5.3 Oracle AQ Nonprogram Elements

Oracle AQ defines a number of data structures, exceptions, and other nonprogram elements you'll use when creating and manipulating queues. In addition, there are several data structures you will create and pass to Oracle AQ programs. In many cases, you will find yourself creating and manipulating objects, index-by tables (formerly known as PL/SQL tables), and records. If you are not familiar with these programming constructs, you should review the appropriate chapters in the second edition of *Oracle PL/SQL Programming* .

5.3.1 Constants

Oracle AQ predefines a set of constants that you then use in various calls to procedures and functions. The following two lists break out these constants into those that are used for administrative tasks and those that figure into operational actions. In both cases, I intentionally do not show the values assigned to these constants. You should never hard-code those values into your code. Instead, always rely on the constants.

5.3.1.1 Administrative tasks

When you are performing administrative tasks in AQ (such as creating queue tables and queues), you may need to use one of the following constants:

Task	Constant
Specify the type of payload	DBMS_AQADM.OBJECT_TYPE_PAYLOAD DBMS_AQADM.RAW_TYPE
Enable or disable a queue for multiple consumers	DBMS_AQADM.SINGLE DBMS_AQADM.MULTIPLE
Request that messages on a queue never expire	DBMS_AQADM.INFINITE
Specify type of message grouping for a queue table	DBMS_AQADM.TRANSACTIONAL DBMS_AQADM.NONE
Specify type of queue	DBMS_AQADM.NORMAL_QUEUE DBMS_AQADM.EXCEPTION_QUEUE

5.3.1.2 Operational tasks

When you are enqueuing and dequeuing messages (the operational tasks in AQ), you may need to use one of the following constants:

Description	Constant
Specify visibility of the queue message	DBMS_AQ.IMMEDIATE DBMS_AQ.ON_COMMIT
Specify dequeuing mode	DBMS_AQ.BROWSE DBMS_AQ.LOCKED DBMS_AQ.REMOVE
Specify method of inter-message navigation	DBMS_AQ.FIRST_MESSAGE DBMS_AQ.NEXT_MESSAGE DBMS_AQ.NEXT_TRANSACTION
Specify state of the message	DBMS_AQ.WAITING DBMS_AQ.READY DBMS_AQ.PROCESSED DBMS_AQ.EXPIRED
Specify deviation from normal dequeuing sequence	DBMS_AQ.BEFORE DBMS_AQ.TOP
Specify amount of time to wait for a dequeue operation to succeed	DBMS_AQ.FOREVER DBMS_AQ.NO_WAIT
Specify amount of time to delay before making a message available for dequeuing	DBMS_AQ.NO_DELAY
Specify amount of time to elapse before a message expires	DBMS_AQ.NEVER

5.3.2 Object Names

You will specify the name of an Oracle AQ object (queue, queue table, or object type) in many different program calls. An AQ object name can be up to 24 characters long, and can be qualified by an optional schema name (also a maximum of 24 characters in length). If you do not specify a schema, then the current schema is used.

In the following block I create a RAW queue table for use with my own schema:

```
DECLARE
   v_queuetable VARCHAR2(24) := 'myqueue';
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE (
      queue_table => v_queuetable,
      queue_payload_type => 'RAW');
```

But in the next call to the same built-in procedure, I create a queue table in another

schema:

```
DECLARE
   v_queuetable VARCHAR2(49) := 'scott.myqueue';
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE (
      queue_table => v_queuetable,
      queue_payload_type => 'RAW');
```

I specified 49 characters, since I needed room (potentially) for the period.

Now you know the rules for object names. However, you should never hard-code those rules into your programs as shown in the previous examples. What if Oracle decides to increase the allowable size for these names? Your programs will be stuck using the old limitations. Instead, you should define subtypes that you can then use to declare queue-related variables without any hard-coded restraints.

