**What Is Messaging?**

Messaging is the ability to send a message containing data from one application/process to another application/process. It is a widely used technique in distributed systems, particularly high volume OLTP systems. Unlike client server applications, which are typically synchronous, messaging is typically asynchronous (i.e., the sender, referred to as the producer, is not blocked waiting for a reply from the recipient(s), referred to as consumer(s)). Oracle Advance Queuing (AQ) does not support synchronous messaging.

Messaging allows applications and systems to communicate and cooperate in an API independent manner.

Messages are placed into queues, called enqueuing. The enqueuing applications are called the producers. The application data portion of the message is referred to as the payload. Messages are read and removed from the queue by dequeuing the message. Applications dequeuing messages are referred to as consumers.

There are three general categories of messaging:

* Single consumer (i.e., point-to-point) — a message is dequeued by a single consumer
* Multicast — the producer effectively names designated consumers for the message
* Broadcast — consumers may dynamically gain access to a message queue by subscribing.

Features include:

* Error handling
* Timeouts and expirations
* Enqueuing/dequeuing of messages as a group
* Dequeueing messages by criteria other than FIFO, including, but not limited to: enqueue time, priority and contents of messages
* Reliability
* Propagation — pushing messages to destinations
* Other queues
* Other databases
* Other messaging systems (e.g., JMS, middleware, gateways)
* Retention of messages and history of actions
* Nonrepudiation
* Logging
* Performance evaluation
* Warehousing
* Wide range of message content data types (i.e., payload), including: text; XML; BLOB, LOB and CLOB; and structured records
* Notification to consumers of message availability
* Guaranteed delivery
* High performance

Oracle AQ also provides the ability to browse messages without dequeuing.

In a single consumer queue, a given message is dequeued by only one consumer, after which it is removed from the queue.

Queues need to be started after creation via START\_QUEUE. Queues can be stopped via STOP\_QUEUE. Both procedures allow control of enqueuing and dequeuing separately.

Message States  
  
A message may be in one of the following states:

* READY – message is available to be dequeued
* WAITING – availability for dequeuing is delayed
* EXPIRED – message has timed out and been moved to exception queue
* PROCESSED – message has been consumed by all consumers

An AQ table is an abstract object type that may be implemented by one or more underlying tables, indexes and index organized tables depending on whether the AQ table supports single or multiconsumer queues. An AQ table typically holds one or more queues, which can be created and destroyed dynamically.

DBMS\_AQADM.CREATE\_QUEUE\_TABLE(  
 queue\_table IN VARCHAR2,  
 queue\_payload\_type IN VARCHAR2,

sort\_list IN VARCHAR2 DEFAULT NULL,  
 multiple\_consumers IN BOOLEAN DEFAULT FALSE,  
 message\_grouping IN BINARY\_INTEGER DEFAULT NONE,  
 comment IN VARCHAR2 DEFAULT NULL,  
 primary\_instance IN BINARY\_INTEGER DEFAULT 0,  
 secondary\_instance IN BINARY\_INTEGER DEFAULT 0,  
 compatible IN VARCHAR2 DEFAULT NULL,  
 secure IN BOOLEAN DEFAULT FALSE);

/\* sort\_list is Order like FIFO \*/

Creating Queues

Queues are created via:

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,  
 auto\_commit IN BOOLEAN DEFAULT TRUE);

The following PL/SQL API is used to enqueue messages:

DBMS\_AQ.ENQUEUE(  
 queue\_name IN VARCHAR2,  
 enqueue\_options IN enqueue\_options\_t,  
 message\_properties IN message\_properties\_t,  
 payload IN “type\_name”,   
 msgid OUT RAW);

* queue\_name – the name of the queue in which the message is to be enqueued
* payload – the type definition of the payload, typically, but not limited to, a PL/SQL abstract type
* msg\_id – the unique identifier of the message

DBMS\_AQ.ENQUEUE\_OPTIONS\_T

The DBMS\_AQ.ENQUEUE\_OPTIONS\_T record contains the options for enqueuing the message as described below:

TYPE SYS.ENQUEUE\_OPTIONS\_T IS RECORD (  
 visibility BINARY\_INTEGER DEFAULT ON\_COMMIT,  
 relative\_msgid RAW(16) DEFAULT NULL,  
 sequence\_deviation BINARY\_INTEGER DEFAULT NULL,  
 transformation VARCHAR2(61) DEFAULT NULL,  
 delivery\_mode PLS\_INTEGER NOT NULL DEFAULT PERSISTENT);

The DBMS\_AQ.MESSAGE\_PROPERTIES\_T record is used for both enqueuing and dequeuing operations.

