Advanced Queuing payload type.
- Remove the Messaging Gateway propagation job and create a new job whose Oracle Database Advanced Queuing queue payload type matches the canonical message type generated by the non-Oracle link.

[725] Text message not supported for RAW payload

A Messaging Gateway propagation job is configured for inbound propagation to an Oracle Database Advanced Queuing destination having a `RAW` payload type. A text message was received from the source (non-Oracle) queue resulting in a message conversion failure.

If support for text data is required, remove the Messaging Gateway propagation job and create a new job to an Oracle Database Advanced Queuing destination whose payload type supports text data.

[726] Message size <*size*> too large for RAW payload; maximum size is <*size*>

A Messaging Gateway propagation job is configured for inbound propagation to an Oracle Database Advanced Queuing destination having a `RAW` payload type. A message conversion failure occurred when a message containing a large `RAW` value was received from the source (non-Oracle) queue.

If large data support is required, remove the Messaging Gateway propagation job and create a new job to an Oracle Database Advanced Queuing destination whose payload type supports large data, usually in the form of an object type with a BLOB attribute.

[728] Message contains too many large (BLOB) fields

The source message contains too many fields that must be stored in `BLOB` types. `SYS.MGW_TIBRV_MSG_T` is limited to three `BLOB` fields. Reduce the number of large fields in the message, perhaps by breaking them into smaller fields or combining them into fewer large fields.

[729] Message contains too many large (CLOB) fields

The source message contains too many fields that contain a large text value that must be stored in a `CLOB`. `SYS.MGW_TIBRV_MSG_T` is limited to three `CLOB` fields. Reduce the number of large fields in the message, perhaps by breaking them into smaller fields or combining them into fewer large fields.

[805] MQSeries Message error while enqueuing to queue: <*queue*>

WebSphere MQ returned an error when an attempt was made to put a message in a WebSphere MQ queue. Check the linked exception error code and message in the log file. Consult WebSphere MQ documentation.

Oracle Messaging Gateway Views

MGW_GATEWAY: Configuration and Status Information

This view lists configuration and status information for Messaging Gateway.

Table C-13 MGW_GATEWAY View Properties

Name	Type	Description
AGENT_DATABASE	VARCHAR2	The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection is used.
AGENT_INSTANCE	NUMBER	The database instance on which the Messaging Gateway agent is currently running. This should be NULL if the agent is not running.
AGENT_JOB	NUMBER	[Deprecated] Job number of the queued job used to start the Messaging Gateway agent process. The job number is set when Messaging Gateway is started and cleared when it shuts down.
AGENT_NAME	VARCHAR2	Name of the Messaging Gateway agent
AGENT_PING	VARCHAR2	Gateway agent ping status. Values: • NULL means no ping attempt was made. • REACHABLE means ping attempt was successful. • UNREACHABLE means ping attempt failed. AGENT_PING attempts to contact the Messaging Gateway agent. There is a short delay (up to 5 seconds) if the ping attempt fails. No ping is attempted if the AGENT_STATUS is NOT_STARTED or START_SCHEDULED.
AGENT_START_TIME	TIMESTAMP	The time when the Messaging Gateway agent job currently running was started. This should be NULL if the agent is not running.