My aq package (*aq.spp*) demonstrates this technique. Here are the first few lines of that package's specification:

```
/* Filename on companion disk:

aq.spp */*
CREATE OR REPLACE PACKAGE aq
IS
   v_msgid RAW(16);
   SUBTYPE msgid_type IS v_msgid%TYPE;

   v_name VARCHAR2(49);
   SUBTYPE name_type IS v_name%TYPE;
```

With the aq package defined in my schema, I would set up my raw queue table as follows:

```
DECLARE
   v_queuetable aq.name_type := 'myqueue';
BEGIN
   DBMS_AQADM.CREATE_QUEUE_TABLE (
      queue_table => v_queuetable,
      queue_payload_type => 'RAW');
```

No more literal value in the datatype of my declaration!

5.3.3 Queue Type Names

When you specify the name of a queue type (also referred to as "payload type"), you provide either the name of an object type (previously defined in the database) or you specify the constant "RAW" (as shown in the previous section).

If you specify a payload type of RAW, AQ creates a queue table with a LOB column as the repository for any messages in its queues. The LOB value is limited to a maximum of 32K bytes of data. In addition, since LOB columns are used, the AQ administrator can specify the LOB tablespace and configure the LOB storage by providing a storage string in the storage_clause parameter in the call to the DBMS_AQADM. CREATE_QUEUE_TABLE procedure.

5.3.4 Agents Object Type

An agent is an object that produces or consumes a message. You will create agents in

order to specify subscribers for queues and also to create recipient lists for the dissemination of messages. You define an agent as an instance of the following object type,

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

where the name is the name of the agent, and address is a character field of up to 1024 bytes that is interpreted according to the protocol value (of which the only supported value is currently 0). A protocol of 0 indicates that the address is to be interpreted as a database link. The address will therefore have this form: queue_name@dblink, where queue_name has the form [schema.]queue and dblink is either a fully qualified database link name or a database link name that does not incorporate the domain name.

Here is an example of defining an agent to be used with AQ:

```
DECLARE
   consumer_agent SYS.AQ$_AGENT;
BEGIN
   /* And now I use the constructor method to give
      a name to that object. */
   consumer_agent := SYS.AQ$_AGENT ('CLERK');
```

5.3.5 Recipient and Subscriber List Table Types

The subscriber and recipient lists are lists of agents, each of which is an instance of one of the following two index-by table types:

```
TYPE DBMS_AQ.AQ$_RECIPIENT_LIST_T IS TABLE OF SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;

TYPE DBMS_AQADM.AQ$_SUBSCRIBER_LIST_T IS TABLE OF
SYS.AQ$_AGENT
   INDEX BY BINARY_INTEGER;
```

The recipient list is used when enqueuing a message to establish a specific list of agents who can dequeue or consume a message. It is therefore defined in the DBMS_AQ package.

The subscriber list is used to enqueue a message to a list of subscribers for a given queue. You will call the DBMS_AQADM.QUEUE_SUBSCRIBERS function to obtain the subscript list for a queue. The subscripter list table type is therefore defined in the DBMS_AQADM package. As you can see, these table types are identical in structure; only their names differ.

The following block of code demonstrates the creation of a recipient list:

```
DECLARE
   recipients DBMS_AQ.AQ$_RECIPIENT _LIST_T;
BEGIN
   recipients(1) := SYS.AQ$_AGENT ('DBA');
   recipients(2) := SYS.AQ$_AGENT ('DESIGNER');
   recipients(3) := SYS.AQ$_AGENT ('DEVELOPER');
```

See the Section 5.7 " section entitled Section 5.7.8, "Working with Multiple Consumers" " for a complete example of the creation of recipient lists and the association of those lists with a queued message.