TYPE message\_properties\_t IS RECORD (  
 priority BINARY\_INTEGER NOT NULL DEFAULT 1,  
 delay BINARY\_INTEGER NOT NULL DEFAULT NO\_DELAY,  
 expiration BINARY\_INTEGER NOT NULL DEFAULT NEVER,  
 correlation VARCHAR2(128) DEFAULT NULL,  
 attempts BINARY\_INTEGER,  
 recipient\_list AQ$\_RECIPIENT\_LIST\_T,  
 exception\_queue VARCHAR2(61) DEFAULT NULL,  
 enqueue\_time DATE,  
 state BINARY\_INTEGER,  
 sender\_id SYS.AQ$\_AGENT DEFAULT NULL,  
 original\_msgid RAW(16) DEFAULT NULL,  
 signature aq$\_sig\_prop DEFAULT NULL,  
 transaction\_group VARCHAR2(30) DEFAULT NULL,  
 user\_property SYS.ANYDATA DEFAULT NULL  
 delivery\_mode PLS\_INTEGER NOT NULL DEFAULT DBMS\_AQ.PERSISTENT);

Dequeuing Features

Oracle AQ provides very high performance and functionality. Key features include:

* Concurrent dequeues
* Multiple dequeue methods and options
* Array dequeue
* Message navigation
* Waiting for messages
* Retries with delays
* Transaction protection
* Exception queues

The PL/SQL API is:

DBMS\_AQ.DEQUEUE(  
 queue\_name IN VARCHAR2,  
 dequeue\_options IN dequeue\_options\_t,   
 message\_properties OUT message\_properties\_t,  
 payload OUT “type\_name”,  
 msgid OUT RAW);

* Note that message\_properties\_t is used for both enqueue and dequeue operations.

TYPE DEQUEUE\_OPTIONS\_T IS RECORD (  
 consumer\_name VARCHAR2(30) DEFAULT NULL,  
 dequeue\_mode BINARY\_INTEGER DEFAULT REMOVE,  
 navigation BINARY\_INTEGER DEFAULT NEXT\_MESSAGE,  
 visibility BINARY\_INTEGER DEFAULT ON\_COMMIT,   
 wait BINARY\_INTEGER DEFAULT FOREVER,  
 msgid RAW(16) DEFAULT NULL,  
 correlation VARCHAR2(128) DEFAULT NULL,  
 deq\_condition VARCHAR2(4000) DEFAULT NULL,  
 signature aq$\_sig\_prop DEFAULT NULL,  
 transformation VARCHAR2(61) DEFAULT NULL,  
 delivery\_mode PLS\_INTEGER DEFAULT PERSISTENT);

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);

JMS vs RestAPI Queues

Diagram

Description automatically generated

Two Factor Authentication

You need two things, something you know and something you have.

1. A valid user and password combination
2. A second authentication method, a device something you have like a authentication device.

**CREATE** USER aquser IDENTIFIED **BY** aquser

/

**GRANT** **CONNECT**, RESOURCE, AQ\_ADMINISTRATOR\_ROLE **TO** aquser

/

**GRANT** **EXECUTE** **ON** dbms\_aq **TO** aquser

/

**GRANT** **EXECUTE** **ON** dbms\_aqadm **to** aquser

/

https://oraclefrontovik.com/2013/02/03/queuing-for-oracle-apex-part-1/

https://oraclefrontovik.com/2013/02/26/queuing-for-oracle-apex-part-2/

I now log in to the database as aquser and create a schema level object type. This object will be referenced when the queue is created and will be used to store the feedback as a message on the queue.

|  |
| --- |
| CREATE TYPE q\_message AS OBJECT  (  id NUMBER(10),  title VARCHAR2(100),  details VARCHAR2(4000),  email\_address VARCHAR2(1000)  )  / |