Table C-13 (Cont.) MGW_GATEWAY View Properties

Name	Type	Description
AGENT_STATUS	VARCHAR2	Status of the Messaging Gateway agent. Values: • NOT_STARTED means the Messaging Gateway agent has not been started • START_SCHEDULED means Messaging Gateway agent has been scheduled to start. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, but the queued job used to start the Messaging Gateway agent has not yet run. • STARTING means Messaging Gateway agent is starting. That is, Messaging Gateway has been started using DBMS_MGWADM.STARTUP, the queued job has run, and the Messaging Gateway agent is starting up. • INITIALIZING means the Messaging Gateway agent has started and is initializing • RUNNING means the Messaging Gateway agent is running • SHUTTING_DOWN means the Messaging Gateway agent is shutting down • BROKEN means an unexpected condition has been encountered that prevents the Messaging Gateway agent from starting. DBMS_MGWADM.CLEANUP_GATEWAY must be called before the agent can be started.
AGENT_USER	VARCHAR2	Database user name used by the Messaging Gateway agent to connect to the database
COMMENTS	VARCHAR2	Comments for the agent
CONNTYPE	VARCHAR2	Connection type used by the agent: • JDBC_OCI if the JDBC OCI driver is used • JDBC_THIN if the JDBC Thin driver is used
INITFILE	VARCHAR2	Name of the Messaging Gateway initialization file used by the agent. NULL indicates that the default initialization file is used.
LAST_ERROR_DATE	DATE	Date of last Messaging Gateway agent error. The last error information is cleared when Messaging Gateway is started. It is set if the Messaging Gateway agent fails to start or terminates due to an abnormal condition.
LAST_ERROR_MSG	VARCHAR2	Message for last Messaging Gateway agent error
LAST_ERROR_TIME	VARCHAR2	Time of last Messaging Gateway agent error
MAX_CONNECTIONS	NUMBER	[Deprecated] Maximum number of messaging connections to Oracle Database
MAX_MEMORY	NUMBER	Maximum heap size used by the Messaging Gateway agent (in MB)
MAX_THREADS	NUMBER	Maximum number of messaging threads created by the Messaging Gateway agent
SERVICE	VARCHAR2	Name of the database service that is associated with an Oracle Scheduler job class used by the agent

MGW_AGENT_OPTIONS: Supplemental Options and Properties

This view lists supplemental options and properties for a Messaging Gateway agent.

Table C-14 MGW_AGENT_OPTIONS View

Column	Type	Description
`AGENT_NAME`	`VARCHAR2`	Name of the Messaging Gateway agent
`ENCRYPTED`	`VARCHAR2`	Indicates whether the value is stored as encrypted: • `TRUE` if the value is stored encrypted • `FALSE` if the value is stored as cleartext
`NAME`	`VARCHAR2`	Name of the option
`TYPE`	`VARCHAR2`	Option type or usage: `JAVA_SYSTEM_PROP` if the option is used to set a Java System property
`VALUE`	`VARCHAR2`	Value for the option. This will be <<`ENCRYPTED`>> if the value is stored in an encrypted form.

MGW_LINKS: Names and Types of Messaging System Links

This view lists the names and types of messaging system links currently defined.

Table C-15 MGW_LINKS View Properties

Name	Type	Description
`AGENT_NAME`	`VARCHAR2`	Name of the Messaging Gateway agent that will process propagation jobs for this link
`LINK_COMMENT`	`VARCHAR2`	User comment for the link
`LINK_NAME`	`VARCHAR2`	Name of the messaging system link
`LINK_TYPE`	`VARCHAR2`	Type of messaging system link. Values • `MQSERIES` is for WebSphere MQ links. • `TIBRV` is for TIB/Rendezvous links.

MGW_MQSERIES_LINKS: WebSphere MQ Messaging System Links

This view lists information for the WebSphere MQ messaging system links. The view includes most of the messaging system properties specified when the link is created.

Table C-16 MGW_MQSERIES_LINKS View Properties

Name	Type	Description
`AGENT_NAME`	`VARCHAR2`	Name of the Messaging Gateway agent that will process propagation jobs for this link
`CHANNEL`	`VARCHAR2`	Connection channel
`HOSTNAME`	`VARCHAR2`	Name of the WebSphere MQ host
`INBOUND_LOG_QUEUE`	`VARCHAR2`	Inbound propagation log queue

Table C-16 (Cont.) MGW_MQSERIES_LINKS View Properties

Name	Type	Description
`INTERFACE_TYPE`	`VARCHAR2`	Messaging interface type. Values: • `BASE_JAVA` is for WebSphere MQ Base Java interface • `JMS_CONNECTION` is for WebSphere MQ JMS unified, domain-independent connections • `JMS_QUEUE_CONNECTION` is for WebSphere MQ JMS queue connections • `JMS_TOPIC_CONNECTION` is for WebSphere MQ JMS topic connections
`LINK_COMMENT`	`VARCHAR2`	User comment for the link
`LINK_NAME`	`VARCHAR2`	Name of the messaging system link
`MAX_CONNECTIONS`	`NUMBER`	Maximum number of messaging connections
`OPTIONS`	`SYS.MGW_PROPERTIES`	Link options
`OUTBOUND_LOG_QUEUE`	`VARCHAR2`	Outbound propagation log queue
`PORT`	`NUMBER`	Port number
`QUEUE_MANAGER`	`VARCHAR2`	Name of the WebSphere MQ queue manager

MGW_TIBRV_LINKS: TIB/Rendezvous Messaging System Links

This view lists information for TIB/Rendezvous messaging system links. The view includes most of the messaging system properties specified when the link was created.

