5.5 DBMS_AQADM: Performing AQ Administrative Tasks (Oracle8 only)

Before AQ users can enqueue and dequeue, they must have queues with which to work. The AQ administrator must create queue tables and queues within those tables and then start the queues. Additional administrative tasks include stopping queues and removing queue tables, managing lists of subscribers, and starting and stopping the Queue Monitor.

The DBMS_AQADM package provides an interface to the administrative tasks of Oracle AQ. The DBMS_AQADM programs are listed in Table 5.2 . In order to use these procedures, a DBMS_AQADM user must have been granted the role AQ_ADMINISTRATOR_ROLE from the SYS account.

> **NOTE:** None of the DBMS_AQADM programs may be used inside SQL, directly or indirectly. The reason for this restriction is that many of these programs perform a COMMIT (unless you have set the auto_commit parameter to FALSE).

Table 5.2: DBMS_AQADM Programs

Name	Description	Use in SQL?
ADD_SUBSCRIBER	Adds a subscriber to a queue.	No
ALTER_QUEUE	Alters limited set of properties of an existing queue.	No
CREATE_QUEUE	Creates a queue within the specified queue table.	No
CREATE_QUEUE_TABLE	Creates a queue table in which queues can be defined.	No
DROP_QUEUE	Drops an existing queue table.	No
DROP_QUEUE_TABLE	Drops an existing queue from a queue table.	No

Name	Description	Use in SQL?
GRANT_TYPE_ACCESS	Grants access to multiple consumer tasks.	No
QUEUE_SUBSCRIBERS	Retrieves the list of subscribers associated with a queue.	No
REMOVE_SUBSCRIBER	Removes a subscriber from a queue.	No
SCHEDULE_PROPAGATION	Schedules propagation of messages from a source queue to a destination queue (either in the same or a remote database).	No
START_QUEUE	Enables the queue (i.e., permits enqueue and/or dequeue operations).	No
START_TIME_MANAGER	Starts the Queue Monitor process.	No
STOP_QUEUE	Disables the queue for enqueuing, dequeuing, or both.	No
STOP_TIME_MANAGER	Stops the Queue Monitor process.	No
UNSCHEDULE_PROPAGATION	Unschedules the propagation of messages from a source queue to a destination queue.	No
VERIFY_QUEUE_TYPES	Determines whether two different queues have the same payload type. You can perform this verification for two local queues or for a local and a remote queue.	No

The following sections describe these programs in a number of categories.

5.5.1 Creating Queue Tables

First, you need to create your queue tables and grant the necessary capabilities.

5.5.1.1 The DBMS_AQADM. GRANT_TYPE_ACCESS procedure

If you would like to support multiple consumers with your queue (that is, so that the same or different messages can be dequeued by more than one agent), call the GRANT_TYPE_ACCESS program to grant that capability,

```
PROCEDURE DBMS_AQADM.GRANT_TYPE_ACCESS (user_name IN
VARCHAR2);
```

where user_name is the name of the user who needs to perform multiple consumer queue activities.

The SYS account must call this procedure to grant the privilege to the AQ administrator. The AQ administrator then runs this program to grant the privilege to AQ users. Here is an example of granting multiple consumer capabilities to the SCOTT account:

```
SQL> exec DBMS_AQADM.GRANT_TYPE_ACCESS ('scott');
```

5.5.1.2 The DBMS_AQADM. CREATE_QUEUE_TABLE procedure

The CREATE_QUEUE_TABLE procedure creates a queue table. A queue table is the named repository for a set of queues and their messages. A queue table may contain numerous queues, each of which may have many messages. But a given queue and its messages may exist in only one queue table. The specification for the procedure follows:

```
PROCEDURE 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);
```

Parameters are summarized in the following table.

Name	Description
queue_table	Name of a queue table to be created. Maximum of 24 characters in length, unless you have combined the schema and the table name, as in `SCOTT.MSG_TABLE', in which case the maximum length of the argument is 49 characters.
queue_payload_type	Type of the user data stored. This is either the name of an already defined object type or it is the string "RAW," indicating that the payload for this queue table will consist of a single LOB column.
storage_clause	Storage parameter, which will be included in the CREATE TABLE statement when the queue table is created. The storage parameter can be made up of any combination of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLEPSACE, LOB, and a table storage clause. Refer to the *Oracle SQL Reference Guide* for more information about defining a storage parameter.
sort_list	Columns to be used as the sort key in ascending order; see the following discussion.
multiple_consumers	Specifies whether queues created in this table can have more than one consumer per message. The default is FALSE (only one consumer per message). If a value of TRUE is passed for this argument, the user must have been granted type access by executing the DBMS_AQADM.GRANT_TYPE_ACCESS procedure.
message_grouping	Message grouping behavior for queues created in the table. Valid arguments for this parameter: *DBMS_AQADM.NONE* Each message is treated individually (this is the default).

