5.4 DBMS_AQ: Interfacing to Oracle AQ (Oracle8 only)

The DBMS_AQ package provides an interface to the operational tasks of Oracle AQ as performed by the programs listed in Table 5.1 . To use these programs, you must have been granted the new role AQ_USER_ROLE.

Table 5.1: DBMS_AQ Programs

Name	Description	SQL?
ENQUEUE	Adds a message to the specified queue.	No
DEQUEUE	Retrieves a message from the specified queue.	No

The following sections describe how to call these programs.

5.4.1 Enqueuing Messages

The ENQUEUE procedure allows you to add a message to a specified queue.

5.4.1.1 The DBMS_AQ. ENQUEUE procedure

Use the ENQUEUE procedure to add a message to a particular queue. Here's the header for the procedure:

```
PROCEDURE DBMS_AQ.ENQUEUE
   (queue_name IN VARCHAR2,
    enqueue_options IN DBMS_AQ.ENQUEUE_OPTIONS_T,
    message_properties IN DBMS_AQ.MESSAGE_PROPERTIES_T,
    payload IN <type_name>,
    msgid OUT RAW);
```

Parameters are summarized in the following table.

Name	Description
queue_name	Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue and must have been previously defined by a call to DBMS_AQADM.CREATE_QUEUE.
enqueue_options	A record containing the enqueuing options, defined using the specified record type. See Section 5.3.7, "Enqueue Options Record Type" " for more details.
message_properties	A record containing the message properties, defined using the specified record type. See Section 5.3.6, "Message Properties Record Type" " for more details.

Name	Description
payload	The data or payload that is placed on the queue. This is either an object (an instance of an object type), a RAW value, or NULL. The payload must match the specification in the associated queue table. This value is not interpreted by Oracle AQ, so any object type can be passed into the procedure.
msgid	This is the ID number of the message generated by AQ. It is a globally unique identifier that can be used to identify the message at dequeue time. In other words, you can specifically request that the message with this message ID be dequeued next.

5.4.1.2 Examples

The Section 5.7 section at the end of this chapter offers many different illustrations of using DBMS_AQ.ENQUEUE to send messages through a queue. In this section, I offer some initial examples to get you familiar with the kind of code you would write when enqueuing messages. In all these cases, assume that I have defined an object type as follows:

```
CREATE TYPE message_type AS OBJECT
   (title VARCHAR2(30),
    text VARCHAR2(2000));
```

My AQ administrator created a queue table and a message queue as follows:

```
EXEC DBMS_AQADM.CREATE_QUEUE_TABLE
   (queue_table => 'msg',
    queue_payload_type => 'message_type');

EXEC DBMS_AQADM.CREATE_QUEUE
   (queue_name => 'msgqueue',
    queue_table => 'msg');

EXEC DBMS_AQADM.START_QUEUE (queue_name => 'msgqueue');
```

So now I can enqueue a message to this queue as follows:

```
/* Filename on companion disk:

aqenq1.sql */
DECLARE
   queueopts DBMS_AQ.ENQUEUE_OPTIONS_T;
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid RAW(16);
   my_msg message_type;
BEGIN
   my_msg := message_type ('First Enqueue', 'May there be many
more...');

   DBMS_AQ.ENQUEUE ('msgqueue',
      queueopts,
      msgprops,
      my_msg,
      msgid);
END;
/
```

This is the simplest usage possible of DBMS_AQ.ENQUEUE. I declare my two record structures, because I *must* pass them in as arguments. I do not, however, modify any of the values in the fields; all have the default values documented in the Section 5.3, "Oracle AQ Nonprogram Elements" " section for the message properties record type.

As you can see, the message ID is a RAW of length 16. Rather than hard-coding that

declaration again and again in your application, I suggest that you instead declare a subtype that hides that information. My aq package (*aq.spp*), for example, offers this predefined type:

```
v_msgid RAW(16);
SUBTYPE msgid_type IS v_msgid%TYPE;
```

So when you want to declare a variable to store AQ message IDs, you can do the following:

```
DECLARE
   mymsgid aq.msgid_type;
```

Are you curious about those message IDs? When I ran the previous script, I also asked PL/SQL to display the msgid value with the following statement,

```
DBMS_OUTPUT.PUT_LINE (RAWTOHEX (msgid));
```

and this is what I saw:

```
E2CEA14B51F411D1B977E59CD6EE8E46
```

That is certainly a mouthful. When would you use this ID? Well, I can go right back to the data dictionary and ask to see all the information about this message. Every time you create a queue table, Oracle AQ creates an underlying database view with the name aq$<queue_table_name>. So if I created a queue table named "msg," I should be able to examine the contents of a table called aq$msg in my own schema.