5.3.6 Message Properties Record Type

When you enqueue a message, you can associate a set of properties with that message. You can then also receive these properties (or most of them) when you dequeue the message. You define the properties of a message by declaring and populating a PL/SQL record based on the following record type:

```
TYPE DBMS_AQ.MESSAGE_PROPERTIES_T IS RECORD
   (priority        BINARY_INTEGER DEFAULT 1,
    delay           BINARY_INTEGER DEFAULT DBMS_AQ.NO_DELAY,
    expiration      BINARY_INTEGER DEFAULT DBMS_AQ.NEVER,
    correlation     VARCHAR2(128) DEFAULT NULL,
    attempts        BINARY_INTEGER,
    recipient_list  DBMS_AQ.AQ$_RECIPIENT_LIST_T,
    exception_queue VARCHAR2(51) DEFAULT NULL,
    enqueue_time    DATE,
    state           BINARY_INTEGER);
```

Here is an explanation of the various fields of this record type:

priority

Specifies the priority of the message you are queueing. A smaller number indicates a higher priority. The priority can be any number, including negative numbers. The default is 1.

delay

Specifies the delay of the enqueued message. This value indicates the number of seconds after which a message becomes available for dequeuing. If you specify DBMS_AQ.NO_DELAY (the default), then the message is available for immediate dequeueing. A message enqueued with a delay set will be placed in the WAITING state. When the delay time expires, the message changes to the READY state. Delay processing requires that the Queue Monitor be started.

NOTE: Dequeuing by the message ID overrides the delay specification. In addition, the delay is set by the producer, who enqueues the message, not the consumer, who dequeues the message.

expiration

Specifies the time in seconds after which the message expires. This value determines the number of seconds a message will remain in the READY state, available for dequeuing. If you specify DBMS_AQ.NEVER, then the message will never expire (the default behavior). If the message is not dequeued before it expires, it will be moved to the exception queue in the EXPIRED state.

This parameter is an offset from the delay value specified (see earlier). Expiration processing requires that the Queue Monitor be running.

correlation

Specifies identification supplied by the producer for a message at enqueuing. This is a free-form text field. Place whatever value you would like to use to later identify this message for dequeuing.

attempts

Specifies the number of attempts that have been made to dequeue this message. This parameter cannot be set at enqueue time. Instead, it is maintained automatically by AQ and is available when you have dequeued the message.

recipient_list

A table containing a list of agents. This parameter is valid only for queues that allow multiple consumers. If you do not specify a recipient list, then the default recipients of this message are the agents identified as subscribers to the queue (with a call to DBMS_AQADM.ADD_SUBSCRIBER). This parameter is not returned to a consumer at dequeue time.

exception_queue

Specifies the name of the queue to which the message is moved if it cannot be processed successfully. You specify this value at enqueue time.

Messages are moved in two cases: the number of unsuccessful dequeue attempts has exceeded the maximum number of retries, or the message has expired. All messages in the exception queue are set to the EXPIRED state. If you do not specify an exception queue, the exception queue associated with the queue table is used. If the exception queue specified does not exist at the time of the move, the message will be moved to the default exception queue associated with the queue table. A warning will then be logged in the Oracle alert file. If the default exception queue is used, the parameter will return a NULL value at dequeue time.

You will find an example of using a non-default exception queue in the Section 5.7 " section entitled Section 5.7.6, "Using Time Delay and Expiration" ."

enqueue_time

Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user. This parameter cannot be set at enqueue time. It is available only when the message is dequeued.

state

Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. Instead, this state is maintained automatically by AQ and can be one of the following values:

DBMS_AQ.WAITING

The message delay has not yet been reached (value = 1).

DBMS_AQ.READY

The message is ready to be processed (value = 0).

DBMS_AQ.PROCESSED

The message has been processed and is retained (value = 3).

DBMS_AQ.EXPIRED

The message has been moved to the exception queue (value = 4).