Name	Description
	DBMS_AQADM.TRANSACTIONAL Messages enqueued as part of one transaction are considered part of the same group and must be dequeued as a group of related messages.
comment	User-specified description of the queue table. This user comment will be added to the queue catalog.
auto_commit	Specifies the transaction behavior for queues associated with this table. Valid arguments: *TRUE* Causes the current transaction, if any, to commit before the operation is carried out. The operation becomes persistent when the call returns. This is the default. *FALSE* Any administrative operation on a queue is part of the current transaction and will become persistent only when the caller issues a commit. In other words, this argument does not apply to enqueue/dequeue operations performed on the queue.

The sort_list has the following format,

```
'<sort_column_1>,<sort_column_2>'
```

where each sort_column_N is either PRIORITY or ENQ_TIME. If both columns are specified, then <sort_column_1> defines the most significant order. In other words, these are the only valid values for sort_list besides NULL:

```
'PRIORITY'
'PRIORITY,ENQ_TIME'
'ENQ_TIME' (this is the default)
'ENQ_TIME,PRIORITY'
```

Once a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same default ordering. This order cannot be altered once the queue table has been created. If no sort list is specified, all the queues in this queue table will be sorted by the enqueue time in ascending order. This order is equivalent to FIFO (first-in-first-out) order.

Even with the default ordering defined, a consumer can override this order by specifying the msgid or correlation value for a specific message. The msgid, correlation, and sequence_deviation take precedence over the default dequeueing order if they are specified.

When you create a queue table, the following objects are created:

- The default exception queue associated with the queue table. It is named,

  ```
  aq$_<queue_table_name>_e
  ```

  where `<queue_table_name>` is the name of the queue table just created. So if my

queue table is named msg, then my exception queue table is named aq$_msg_e (and now you can see why the maximum length name for a queue table is 24 characters!).

- A read-only view that is used by AQ applications for querying queue data called:

```
aq$<queue_table_name>.
```

- An index for the Queue Monitor operations called:

```
aq$_<queue_table_name>_t.
```

- An index (or an index organized table [IOT] in the case of multiple consumer queues) for dequeue operations. It is named:

```
aq$_<queue_table_name>_i
```

5.5.1.2.1 Example

In the following example, I construct a basic queue table in the current schema with a comment as to when it was created:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE
      (queue_table => 'msg',
       queue_payload_type => 'message_type',
       comment => 'General message queue table created on '
||
          TO_CHAR(SYSDATE,'MON-DD-YYYY HH24:MI:SS'));
END;
/
```

Notice that I pass the payload type as a string: the name of the object type I defined in the section explaining how to enqueue messages. I can verify the creation of this queue table by querying the USER_QUEUE_TABLES.

```
SQL> SELECT queue_table, user_comment FROM
USER_QUEUE_TABLES;

QUEUETABLE

USER_COMMENT

MSG                            General message queue table
created on JUN-08-1997 14:22:01
```

The following request to create a queue table specifies support for multiple consumers of a single message and also enables message grouping:

```
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE
      (queue_table => 'msg',
       queue_payload_type => 'message_type',
       multiple_consumers => TRUE,
       message_grouping => DBMS_AQADM.TRANSACTIONAL,
       comment => 'Specialized queue table created on ' ||
          TO_CHAR(SYSDATE,'MON-DD-YYYY HH24:MI:SS'));
END;
/
```

Notice the extensive use of named notation (the "=>" symbols). This feature of PL/SQL comes in very handy when working with programs that have long lists of parameters, or with programs that are used infrequently. The named notation approach, which explicitly associates a parameter with an argument value, documents more clearly how the program is being used.

See the Section 5.7 " section for a more thorough explanation of the message grouping and multiple consumers feature.

5.5.1.2.2 Notes on usage

Note the following about using the CREATE_QUEUE_TABLE procedure:

- If you specify a schema for your queue table, then the payload type (if an object type) must also be defined in that schema.
- You do not need to specify the schema for the payload type, but you can. If you do specify the object type schema for the payload type, it must be the same schema as that of the queue table or you will receive an ORA-00902: invalid datatype error.
- If you are going to create your queue tables from one (administrator) account and manage those queues from another (user) account, then you will need to grant EXECUTE privilege to the administrator account on the queue message types from the user account.

5.5.2 Creating and Starting Queues

Once a queue table has been created, you can create and then start individual queues in that queue table.

5.5.2.1 The DBMS_AQADM. CREATE_QUEUE procedure

Call the CREATE_QUEUE to create a queue in the specified queue table. All queue names must be unique within a schema. Once a queue is created, it can be enabled by calling DBMS_AQADM.START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

