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.