Table C-17 MGW_TIBRV_LINKS View Properties

Property Name	Type	Description
`AGENT_NAME`	`VARCHAR2`	Name of the Messaging Gateway agent that will process propagation jobs for this link
`CM_LEDGER`	`VARCHAR2`	TIB/Rendezvous CM ledger file name
`CM_NAME`	`VARCHAR2`	TIB/Rendezvous CM correspondent name
`DAEMON`	`VARCHAR2`	TIB/Rendezvous daemon parameter for RVD transport
`LINK_COMMENT`	`VARCHAR2`	User comment for the link
`LINK_NAME`	`VARCHAR2`	Name of the messaging system link
`NETWORK`	`VARCHAR2`	TIB/Rendezvous network parameter for rvd transport
`OPTIONS`	`SYS.MGW_PROPERTIES`	Link options
`SERVICE`	`VARCHAR2`	TIB/Rendezvous service parameter for rvd transport

MGW_FOREIGN_QUEUES: Foreign Queues

This view lists information for foreign queues. The view includes most of the queue properties specified when the queue is registered.

Table C-18 MGW_FOREIGN_QUEUES View Properties

Name	Type	Description
DOMAIN	VARCHAR2	Queue domain type. Values: • NULL means the queue domain type is automatically determined by the messaging system • QUEUE is for a queue (point-to-point) model • TOPIC is for a topic (publish-subscribe) model
LINK_NAME	VARCHAR2	Name of the messaging system link
NAME	VARCHAR2	Name of the registered queue
OPTIONS	SYS.MGW_PROPERTIES	Optional queue properties
PROVIDER_QUEUE	VARCHAR2	Message provider (native) queue name
QUEUE_COMMENT	VARCHAR2	User comment for the foreign queue

MGW_JOBS: Messaging Gateway Propagation Jobs

This view lists information for Messaging Gateway propagation jobs. The view includes most of the job properties specified when the propagation job was created, as well as other status and statistical information.

Table C-19 MGW_JOBS View

Column	Type	Description
AGENT_NAME	VARCHAR2	Name of the Messaging Gateway agent that processes this job
COMMENTS	VARCHAR2	Comments for the propagation job
DESTINATION	VARCHAR2	Destination queue to which messages are propagated
ENABLED	VARCHAR2	Indicates whether the job is enabled or not: • TRUE if the job is enabled • FALSE if the job is disabled
EXCEPTION_QUEUE	VARCHAR2	Exception queue used for propagation logging purposes
EXCEPTIONQ_MSGS	NUMBER	Option type or usage: JAVA_SYSTEM_PROP if the option is used to set a Java System property
FAILURES	NUMBER	Number of messages moved to exception queue since the last time the agent was started
JOB_NAME	VARCHAR2	Name of the propagation job
LAST_ERROR_MSG	VARCHAR2	Message for the last propagation error
LAST_ERROR_DATE	DATE	Date of the last propagation error
LAST_ERROR_TIME	VARCHAR2	Time of the last propagation error
LINK_NAME	VARCHAR2	Name of the Messaging Gateway link used by this job
OPTIONS	SYS.MGW_PROPERTIES	Job options
POLL_INTERVAL	INTEGER	Propagation poll interval (in seconds)
PROPAGATED_MSGS	NUMBER	Number of messages propagated since the last time the agent was started