```
PROCEDURE DBMS_AQADM.CREATE_QUEUE
   (queue_name IN VARCHAR2,
    queue_table IN VARCHAR2,
    queue_type IN BINARY_INTEGER default
DBMS_AQADM.NORMAL_QUEUE,
    max_retries IN NUMBER default 0,
    retry_delay IN NUMBER default 0,
    retention_time IN NUMBER default 0,
    dependency_tracking IN BOOLEAN default FALSE,
    comment IN VARCHAR2 default NULL,
    auto_commit IN BOOLEAN default TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the queue to be created. All queue names must be unique within a schema.
queue_table	Name of the queue table in which this queue is to be created.

Name	Description
queue_type	Specifies the type of queue. Valid options for this parameter: *DBMS_AQADM.NORMAL_QUEUE* The default, a normal queue. *DBMS_AQADM.EXCEPTION_QUEUE* An exception queue. Only dequeue operations are valid on an exception queue.
max_retries	Maximum number of times a dequeue with the REMOVE mode can be attempted on a message. The count is incremented when the application issues a rollback after executing a dequeue (but before a commit is performed). The message is moved to the exception queue when the number of failed attempts reaches its max_retries. Specify 0, the default, to indicate that no retries are allowed.
retry_delay	Delay in seconds to wait after an application rollback before the message is scheduled for processing again. The default of 0 means that the dequeue operation should be retried as soon as possible. The retry parameters in effect allow you to control the fault tolerance of the message queue. This value is ignored if max_retries is set to 0 (the default). This value may not be specified if this is a multiple consumer queue table.
retention_time	Number of seconds for which a message will be retained in the queue table after being dequeued from the queue. The default is 0, meaning no retention. You can also specify the DBMS_AQADM.INFINITE packaged constant to request that the message be retained forever.
dependency_tracking	Reserved for future use by Oracle Corporation. FALSE is the default, and TRUE is not permitted.
comment	User comment to associate with the message queue in the queue catalog.
auto_commit	Specify TRUE (the default) to cause the current transaction, if any, to commit before the operation is carried out. The operation becomes persistent when the call returns. Specify FALSE to make this operation part of the current transaction. In this case, it will become persistent only when the caller issues a commit.

5.5.2.1.1 Example

In the following example, I create a new message queue within the previously created message queue table. I want it to allow for up to ten retries at hourly delays and keep ten days worth of history before deleting processed messages.

```
BEGIN
  DBMS_AQADM.CREATE_QUEUE(
       queue_name => 'never_give_up_queue',
       queue_table => 'msg',
       max_retries => 10,
```

```
      retry_delay => 3600,
      retention_time => 10 * 24 * 60 * 60,
      comment => 'Test Queue Number 1');



END;
/
```

5.5.2.2 The DBMS_AQADM. START_QUEUE procedure

It is not enough to simply create a queue inside a queue table. You must also enable it for enqueuing and/or dequeuing operation, with the START_QUEUE procedure:

```
PROCEDURE DBMS_AQADM.START_QUEUE
   (queue_name IN VARCHAR2,
    enqueue IN BOOLEAN DEFAULT TRUE,
    dequeue  IN BOOLEAN DEFAULT TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the queue to be started.
enqueue	Flag indicating whether the queue should be enabled for enqueuing (TRUE means enable, the default; FALSE means do not alter the current setting.)
dequeue	Flag indicating whether the queue should be enabled for dequeuing (TRUE means enable, the default; FALSE means do not alter the current setting.)

Notice that a value of FALSE for either of the Boolean parameters does not disable a queue for the respective operation. It simply does not change the current status of that operation on the specified queue. To disable queuing or enqueuing on a queue, you must call the DBMS_AQADM.STOP_QUEUE procedure.

5.5.2.2.1 Example

The following block starts a queue for enqueuing actions only:

```
BEGIN
VP   DBMS_AQADM.START_QUEUE ('number_stack',
dequeue=>FALSE);



END;VPU
/
```

You will often want to perform the following steps in sequence:

DBMS_AQADMP.CREATE_QUEUE_TABLE
DBMS_AQADM.CREATE_QUEUE
DBMS_AQADM.START_QUEUE

The following files on the companion disk provide various examples of these steps:

aqcrepq.sql
aqcreq.sql
aqcref.sql

5.5.2.3 The DBMS_AQADM. ALTER_QUEUE procedure

The ALTER_QUEUE procedure of the DBMS_AQADM package modifies an existing message queue. An error is returned if the message queue does not exist. The specification for the procedure follows:

```
PROCEDURE DBMS_AQADM.ALTER_QUEUE
   queue_name IN VARCHAR2,
   max_retries IN NUMBER default NULL,
   retry_delay IN NUMBER default NULL,
   retention_time IN NUMBER default NULL,
   auto_commit IN BOOLEAN default TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the message queue to be altered.
max_retries	Specifies the maximum number of times a dequeue with the REMOVE mode can be attempted on a message. The count is incremented when the application issues a rollback after executing a dequeue (but before a commit is performed). The message is moved to the exception queue when the number of failed attempts reach its max_retries. Specify 0, the default, to indicate that no retries are allowed.
retry_delay	Specifies the delay in seconds to wait after an application rollback before the message is scheduled for processing again. The default of 0 means that the dequeue operation should be retried as soon as possible. The retry parameters in effect allow you to control the fault tolerance of the message queue. This value is ignored if max_retries is set to 0 (the default). This value may not be specified if this is a multiple consumer queue table.
retention_time	Specifies the number of seconds for which a message will be retained in the queue table after being dequeued from the queue. The default is 0, meaning no retention. You can also specify the DBMS_AQADM.INFINITE packaged constant to request that the message be retained forever.
auto_commit	Specifies the transaction behavior for queues associated with this table. Specify TRUE (the default) to cause the current transaction, if any, to commit before the operation is carried out. The operation becomes persistent when the call returns. If you specify FALSE, any administrative operation on a queue is part of the current transaction and will become persistent only when the caller issues a commit. In other words, this argument does not apply to enqueue/dequeue operations performed on the queue.

5.5.2.3.1 Example

In the following example, I modify the properties of the queue created under CREATE_QUEUE. I now want it to allow for up to 20 retries at hourly delays and to keep 30 days worth of history before deleting processed messages.

```
BEGIN
  DBMS_AQADM.ALTER_QUEUE(
      queue_name => 'never_give_up_queue',
      max_retries => 20,
      retention_time => 30 * 24 * 60 * 60);
END;
/
```

I can verify the impact of this call by querying the USER_QUEUES data dictionary view.

```
SQL> SELECT name, max_retries, retention FROM USER_QUEUES;

NAME                           MAX_RETRIES        RETENTION
------------------------------     -----------
----------
AQ$_MSG_E                      0                  0
MSGQUEUE                       0                  0
NEVER_GIVE_UP_QUEUE            20                 2592000
```

The first line in the listing is the exception queue for the "msg" queue table. The "msgqueue" queue in the "msg" queue table is a previously defined queue. The third line displays the information for the queue modified by the call to DBMS_AQADM.ALTER_QUEUE.

5.5.3 Managing Queue Subscribers

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. A subscriber to a queue is an agent that is registered to dequeue messages from a queue.

You can add and remove subscribers, as well as retrieve the current set of subscribers for a queue. These operations will work only with queues that allow multiple consumers (i.e., the multiple_consumers parameter is set to TRUE when you called DBMS_AQADM.CREATE_QUEUE_TABLE). The command takes effect immediately, and the containing transaction is committed. Enqueue requests executed after the completion of this call will reflect the new behavior. Users attempting to modify the subscriber list of a queue must have been granted type access by executing the DBMS_AQADM.GRANT_TYPE_ACCESS procedure.