With the object type created, the next step is to create the queue table. The queue table is used to store the messages until they are dequeued. It is created via the api dbms\_aqadm.create\_queue\_table. The parameters show the name of the queue table and the reference to the object that was created in the previous step.

|  |
| --- |
| **BEGIN**     dbms\_aqadm.create\_queue\_table     (        queue\_table        => 'feedback\_queue\_table',        queue\_payload\_type => 'q\_message',        comment            => 'Creating feedback queue table'     );  **END**;  / |

With the queue table in place, the queue can now be created. This is achieved by running:

|  |
| --- |
| **BEGIN**        dbms\_aqadm.create\_queue      (         queue\_name  => 'feedback\_queue',         queue\_table => 'feedback\_queue\_table',         comment     => 'feedback queue'      );    **END**;  / |

Once created, the new feedback queue can be started by running:

|  |
| --- |
| **BEGIN**        dbms\_aqadm.start\_queue      (         queue\_name => 'feedback\_queue'      );    **END**;  / |

Now that the Feedback queue has been created and started, it is ready for messages to be enqueued. To do that I have created a feedback\_nq procedure which in essence is just a wrapper for the dbms\_aq.enqueue procedure. Feedback\_nq accepts one parameter, which is of the same type used in creating the Feedback queue:

|  |
| --- |
| **CREATE** OR REPLACE **PROCEDURE** feedback\_nq  (      p\_feedback IN q\_message  )  **IS**        lkv\_queue\_name    CONSTANT VARCHAR2(60) := 'aquser.feedback\_queue';      lt\_eopt           dbms\_aq.enqueue\_options\_t;      lt\_mprop          dbms\_aq.message\_properties\_t;      lo\_the\_feedback   q\_message;      lt\_enq\_msgid      RAW(16);    **BEGIN**        dbms\_aq.enqueue      (          queue\_name         => lkv\_queue\_name,          enqueue\_options    => lt\_eopt,          message\_properties => lt\_mprop,          payload            => p\_feedback,          msgid              => lt\_enq\_msgid      );    **COMMIT**;    **END** feedback\_nq;  / |
|  |

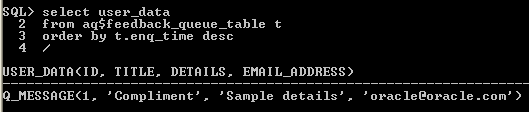
I can now run a pl/sql anonymous block to test this procedure. Before I run the test I check the the queue table is empty:

|  |
| --- |
| **select** t.user\_data  **from**   aq$feedback\_queue\_table t |

This query completes successfully but returns zero rows because there are now messages on the queue. So lets add a message:

|  |
| --- |
| **DECLARE**       lt\_feedback   aquser.q\_message;    **BEGIN**       lt\_feedback := aquser.q\_message                    (                       id            => 1,                       title         => 'Compliment',                       details       => 'Sample details',                       email\_address => 'oracle@oracle.com'                    );       feedback\_nq     (        p\_feedback => lt\_feedback     );    **END**; |

Once run, a message will be enqueued. This can be confirmed by querying the feedback queue table



Because the Application that will ultimately call the procedure is in another schema, grants are required on the feedback\_nq procedure and the object used to contain the message.

**GRANT** **EXECUTE** **ON** feedback\_nq **to**

**GRANT** **EXECUTE** **ON** q\_message **to**

All the objects required by the new user AQuser have now been created so we can now move on to creating a couple of supporting objects for our Apex interface.

Start a new database session, this time logging into the application schema create the following objects.

|  |
| --- |
| **CREATE** **SEQUENCE** feedback\_seq  /    **CREATE** OR REPLACE **PROCEDURE** add\_feedback  (      pv\_title         IN VARCHAR2,      pv\_details       IN VARCHAR2,      pv\_email\_address IN VARCHAR2  )  **IS**       lo\_feedback\_message   aquser.q\_message;    **BEGIN**      lo\_feedback\_message := aquser.q\_message                           (                              id            => feedback\_seq.nextval,                              title         => pv\_title,                              details       => pv\_details,                              email\_address => pv\_email\_address                           );       aquser.feedback\_nq     (        p\_feedback => lo\_feedback\_message     );    **END** add\_feedback; |

The procedure assigns the incoming parameters to the queue object type and once done it calls the feedback\_nq procedure.