Table C-19 (Cont.) MGW_JOBS View

Column	Type	Description
PROP_STYLE	VARCHAR2	Message propagation style: • NATIVE for native message propagation • JMS for JMS message propagation
PROPAGATION_TYPE	VARCHAR2	Propagation type: • OUTBOUND is for Oracle Database AQ to non-Oracle propagation • INBOUND is for non-Oracle to Oracle Database AQ propagation
RULE	VARCHAR2	Subscription rule used for the propagation source
SOURCE	VARCHAR2	Source queue from which messages are propagated
STATUS	VARCHAR2	Job status: • READY means the job is ready for propagation. The job must be enabled and the Messaging Gateway agent running before messages are actually propagated. • RETRY means the agent encountered errors when attempting to propagate messages for the job and will retry the operation • FAILED means the job has failed and agent has stopped trying to propagate messages. Usually this is due to an unrecoverable error or the propagation failure limit being reached. The job must be reset before the agent will attempt to propagate messages. The job is automatically reset each time the agent is started and can be manually reset by DBMS_MGWADM.RESET_JOB. • DELETE_PENDING means that job removal is pending. DBMS_MGWADM.REMOVE_JOB has been called but certain cleanup tasks for this job are still outstanding. • SUBSCRIBER_DELETE_PENDING means that removal is pending for the subscriber associated with the job. DBMS_MGWADM.REMOVE_SUBSCRIBER has been called but certain cleanup tasks are still outstanding.
TRANSFORMATION	VARCHAR2	Transformation used for message conversion

MGW_SUBSCRIBERS: Information for Subscribers

This view lists configuration and status information for Messaging Gateway subscribers. The view includes most of the subscriber properties specified when the subscriber is added, as well as other status and statistical information.

Table C-20 MGW_SUBSCRIBERS View Properties

Name	Type	Description
DESTINATION	VARCHAR2	Destination queue to which messages are propagated
EXCEPTIONQ_MSGS	NUMBER	Number of messages moved to the propagation exception queue since the last time the agent was started
EXCEPTION_QUEUE	VARCHAR2	Exception queue used for logging purposes
FAILURES	NUMBER	Number of propagation failures
LAST_ERROR_DATE	DATE	Date of last propagation error
LAST_ERROR_MSG	VARCHAR2	Message for last propagation error

Table C-20 (Cont.) MGW_SUBSCRIBERS View Properties

Name	Type	Description
LAST_ERROR_TIME	VARCHAR2	Time of last propagation error
OPTIONS	SYS.MGW_PROPERTIES	Subscriber options
PROP_STYLE	VARCHAR2	Message propagation style. Values: • NATIVE is for native message propagation • JMS is for JMS message propagation
PROPAGATED_MSGS	NUMBER	Number of messages propagated to the destination queue since the last time the agent was started
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: • OUTBOUND is for Oracle Database AQ to non-Oracle propagation • INBOUND is for non-Oracle to Oracle Database AQ propagation
QUEUE_NAME	VARCHAR2	Subscriber source queue
RULE	VARCHAR2	Subscription rule
STATUS	VARCHAR2	Subscriber status. Values: • ENABLED means the subscriber is enabled • DELETE_PENDING means subscriber removal is pending, usually because DBMS_MGWADM.REMOVE_SUBSCRIBER has been called but certain cleanup tasks pertaining to this subscriber are still outstanding
SUBSCRIBER_ID	VARCHAR2	Propagation subscriber identifier
TRANSFORMATION	VARCHAR2	Transformation used for message conversion

MGW_SCHEDULES: Information About Schedules

This view lists configuration and status information for Messaging Gateway schedules. The view includes most of the schedule properties specified when the schedule is created, as well as other status information.

Table C-21 MGW_SCHEDULES View Properties

Name	Type	Description
DESTINATION	VARCHAR2	Propagation destination
LATENCY	NUMBER	Propagation window latency (in seconds)
NEXT_TIME	VARCHAR2	Reserved for future use
PROPAGATION_TYPE	VARCHAR2	Propagation type. Values: • OUTBOUND is for Oracle Database AQ to non-Oracle propagation • INBOUND is for non-Oracle to Oracle Database AQ propagation
PROPAGATION_WINDOW	NUMBER	Reserved for future use
SCHEDULE_DISABLED	VARCHAR2	Indicates whether the schedule is disabled. Y means the schedule is disabled. N means the schedule is enabled.
SCHEDULE_ID	VARCHAR2	Propagation schedule identifier
SOURCE	VARCHAR2	Propagation source
START_DATE	DATE	Reserved for future use