5.5.3.1 The DBMS_AQADM. ADD_SUBSCRIBER procedure

To add a subscriber to a queue, call the ADD_SUBSCRIBER procedure:

```
PROCEDURE DBMS_AQADM.ADD_SUBSCRIBER
   (queue_name IN VARCHAR2, subscriber IN SYS.AQ$_AGENT);
```

Parameters are summarized in the following table.

Name	Description

Name	Description
queue_table	Name of the queue to which the subscriber is being added.
subscriber	Subscriber to be added. Not the actual name of the subscriber, but an object of type SYS.AQ$_AGENT. If you try to add a subscriber that is already on the list, AQ will raise the ORA-24034 exception. Agent names are case-insensitive.

5.5.3.1.1 Example

Here is an example of adding a subscriber to a queue:

```
BEGIN
   DBMS_AQADM.ADD_SUBSCRIBER
      ('msgqueue', SYS.AQ$_AGENT ('multiconsqueue', NULL,
NULL));
```

In this case, I have embedded the call to the object constructor method to convert a name to an agent. You can also perform this task in two steps as follows:

```
DECLARE
   v_agent SYS.AQ$_AGENT;
BEGIN
   v_agent := SYS.AQ$_AGENT ('Danielle', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER ('multiconsqueue', v_agent);
```

5.5.3.2 The DBMS_AQADM. REMOVE_SUBSCRIBER procedure

To remove a default subscriber from a queue, call the REMOVE_SUBSCRIBER procedure:

```
PROCEDURE DBMS_AQADM.REMOVE_SUBSCRIBER
   (queue_name IN VARCHAR2, subscriber IN SYS.AQ$_AGENT);
```

Parameters are summarized in the following table.

Name	Description
queue_table	Name of the queue from which the subscriber is being removed.
subscriber	Subscriber to be removed. Not the actual name of the subscriber, but an object of type SYS.AQ$_AGENT. Agent names are case-insensitive.

5.5.3.2.1 Example

Here is an example of removing a subscriber from a queue:

```
BEGIN
   DBMS_AQADM.REMOVE_SUBSCRIBER
      ('multiconsqueue', SYS.AQ$_AGENT ('CEO', NULL, NULL));
```

In this case I have embedded the call to the object constructor method to convert a name to an agent. You can also perform this task in two steps as follows:

```
DECLARE
   v_agent SYS.AQ$_AGENT;
```

```
BEGIN
   v_agent := SYS.AQ$_AGENT ('CEO', NULL, NULL);
   DBMS_AQADM.REMOVE_SUBSCRIBER ('multiconsqueue', v_agent);
```

All references to the subscriber in existing messages are removed as part of the operation.

If you try to remove a subscriber that does not exist for this queue, you will receive this error message:

```
ORA-24035: application <subscriber> is not a subscriber for
queue <queue>
```

5.5.3.3 The DBMS_AQADM. QUEUE_SUBSCRIBERS procedure

The QUEUE_SUBSCRIBERS function returns the list of subscribers associated with the specified queue. This list is an index-by table, as shown in the header,

```
FUNCTION DBMS_AQADM.QUEUE_SUBSCRIBERS
   (queue_name IN VARCHAR2)
RETURN DBMS_AQADM.AQ$_SUBSCRIBER_LIST_T;
```

where queue_name is the name of the queue.

5.5.3.3.1 Example

The following procedure encapsulates the steps needed to obtain this list and then to display it:

```
/* Filename on companion disk:

showsubs.sp */*
CREATE OR REPLACE PROCEDURE showsubs (qname IN VARCHAR2)
IS
   sublist DBMS_AQADM.AQ$_SUBSCRIBER_LIST_T;
   v_row PLS_INTEGER;
BEGIN
   /* Retrieve the list. */
   sublist := DBMS_AQADM.QUEUE_SUBSCRIBERS (qname);

   v_row := sublist.FIRST;
   LOOP
      EXIT WHEN v_row IS NULL;
      DBMS_OUTPUT.PUT_LINE (v_row);
      DBMS_OUTPUT.PUT_LINE (sublist(v_row).name);
      v_row := sublist.NEXT (v_row);
   END LOOP;
END;
/
```

Now let's put the procedure to use. First of all, you can associate a set of subscribers only with a queue that supports multiple consumers. Here are the steps:

```
/* Filename on companion disk:

aqcremq.sql */*
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE
      (queue_table => 'multicons',
       queue_payload_type => 'message_type',
       multiple_consumers => TRUE);
```

```
   DBMS_AQADM.CREATE_QUEUE
      (queue_name => 'multiconsqueue',
       queue_table => 'multicons');

   DBMS_AQADM.START_QUEUE (queue_name => 'multiconsqueue');

END;
/
```