The following block of code demonstrates how to define a message properties record and set several of the fields:

```
DECLARE
   msgprop DBMS_AQ.MESSAGE_PROPERTIES_T;
BEGIN
   msgprop.priority := -100; /* high priority */
   msgprop.delay := 60*60*24 /* delay for one day */
   msgprop.expiration := 60*60; /* expire one hour after
delay */
```

5.3.7 Enqueue Options Record Type

When you enqueue a message, you can specify the options you want associated with that message. You do this by declaring and populating a record based on the following record type:

```
TYPE DBMS_AQ.ENQUEUE_OPTIONS_T IS RECORD
   (visibility          BINARY_INTEGER DEFAULT
DBMS_AQ.ON_COMMIT,
    relative_msgid      RAW(16) DEFAULT NULL,
    sequence_deviation  BINARY_INTEGER DEFAULT NULL);
```

Fields have the following meanings:

visibility

> Specifies the transactional behavior of the enqueue request. There are two possible values:

DBMS_AQ.ON_COMMIT

> The enqueue is treated as part of the current transaction. The enqueue operation completes only when the transaction commits. This is the default case.

DBMS_AQ.IMMEDIATE

> The enqueue is not treated as part of the current transaction. Instead, the enqueue operation acts as its own transaction. The queued message is then immediately available for dequeuing by other Oracle sessions.

relative_msgid

> Specifies the message identifier of the message referenced in the sequence deviation operation. This field is valid if, and only if, BEFORE is specified in the sequence_deviation field (see the next field description). This parameter will be ignored if sequence deviation is not specified (i.e., if the default of NULL is used for the sequence_deviation field).

sequence_deviation

> Specifies whether the message being enqueued should be dequeued before other message(s) already in the queue. There are three valid options:

DBMS_AQ.BEFORE

> The message is enqueued ahead of the message specified by relative_msgid.

DBMS_AQ.TOP

The message is enqueued ahead of any other messages.

NULL

The default value, specifying that there is no deviation from the normal sequence for dequeueing.

The following block of code sets up the enqueue properties such that the queued message goes to the top of the queue and is made immediately visible to other sessions:

```
DECLARE
   queueopts DBMS_AQ.ENQUEUE_OPTIONS_T;
BEGIN
   queueopts.visibility := DBMS_AQ.IMMEDIATE;
   queueopts.sequence_deviation := DBMS_AQ.TOP;
```

5.3.8 Dequeue Options Record Type

When you dequeue a message, you can specify the options you want associated with that message by declaring and populating a record based on the following record type:

```
TYPE DBMS_AQ.DEQUEUE_OPTIONS_T IS RECORD
   (consumer_name  VARCHAR2(30) DEFAULT NULL,
    dequeue_mode   BINARY_INTEGER DEFAULT DBMS_AQ.REMOVE,
    navigation     BINARY_INTEGER DEFAULT
DBMS_AQ.NEXT_MESSAGE,
    visibility     BINARY_INTEGER DEFAULT DBMS_AQ.ON_COMMIT,
    wait           BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER
    msgid          RAW(16) DEFAULT NULL,
    correlation    VARCHAR2(128) DEFAULT NULL);
```

Fields have the following meanings:

consumer_name

Specifies the name of the consumer of this message. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers (either subscribers to the queue as a whole or the recipient list specified at the time of queuing), this field should be set to NULL (the default).

dequeue_mode

Specifies the locking behavior associated with the dequeue operation. These are the valid options:

DBMS_AQ.BROWSE

Read the message without acquiring any lock on the message. This is equivalent to a query: "readers never block writers or readers."

DBMS_AQ.LOCKED

Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This is equivalent to a SELECT FOR UPDATE statement.

DBMS_AQ.REMOVE

Read the message and update or delete it. This is the default behavior. When you read from the queue, the message is removed from the queue. Note that the

message can be retained in the queue table based on its retention properties.

navigation

Specifies the position of the message that will be retrieved next. When you perform a dequeue, the following steps are taken: (a) the position in the queue is determined; (b) the search criterion specified by this and other fields is applied; and (c) the appropriate message is retrieved. These are the valid options for this field:

DBMS_AQ.NEXT_MESSAGE

Retrieve the next message that is available and matches all the search criteria. If the previous message belongs to a message group, AQ will retrieve the next available message that matches the search criteria and belongs to the message group. This is the default behavior.