Table C-21 (Cont.) MGW_SCHEDULES View Properties

Name	Type	Description
`START_TIME`	`VARCHAR2`	Reserved for future use

D

Advanced Queuing Sharded Queues

Table D-1 Comparison of Oracle Database Advanced Queuing Programmatic Interfaces for Sharded Queues: Administrative Interface

Use Case	PL/SQL	Java (JMS)
Create a sharded queue	`DBMS_AQADM.CREATE_SHARDED_QUEUE`	`AQjmsDestination.createJMSShardedQueue`
Drop a sharded queue	`DBMS_AQADM.DROP_SHARDED_QUEUE`	`AQjmsDestination.dropJMSShardedQueue`
Alter a sharded queue	`DBMS_AQADM.ALTER_SHARDED_QUEUE`	None. Use PL/SQL API.

Managing Sharded Queues

These topics describe how to manage sharded queues.

Note:

Starting and stopping a sharded queue use the same APIs as non-sharded queues.

- Creating a Sharded Queue
- Dropping a Sharded Queue
- Altering a Sharded Queue
- Setting a Queue Parameter
- Unsetting a Queue Parameter
- Getting a Queue Parameter
- Creating an Exception Queue

Creating a Sharded Queue

The `CREATE_SHARDED_QUEUE` API creates a sharded queue.

```
PROCEDURE CREATE_SHARDED_QUEUE (
    queue_name              IN VARCHAR2,
    storage_clause          IN VARCHAR2       DEFAULT NULL,
    multiple_consumers      IN BOOLEAN        DEFAULT FALSE,
    max_retries             IN NUMBER         DEFAULT NULL,
    comment                 IN VARCHAR2       DEFAULT NULL,
    queue_payload_type      IN VARCHAR2       DEFAULT JMS_TYPE,
    queue_properties        IN QUEUE_PROPS_T  DEFAULT NULL,
    replication_mode        IN BINARY_INTEGER DEFAULT NONE);
```

It has the following parameters:

Parameter	Description
`queue_name`	This required parameter specifies the name of the new queue. Maximum of 128 characters allowed.
`storage_clause`	The storage parameter is included in the `CREATE TABLE` statement when the queue table is created. The `storage_clause` argument can take any text that can be used in a standard `CREATE TABLE storage_clause` argument. The storage parameter can be made up of any combinations of the following parameters: `PCTFREE`, `PCTUSED`, `INITRANS`, `MAXTRANS`, `TABLESPACE`, `LOB`, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See *Oracle Database SQL Language Reference* for the usage of these parameters.
`multiple_consumers`	`FALSE` means queues can only have one consumer for each message. This is the default. `TRUE` means queues created in the table can have multiple consumers for each message.
`max_retries`	This optional parameter limits the number of times that a dequeue can reattempted on a message after a failure. The maximum value of `max_retries` is `2**31 -1`. After the retry limit has been exceeded, the message will be purged from the queue. `RETRY_COUNT` is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including `ALTER SYSTEM KILL SESSION`) or `SHUTDOWN ABORT` on the instance, then `RETRY_COUNT` is not incremented.
`comment`	This optional parameter is a user-specified description of the queue table. This user comment is added to the queue catalog.
`queue_payload_type`	Payload can be `RAW`, `DBMS_AQADM.JMS_TYPE`, or an object type. Default is `DBMS_AQADM.JMS_TYPE`.
`queue_properties`	Properties such as Normal or Exception Queue, Retry delay, retention time, sort list and cache hint. See also *Oracle Database PL/SQL Packages and Types Reference* for more information about `queue_properties`.
`replication_mode`	Reserved for future use. `DBMS_AQADM.REPLICATION_MODE` if queue is being created in the Replication Mode or else `DBMS_AQADM.NONE`. Default is `DBMS_AQADM.NONE`.

Dropping a Sharded Queue

This procedure drops an existing sharded queue from the database queuing system. You must stop the queue before calling `DROP_SHARDED_QUEUE`. User must stop the queue explicitly if force is set to `FALSE` before calling `DROP_SHARDED_QUEUE`. If force is set to `TRUE` then queue will be stopped internally and then dropped.