I put together a little SQL*Plus script to show the status and user data for a message:

```
/* Filename on companion disk:

aqshomsg.sql */*
SELECT msg_state, user_data
  FROM aq$&1
 WHERE msg_id = HEXTORAW ('&2');
```

I can then call this script in SQL*Plus to show me the information about the just-queued message as follows:

```
SQL> @aqshomsg msg E2CEA14B51F411D1B977E59CD6EE8E46

MSG_STATE USER_DATA(TITLE, TEXT)
---------
------------------------------------------------------------------
READY     MESSAGE_TYPE('First Enqueue', 'May there be many
more...')
```

Notice that the query automatically detected the fact that my user_data is in fact an object, and showed me the full contents of that object quite neatly. You can, of course, also see other attributes of the message; see the later section entitled Section 5.6 for more details on this table and how best to retrieve information from it.

Of course, you will sometimes want to modify the message properties or enqueue options before performing your enqueue. To do this, simply change the values of the fields in the record. The following example shows how you can delay the availability of a message for dequeuing by three days and also request that one message be dequeued before another:

```
/* Filename on companion disk:

aqenq2.sql */*
DECLARE
   queueopts DBMS_AQ.ENQUEUE_OPTIONS_T;
```

```
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid1 aq.msgid_type;
   msgid2 aq.msgid_type;
   my_msg message_type;
BEGIN
   my_msg := message_type ('First Enqueue', 'May there be many
more...');

   /* Delay first message by three days, but otherwise rely on
defaults. */

   msgprops.delay := 3 * 60 * 60  * 24;

   DBMS_AQ.ENQUEUE ('msgqueue', queueopts, msgprops, my_msg,
msgid1);

   /* Now use the same properties record, but modify the enqueue
options
      to deviate from the normal sequence. */

   my_msg := message_type ('Second Enqueue', 'And this one goes
first...');

   queueopts.sequence_deviation := DBMS_AQ.BEFORE;
   queueopts.relative_msgid := msgid1;



DBMS_AQ.ENQUEUE ('msgqueue', queueopts, msgprops, my_msg, msgid2);


END;
/
```

5.4.2 Dequeuing Messages

Once you have placed a message on a queue, you need to extract that message from the queue. This is done with the DEQUEUE procedure.

5.4.2.1 The DBMS_AQ.DEQUEUE procedure

Use the DEQUEUE procedure to extract a message from a particular queue. Here's the header for this procedure:

```
PROCEDURE DBMS_AQ.DEQUEUE
   (queue_name IN VARCHAR2,
    dequeue_options IN DBMS_AQ.DEQUEUE_OPTIONS_T,
    message_properties OUT DBMS_AQ.MESSAGE_PROPERTIES_T,
    payload OUT <type_name>,
    msgid OUT RAW)
```

Parameters are summarized in the following table.

Name	Description
queue_name	Name of the queue from which the message should be dequeued. The queue cannot be an exception queue and must have been previously defined by a call to DBMS_AQADM.CREATE_QUEUE.
dequeue_options	Record containing the dequeuing options, defined using the specified record type. See Section 5.3.8, "Dequeue Options Record Type" " for more details.
message_properties	Record containing the message properties supplied when the message was enqueued. See Section 5.3.6 " for more details.

Name	Description
payload	Data or "payload" that is associated with this message on the queue. This is an object (an instance of an object type), a RAW value, or NULL. The payload must match the specification in the associated queue table.
msgid	ID number of the message generated by AQ.

5.4.2.2 Examples

The Section 5.7 " section at the end of this chapter offers many different illustrations of using DBMS_AQ.DEQUEUE to retrieve messages from a queue. In the remainder of this section, though, I offer some examples to get you familiar with the kind of code you would write when dequeuing messages. In all of these cases, assume that I have defined an object type as follows:

```
CREATE TYPE message_type AS OBJECT
   (title VARCHAR2(30),
    text VARCHAR2(2000));
```

Assume further that my AQ administrator has created a queue table and a message queue as follows:

```
EXEC DBMS_AQADM.CREATE_QUEUE_TABLE
   (queue_table => 'msg',
    queue_payload_type => 'message_type');

EXEC DBMS_AQADM.CREATE_QUEUE
   (queue_name => 'msgqueue',
    queue_table => 'msg');

EXEC DBMS_AQADM.START_QUEUE (queue_name => 'msgqueue');
```