DBMS_AQ.NEXT_TRANSACTION

Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This option can be used only if message grouping is enabled for the current queue.

DBMS_AQ.FIRST_MESSAGE

Retrieve the first message that is available and matches the search criteria. This will reset the current position to the beginning of the queue.

visibility

Specifies whether the new message is dequeued as part of the current transaction. This parameter is ignored when you have specified the BROWSE mode to read the queue. The following options are valid:

DBMS_AQ.ON_COMMIT

The dequeue is treated as part of the current transaction. The dequeue operation completes only when the transaction commits. This is the default case.

DBMS_AQ.IMMEDIATE

The dequeue is not treated as part of the current transaction. Instead, the dequeue operation acts as its own transaction. The queued message is then immediately available for dequeuing by other Oracle sessions.

wait

Specifies the number of seconds to wait if there is currently no message available matching the search criteria. If the queue table for this queue specified message grouping, then this value is applied only after the last message in a group has been dequeued. You can specify a number of seconds or one of the following named constants:

DBMS_AQ.FOREVER

Wait forever. This is the default.

DBMS_AQ.NO_WAIT

Do not wait at all. If there is no matching message, then return to the calling program immediately.

msgid

Specifies the message identifier of the message to be dequeued. If you specify the message ID, then you can bypass other criteria establishing the next message for dequeuing.

correlation

Specifies the correlation identifier of the message to be dequeued. If you provided a correlation string when you enqueued this message, that string will be used as part of the criteria to establish the next message. You can perform pattern matching by including the percent sign (%) or the underscore (_) in your correlation identifier. These characters follow the standard SQL wildcard rules. If more than one message matches the pattern, the order of dequeuing is not determined.

5.3.9 Oracle AQ Exceptions

There are no named exceptions defined in either of the AQ packages. Instead, Oracle has set aside error messages for Oracle AQ in the following ranges:

-24099 through -24000
-25299 through -25200

Here are some of the more common exceptions you will encounter:

ORA-24010: QUEUE <queue> does not exist

You have tried to perform an operation on a queue that does not yet exist.

ORA-24001: cannot create QUEUE_TABLE, <queue_table> already exists

You have tried to create a queue table, but there is already one by that name.

ORA-24011: cannot drop QUEUE, <queue> should be stopped first

You have tried to drop a queue that has not been stopped.

ORA-24012: cannot drop QUEUE_TABLE, some queues in <queue_table> have not been dropped

You must stop and drop all queues in a queue table before the queue table itself can be dropped.

ORA-24034: application <agent_name> is already a subscriber for queue <queue>

You tried to add an agent to a subscriber list that is already present. Note that agent names are not case-sensitive.

ORA-25215: user_data type and queue type do not match

The object type specified in an enqueue operation does not match the object type used to define the queue table.

ORA-25228: timeout in dequeue from <queue> while waiting for a message

This error usually occurs when you try to dequeue a message from an empty queue.

ORA-25235: fetched all messages in current transaction

You have dequeued the last message in the current message group. You must now specify NEXT_TRANSACTION navigation in order to start dequeuing messages from the next available group.

ORA-25237: navigation option used out of sequence

The NEXT_MESSAGE or NEXT_TRANSACTION option was specified after dequeuing all the messages. You must reset the dequeuing position using the FIRST_MESSAGE navigation option and then specify the NEXT_MESSAGE or NEXT_TRANSACTION option.

PREVIOUS
5.2 Getting Started with Oracle AQ

HOME
BOOK INDEX

NEXT
5.4 DBMS_AQ: Interfacing to Oracle AQ (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