Syntax

```
DBMS_AQADM.DROP_SHARDED_QUEUE(
       queue_name IN VARCHAR2,
       force      IN BOOLEAN DEFAULT FALSE )
```

Parameters

Table D-2 CREATE_SHARDED_QUEUE Procedure Parameters

Parameter	Description
`queue_name`	This required parameter specifies the name of the sharded queue.
`force`	The sharded queue is dropped even if the queue is not stopped.

Altering a Sharded Queue

This procedure provides user the ability to alter *`queue_properties`* of a sharded queue.

Syntax

```
PROCEDURE ALTER_SHARDED_QUEUE(
    queue_name              IN VARCHAR2,
    max_retries             IN NUMBER         DEFAULT NULL,
    comment                 IN VARCHAR2       DEFAULT NULL,
    queue_properties        IN QUEUE_PROPS_T  DEFAULT NULL,
    replication_mode        IN BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table D-3 ALTER_SHARDED_QUEUE Procedure Parameters

Parameter	Description
`queue_name`	This parameter specifies the name of the sharded queue. A maximum of 128 characters are allowed.
`max_retries`	The maximum number of retries allowed.
`comment`	The parameter comment.
`queue_properties`	Properties such as Normal or Exception Queue, Retry delay, retention time, sort list and cache hint. See also *Oracle Database PL/SQL Packages and Types Reference* for more information about `queue_properties`.
`replication_mode`	Reserved for future use. `DBMS_AQADM.REPLICATION_MODE` if queue is being altered to be in the Replication Mode or else `DBMS_AQADM.NONE`. Default is `NULL`.

Setting a Queue Parameter

This procedure allows user to set different parameters for sharded queues at queue or database level. For database level the `queue_name` should be `NULL`. Note that queue overrides database level parameter values.

Syntax

```
  PROCEDURE  SET_QUEUE_PARAMETER(
    queue_name          IN VARCHAR2,
    param_name          IN VARCHAR2,
    param_value         IN NUMBER);
```

Parameters

Table D-4 SET_QUEUE_PARAMETER Procedure Parameters

Parameter	Description
`queue_name`	The name of the sharded queue.
`param_name`	The name of the parameter.
`param_value`	The value of the parameter.

Unsetting a Queue Parameter

This procedure allows user to unset different parameters for sharded queues at queue or database level. For database level the `queue_name` should be `NULL`. Note that queue overrides database level parameter values.

Syntax

```
PROCEDURE  UNSET_QUEUE_PARAMETER(
  queue_name           IN VARCHAR2,
  param_name           IN VARCHAR2);
```

Parameters

Table D-5 UNSET_QUEUE_PARAMETER Procedure Parameters

Parameter	Description
`queue_name`	The name of the sharded queue.
`param_name`	The name of the parameter.

Getting a Queue Parameter

This procedure allows user to get different parameters for sharded queues at queue or database level. For database level the `queue_name` should be `NULL`. Note that queue overrides database level parameter values.

Syntax

```
PROCEDURE  GET_QUEUE_PARAMETER(
  queue_name           IN VARCHAR2,
  param_name           IN VARCHAR2,
  param_value          OUT NUMBER);
```

Parameters

Table D-6 GET_QUEUE_PARAMETER Procedure Parameters

Parameter	Description
`queue_name`	The name of the sharded queue.
`param_name`	The name of the parameter.

Table D-6 (Cont.) GET_QUEUE_PARAMETER Procedure Parameters

Parameter	Description
`param_value`	The value of the parameter.

Creating an Exception Queue

This procedure allows a user to create an exception queue for a sharded queue.

Syntax

```
PROCEDURE CREATE_EXCEPTION_QUEUE(
  sharded_queue_name     IN VARCHAR2,
  exception_queue_name   IN VARCHAR2 DEFAULT NULL
  );
```

Parameters

Table D-7 CREATE_EXCEPTION_QUEUE Procedure Parameters

Parameter	Description
`sharded_queue_name`	The name of the sharded queue.
`exception_queue_name`	The name of the exception queue.