You can then add subscribers to the multicons queue and display the results:

```
/* Filename on companion disk:

showsubs.sql */*
DECLARE
   v_queue VARCHAR2(10) := 'multiconsqueue';
BEGIN
   DBMS_AQADM.ADD_SUBSCRIBER
      (v_queue, SYS.AQ$_AGENT ('Danielle', NULL, NULL));
   DBMS_AQADM.ADD_SUBSCRIBER
      (v_queue, SYS.AQ$_AGENT ('Benjamin', NULL, NULL));
   DBMS_AQADM.ADD_SUBSCRIBER
      (v_queue, SYS.AQ$_AGENT ('Masada', NULL, NULL));
   DBMS_AQADM.ADD_SUBSCRIBER
      (v_queue, SYS.AQ$_AGENT ('Timnah', NULL, NULL));



showsubs (v_queue);


END;
/
```

5.5.4 Stopping and Dropping Queues

DBMS_AQADM offers two programs to clean up queues: STOP_QUEUE and DROP_QUEUE. The stop program disables activity on the queue. The drop program actually removes that queue from the queue table.

5.5.4.1 The DBMS_AQADM. STOP_QUEUE procedure

To disable enqueuing and/or dequeuing on a particular queue, call the STOP_QUEUE procedure:

```
PROCEDURE DBMS_AQADM.STOP_QUEUE
   (queue_name IN VARCHAR2,
    enqueue IN BOOLEAN DEFAULT TRUE,
    dequeue IN BOOLEAN DEFAULT TRUE,
    wait IN BOOLEAN DEFAULT TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the queue to be stopped.
enqueue	Specify TRUE (the default) if you want to disable enqueuing on this queue. FALSE means that the current setting will not be altered.

Name	Description
dequeue	Specify TRUE (the default) if you want to disable dequeuing on this queue. FALSE means that the current setting will not be altered.
wait	If you specify TRUE (the default), then your program will wait for any outstanding transactions to complete. While waiting, no new transactions are allowed to enqueue to or dequeue from this queue. If you specify FALSE, then the program will return immediately and raise ORA-24023 if it was unable to stop the queue.

5.5.4.1.1 Example

The following example shows the disabling of a queue for enqueuing purposes only. I also request that the program wait until all outstanding transactions are completed. You might take these steps in order to allow consumers to empty the queue, while not allowing any new messages to be placed on the queue.

```
BEGIN
   DBMS_AQADM.STOP_QUEUE
      ('msgqueue', enqueue=>TRUE, dequeue=>FALSE,
wait=>TRUE);
END;
```

You can check the status of your queue by querying the USER_QUEUES data dictionary view:

```
SQL> SELECT name, enqueue, dequeue FROM USER_QUEUES
  2   WHERE name = 'MSGQUEUE';


NAME

ENQUEUE

DEQUEUE

MSGQUEUE         NO   YES
```

5.5.4.2 The DBMS_AQADM.DROP_QUEUE procedure

The DROP_QUEUE procedure drops an existing message queue. An error is returned if the message queue does not exist. In addition, this operation is not allowed unless DBMS_AQADM.STOP_QUEUE has been called to disable both enqueuing and dequeuing. If the message queue has not been stopped, then DROP_QUEUE returns an error of queue resource (ORA-24023). Here's the header for the procedure:

```
PROCEDURE DBMS_AQADM.DROP_QUEUE
   (queue_name  IN VARCHAR2,
    auto_commit IN BOOLEAN DEFAULT TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the queue to be dropped.

Name	Description
auto_commit	Specify TRUE (the default) to cause the current transaction, if any, to commit before the operation is carried out. The operation becomes persistent when the call returns. Specify FALSE if you want the drop action to be part of the current transaction, thereby taking effect only when the calling session issues a commit.

5.5.4.3 The DBMS_AQADM.DROP_QUEUE_TABLE procedure

Once you have stopped and dropped all queues in a queue table, you can remove that entire queue table with the DROP_QUEUE_TABLE procedure:

```
PROCEDURE DBMS_AQADM.DROP_QUEUE_TABLE
   (queue_table IN VARCHAR2,
    force IN BOOLEAN default FALSE,
    auto_commit IN BOOLEAN default TRUE);
```

Parameters are summarized in the following table.