The final step is to call this procedure from our Apex application.

I created a new Apex database application, added two HTML pages. On page one I added several form text items and a button. Under page processing I created a new process called add feedback which contains a call to the procedure and passes it the values of the page items.

add\_feedback

(

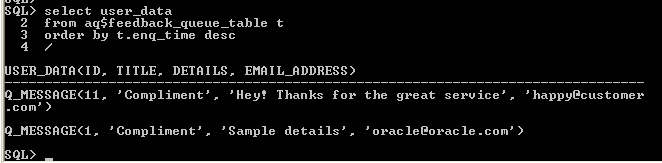
    pv\_title           => P1\_HEADING,

    pv\_details         => P1\_FEEDBACK,

    pv\_email\_address   => P1\_EMAIL

);

If we run our query against the feedback queue table you can see that the message from Apex is has been enqueued:



In [Part 2](https://oraclefrontovik.com/2013/02/26/queuing-for-oracle-apex-part-2/) I will show how to dequeue these messages and display them in an Apex report.

In [Part 1](https://oraclefrontovik.com/?p=368) of this article I described the steps required to first create a queue and then how to enqueue messages using server side PL/SQL called from an Apex application.

In part 2 I will describe the steps required to dequeue the messages and how they may be displayed using Apex.

The first step is to write the procedure that will dequeue the messages from the queue. This procedure should be created in the same schema as the Queue Owner and the execute privilege should be given to Apex user.

|  |
| --- |
| **CREATE** OR REPLACE **PROCEDURE** feedback\_dq  **IS**       lkv\_queue\_name        CONSTANT VARCHAR2(60) := 'aquser.feedback\_queue';     lt\_dq\_opt             dbms\_aq.dequeue\_options\_t;     lt\_mprop              dbms\_aq.message\_properties\_t;     lo\_the\_feedback       q\_message;     lt\_deq\_msgid          RAW(16);     lb\_further\_feedback   BOOLEAN := **TRUE**;       le\_no\_messages        EXCEPTION;     PRAGMA EXCEPTION\_INIT(le\_no\_messages, -25228);    **BEGIN**       lt\_dq\_opt.wait := DBMS\_AQ.NO\_WAIT;       WHILE lb\_further\_feedback     LOOP  **BEGIN**           dbms\_aq.dequeue           (              queue\_name         => lkv\_queue\_name,              dequeue\_options    => lt\_dq\_opt,              message\_properties => lt\_mprop,              payload            => lo\_the\_feedback,              msgid              => lt\_deq\_msgid           );    **INSERT** **INTO** feedback(id,                                title,                                details,                                email\_address,                                date\_time\_created)  **VALUES**(lo\_the\_feedback.ID,                  lo\_the\_feedback.title,                  lo\_the\_feedback.details,                  lo\_the\_feedback.email\_address,                  SYSDATE);    **COMMIT**;          EXCEPTION    **WHEN** le\_no\_messages **THEN**                lb\_further\_feedback := **FALSE**;    **END**;    **END** LOOP;    **END** feedback\_dq; |

The procedure used to dequeue the messages is straight forward so I will only explain the salient points.

The wait parameter of the dequeue option is set to NO\_WAIT (Line 16). This ensures that if there are no messages, control returns from dbms\_aq.dequeue.

dbms\_aq.dequeue is called (line 21) and if there is a message its contents are inserted into the feedback table.

When there are no more messages and because NO\_WAIT has been specified dbms\_aq.dequeue raises an ORA-25228. This is handled in the exception section which sets the boolean controlling the loop false and allows the procedure to complete.

With the dequeue procedure in place all that is left is to create an Apex page that will call the feedback\_dq procedure and display the contents in a report on the same page.

Here is the page I created. The “Get Feedback button” simply calls the procedure feedback\_dq and the report is based on the feedback table. So before any messages have been dequeued the page looks like this:

<https://blogs.oracle.com/developers/post/enterprise-messaging-via-oracle-advanced-queuing-with-autonomous-db-micronaut>

 a full-featured messaging solution right inside the database.

https://blogs.oracle.com/developers/post/send-and-receive-messages-via-rest-with-advanced-queuing-and-ords