Now I can dequeue a message that has previously been placed in this queue as follows:

```
/* Filename on companion disk:

aqdeq1.sql */*
DECLARE
   queueopts DBMS_AQ.DEQUEUE_OPTIONS_T;
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid aq.msgid_type; /* defined in aq.spp */
   my_msg message_type;
BEGIN
   DBMS_AQ.DEQUEUE ('msgqueue',
      queueopts,
      msgprops,
      my_msg,
      msgid);

   /* Now display some of the information. */
   DBMS_OUTPUT.PUT_LINE ('Dequeued message id is ' || RAWTOHEX
(msgid));
   DBMS_OUTPUT.PUT_LINE ('Dequeued title is ' || my_msg.title);
   DBMS_OUTPUT.PUT_LINE ('Dequeued text is ' || my_msg.text);
END;
/
```

Here is an example of output from this script:

```
SQL> @aqdeq1
Dequeued message id is E2CEA14C51F411D1B977E59CD6EE8E46
Dequeued title is First Enqueue
Dequeued text is May there be many more...
```

This is the simplest possible usage of DBMS_AQ.DEQUEUE. I declare my two record structures, because I *must* pass them in as arguments. However, I do not modify any of the values in the fields; all have the default values documented in the Section 5.3 " section for the message properties record type.

You can also modify the dequeue properties to change the behavior of the dequeue operation. The full set of options is explained in the Section 5.3.8 " section under Section 5.3 ." The following script demonstrates how you can request that messages not be removed from the queue after they are dequeued. You would do this when you want to search through a queue for a specific message, leaving all the others in place.

The following script dequeues a message once in BROWSE mode, then dequeues the same message in the default REMOVE mode, and then dequeues with REMOVE again:

```
/* Filename on companion disk:

aqdeq2.sql */*
DECLARE
   queueopts DBMS_AQ.DEQUEUE_OPTIONS_T;
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid aq.msgid_type; /* defined in aq.spp */
   my_msg message_type;

   /* A nested procedure to minimize code redundancy! */
   PROCEDURE getmsg (mode_in IN INTEGER)
   IS
   BEGIN
      queueopts.dequeue_mode := mode_in;

      DBMS_AQ.DEQUEUE ('msgqueue', queueopts, msgprops, my_msg,
msgid);

      /* Now display some of the information. */
      DBMS_OUTPUT.PUT_LINE ('Dequeued msg id is ' || RAWTOHEX
(msgid));
      DBMS_OUTPUT.PUT_LINE ('Dequeued title is ' || my_msg.title);
   END;
BEGIN
   /* Request browse, not remove, for dequeue operation. */
   getmsg (DBMS_AQ.BROWSE);

   /* Do the same thing again, this time with remove. You will
dequeue
      the same entry as before. */
   getmsg (DBMS_AQ.REMOVE);

   /* Dequeue a third time, again with remove, and notice the
different
      message ID. The previous message was, in fact, removed. */
   getmsg (DBMS_AQ.REMOVE);
END;
/
```

Here is the output from running the *aqdeq2.sql* script:

```
SQL> @aqdeq2
Dequeued msg id is E2CEA15251F411D1B977E59CD6EE8E46
Dequeued title is TWO EGGS OVER MEDIUM
Dequeued msg id is E2CEA15251F411D1B977E59CD6EE8E46
Dequeued title is TWO EGGS OVER MEDIUM
Dequeued msg id is E2CEA15351F411D1B977E59CD6EE8E46
Dequeued title is TWO EGGS OVER EASY
```

5.4.2.3 Dequeue search criteria

When you request a dequeue operation, you can specify search criteria. These criteria are used to determine which message is dequeued. The search criteria are established by the following fields in the dequeue options record: consumer_name, msgid, and correlation.

- If you specify a message ID in the dequeue options record, the message with that ID will be dequeued, regardless of its place in the queue.
- If you specify a correlation value, only those messages that have a correlation value matching the one you specify will be candidates for dequeuing. A match can be specified as an exact match or a pattern match.
- If you provide a value for the consumer_name field, only those messages that were enqueued for that consumer (a subscriber either to the queue as a whole or specified in the recipient list at enqueue time) are considered for dequeuing.

The Section 5.7 " section shows you the kind of code you need to write to support these different kinds of search criteria.

5.4.2.4 Dequeue order

The order in which messages are generally dequeued is determined by the characteristics of the queue table, established at the time of creation. For example, you can define a queue table in which the messages are ordered by the priority associated with the message.

You can override the default order by specifying the message ID or a correlation value in the dequeue options record. Remember that a message must be in the READY state to be dequeued -- unless you specify the message ID explicitly. When you use that ID, you override any other search criteria and restrictions.