Name	Description
queue_table	Name of the queue table to be dropped.
force	Specify FALSE (the default) to ensure that the drop action will not succeed unless all queues have been dropped. Specify TRUE if you want to force the dropping of this queue table. In this case, any remaining queues will be automatically stopped and dropped.
auto_commit	Specify TRUE (the default) to cause the current transaction, if any, to commit before the operation is carried out. The operation becomes persistent when the call returns. Specify FALSE if you want the drop action to be part of the current transaction, thereby taking effect only when the calling session issues a commit.

5.5.4.3.1 Example

The following example forces the dropping of the msg queue table, stopping and dropping all queues along the way.

```
BEGIN
   DBMS_AQADM.DROP_QUEUE_TABLE ('msg', force => TRUE);
END;
/
```

5.5.5 Managing Propagation of Messages

In order to propagate messages from one queue to another (an Oracle 8.0.4 and later feature), you need to schedule propagation between queues. You can also unschedule propagation of those messages.

5.5.5.1 The DBMS_AQADM.SCHEDULE_PROPAGATION procedure

Call the SCHEDULE_PROPAGATION procedure to schedule propagation of messages. The header for this procedure follows:

```
PROCEDURE DBMS_AQADM.SCHEDULE_PROPAGATION
   (src_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);
```

Parameters are summarized in the following table.

Name	Description
src_queue_name	Name of the source queue whose messages are to be propagated. This name should include the schema name, if the queue is not located in the default schema, which is the schema name of the Oracle AQ administrator.
destination	Database link for the destination. If this argument is NULL, then the destination is the local database; messages will be propagated to other queues in the local database as determined by the subscription or recipient lists. The maximum length for the destination is currently 128 bytes. If the name is not fully qualified, then the default domain named will be used.
start_time	Initial start date-time for the window of time in which messages are propagated from the source queue to the destination queue.
duration	Length of time in seconds that the propagation window will be open. If you supply a NULL value (the default), then the propagation window is open continually, or at least until the propagation is unscheduled through a call to DBMS_AQ.UNSCHEDULE_PROPAGATION.
next_time	An expression returning a date value that is used to compute the start of the next propagation window. If you supply a NULL value (the default), then propagation will stop when the current (initial) window closes. As an example, suppose you want to make sure the next propagation doesn't start until 24 hours after the last propagation window closed. You would then set next_time to SYSDATE + 1 - duration/86400. If you want to make sure the next propagation starts 24 hours after the last propagation window closed, provide a next_time value of `SYSDATE + 1'.
latency	The maximum wait in seconds that a message may sit in a queue after the start of a propagation window before it is propagated. The default amount of time is 60 seconds. This would mean, for example, that when the propagation window opens, the Queue Monitor propagates any messages present. It would then wait 60 seconds (or until the window closes due to the duration setting) before checking for the presence of and propagating any other messages.

To summarize that relatively complex set of parameters and their interactions, when you schedule propagation you identify an initial start date/time and a span of time (number of seconds) in which messages are available to be propagated out of the queue to other queues. You can request that this window of time be opened on a regular basis (every day, once a week, every morning at 10 a.m., etc.). Finally, you can specify that the Queue Monitor check no less frequently than every N seconds (latency) during the time the propagation window is open to see if there are messages to propagate.

5.5.5.1.1 Example

In this example, I schedule the propagation of a queue to the Boston brokerage office to occur every two hours. The propagation window is five minutes, and during that period of time, I want messages to be flushed out at least every 30 seconds.

```
BEGIN
   DBMS_AQADM.SCHEDULE_PROPAGATION
      ('sell_orders',
       'broker@boston',
       SYSDATE,
       5 * 60,
       'SYSDATE + 2/24',
       30);
END;
/
```

If I do not specify a destination, then propagation occurs to the same database in which the source queue is defined. The following call to DBMS_AQADM.SCHEDULE_PROPAGATION takes all default values (including a local destination database), except that it requests a latency of ten minutes by using named notation:

```
BEGIN
   DBMS_AQADM.SCHEDULE_PROPAGATION
      ('share_the_blame',
       latency => 60 * 10);



END;
/
```

5.5.5.2 The DBMS_AQADM.UNSCHEDULE_PROPAGATION procedure

You can stop or unschedule propagation of a queue with the UNSCHEDULE_PROPAGATION procedure:

```
PROCEDURE DBMS_AQADM.UNSCHEDULE_PROPAGATION
   (src_queue_name IN VARCHAR2,
    destination IN VARCHAR2 DEFAULT NULL);
```

Parameters are summarized in the following table:

Name	Description
src_queue_name	Name of the source queue whose messages are no longer to be propagated. This name should include the schema name if the queue is not located in the default schema, which is the schema name of the Oracle AQ administrator.
destination	Database link for the destination for which propagation will be terminated. If this argument is NULL, then the destination is the local database; messages will no longer be propagated to other queues in the local database as determined by the subscription or recipient lists. The maximum length for destination is currently 128 bytes. If the name is not fully qualified, then the default domain named will be used.

5.5.6 Verifying Queue Types

5.5.6.1 The DBMS_AQADM.VERIFY_QUEUE_TYPES procedure

The VERIFY_QUEUE_TYPES procedure allows you to determine whether two different queues have the same payload type. Here's the header:

```
PROCEDURE DBMS_AQADM.VERIFY_QUEUE_TYPES
   (src_queue_name IN VARCHAR2,
    dest_queue_name IN VARCHAR2
    destination IN VARCHAR2 DEFAULT NULL,
    rc OUT BINARY_INTEGER);
```

Parameters are summarized in the following table.

Name	Description
src_queue_name	Name of the source queue, usually one from which messages are to be propagated. This name should include the schema name if the queue is not located in the default schema, which is the schema name of the user.
dest_queue_name	Name of the destination queue, usually one to which messages are to be propagated. This name should include the schema name if the queue is not located in the default schema, which is the schema name of the user.
destination	Database link for the destination queue. If this argument is NULL, then the destination queue is located in the same database as the source queue. The maximum length for destination is currently 128 bytes. If the name is not fully qualified, then the default domain named will be used.
rc	Status code returned by the procedure. If there is no problem verifying the queue types and if the source and destination queue types match, then the result code is 0. If the queue types do not match (and there is no error), then the result code is 1. If an Oracle exception is raised, the SQLCODE is returned in rc.

Whenever this program is run (either by Oracle AQ itself or by an AQ administrator), it updates the SYS.AQ$_MESSAGE_TYPES table. You can access this table (to verify the success of the type match after propagation has taken place) using the OID (Object ID) of the source queue and the address of the destination queue.

5.5.7 Starting and Stopping the Queue Monitor

If you want to use the delay and expiration features of AQ, you must have the Queue Monitor process running in the background. Before you can do this, you must add an AQ_TM_PROCESS parameter to your database initialization file (see Section 5.2, "Getting Started with Oracle AQ" " at the beginning of this chapter for more information).

5.5.7.1 The DBMS_AQADM. START_TIME_MANAGER procedure

To start the Queue Monitor process, call the START_TIME_MANAGER procedure:

```
PROCEDURE DBMS_AQADM.START_TIME_MANAGER;
```

The operation takes effect when the call completes; there are no transactional dependencies.

You can use the START_TIME_MANAGER to restart the Queue Monitor after you stopped it with a call to STOP_TIME_MANAGER. You can also use it to start the Queue Monitor process if the database initialization parameter was set to 0. In other words, you can override the default state of the database with this programmatic call.

5.5.7.2 The DBMS_AQADM. STOP_TIME_MANAGER procedure

To stop the Queue Monitor process, call the STOP_TIME_MANAGER procedure:

```
PROCEDURE DBMS_AQADM.STOP_TIME_MANAGER;
```

The operation takes effect when the call completes; there are no transactional dependencies.

The STOP_TIME_MANAGER procedure does not actually stop the physical process running in the background. This process is started when the database is initialized. The procedure simply disables the time management features of Oracle AQ in that database instance.

PREVIOUS 5.4 DBMS_AQ: Interfacing to Oracle AQ (Oracle8 only) | HOME BOOK INDEX | NEXT 5.6 Oracle AQ Database Objects

Library Home | Oracle PL/SQL Programming, Second Edition | Oracle PL/SQL Programming: Guide to Oracle8i Features | Oracle Built-in Packages | Advanced PL/SQL Programming with Packages | Oracle Web Applications: PL/SQL Developer's Introduction | Oracle PL/SQL Language Pocket Reference | Oracle PL/SQL Built-ins Pocket Reference