Here is an example of dequeuing a message from the msg queue by specifying a message ID number (passed in as an argument to the procedure):

```
/* Filename on companion disk:

aqdeq3.sql */*
CREATE OR REPLACE PROCEDURE getmsg (msgid_in IN RAW)
IS
   queueopts DBMS_AQ.DEQUEUE_OPTIONS_T;
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid aq.msgid_type; /* defined in aq.spp */
   my_msg message_type;
BEGIN
   queueopts.msgid := msgid_in;

   DBMS_AQ.DEQUEUE ('msgqueue',
      queueopts, msgprops, my_msg, msgid);

   /* Now display some of the information. */
   DBMS_OUTPUT.PUT_LINE
      ('Requested message id is ' || RAWTOHEX (msgid_in));
   DBMS_OUTPUT.PUT_LINE
      ('Dequeued message id is  ' || RAWTOHEX (msgid));
   DBMS_OUTPUT.PUT_LINE ('Dequeued title is ' || my_msg.title);
END;
/
```

Here is an example of using this procedure:

```
/* Filename on companion disk:

aqdeq4.sql */*
DECLARE
   enqueue_opts DBMS_AQ.ENQUEUE_OPTIONS_T;
   dequeue_opts DBMS_AQ.DEQUEUE_OPTIONS_T;
   msgprops DBMS_AQ.MESSAGE_PROPERTIES_T;
   msgid1 aq.msgid_type;
   msgid2 aq.msgid_type;
   my_msg message_type;
BEGIN
   /* Enqueue two messages */
```

```
   my_msg := message_type
      ('Joy of Cooking', 'Classic Recipes for Oral Delight');
   DBMS_AQ.ENQUEUE ('msgqueue', enqueue_opts, msgprops, my_msg,
msgid1);

   my_msg := message_type ('Joy of Sex', 'Classic Recipes for
Delight');
   DBMS_AQ.ENQUEUE ('msgqueue', enqueue_opts, msgprops, my_msg,
msgid2);

   /* Now dequeue the first by its message ID explicitly. */
   getmsg (msgid1);
END;
/
```

And the results from executing *aqdeq4.sql* :

```
SQL> @aqdeq4
Requested message id is E2CEA16351F411D1B977E59CD6EE8E46
Dequeued message id is  E2CEA16351F411D1B977E59CD6EE8E46
Dequeued title is Joy of Cooking
```

5.4.2.5 Dequeue navigation

You specify the navigation method by setting a value in the navigation field of the dequeue options record.

The default navigation through a queue for dequeuing is "next message." This means that each subsequent dequeue retrieves messages from the queue based on the snapshot or view of the queue as it appeared when the first dequeue was performed. This approach offers a read consistency model similar to that of the Oracle RDBMS. For example, if you enqueue a message after you have issued a dequeue command, that message will not be dequeued (or even checked for availability for dequeuing) until all messages already in the queue have been processed.

Specifying DBMS_AQ. NEXT_MESSAGE for message navigation (the default) is often sufficient for dequeue operations. If either of the following conditions occur, however, you may need to change the navigation:

- You enqueue messages after dequeues have occurred, and you want those newly enqueued messages to be immediately considered for a dequeue.
- You have a nondefault (i.e., priority-based) ordering in your queue. In this case, if a higher-priority message enters the queue at any time, you want to consider that message for dequeuing immediately.

In either of these situations you should set the navigation method as follows:

```
DECLARE
   queueopts DBMS_AQ.DEQUEUE_OPTIONS_T;
BEGIN
   queueopts.navigation := DBMS_AQ.

FIRST_MESSAGE;
```

When you use "first message" navigation, you tell Oracle AQ that you want it to consider the entire set of messages in the queue for every dequeue command. Internally, this means that AQ will create a new "snapshot" view of the queue whenever you request a dequeue.

You will find many illustrations of the use of "first message" navigation in the Section 5.7 " section.

5.4.2.6 Dequeuing with message grouping

When you create a queue table specifying DBMS_AQADM.TRANSACTIONAL for the message_ grouping argument, any messages enqueued in the same transaction are considered part of a *group* of messages. This group may consist of only one message; there is also no upper limit on the number of messages allowed in a group.

When you work with queues that are not enabled for message grouping, a dequeue operation that specified LOCKED or REMOVE mode will only affect a single message. In a message grouping queue, on the other hand, a dequeue operation on one message in a group will lock the entire group. This allows you to treat all the messages in a group as a single unit or transaction.

When you are dequeuing against a message group, keep the following rules in mind:

- When you dequeue the last message in the current message group, Oracle AQ will raise the ORA-25235 exception.
- When you dequeue the last message in a message group, you must specify NEXT_TRANSACTION navigation in a dequeue options record in order to start dequeuing messages from the next available group.
- If there aren't any more message groups available in the queue, then the dequeue operation will time out after the period of time specified in the WAIT field of the dequeue options record.

PREVIOUS
5.3 Oracle AQ Nonprogram Elements

HOME
BOOK INDEX

NEXT
5.5 DBMS_AQADM: Performing AQ Administrative Tasks (Oracle8 only)

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