

Oracle Advanced Queuing Overview

Robert Bond
Database Architecture Group
Qwest Communications, Inc.
July 16, 2001



Introduction

In contrast to how client/server applications usually execute user requests immediately, queuing implements deferred execution of work. Queuing enables applications to continue even with a temporary disruption of the network or some other failure.

Oracle advanced queuing combines persistent queuing, so that messages are never lost, with transaction protection, so that transactions are guaranteed to succeed/fail as a complete work unit. Oracle achieves the combination of persistent queuing with transaction protection through integration with the Oracle Server, without having to rely on a TP monitor or other message-oriented middleware.

Beginning with Oracle 8.0.4, the Oracle Server has included advanced queuing capabilities. This document is a basic overview of what Advanced Queuing is and what it offers. Additionally, a code example is provided beginning on page 11.

What is Oracle8 Advanced Queuing?

Oracle8 Advanced Queuing provides message queuing as an integrated part of the Oracle Server. Major features include reliable, asynchronous, distributed messaging plus transactional integrity.

Oracle8 uses the following components to deliver AQ functionality:

Message

A message is the smallest unit of information inserted into and retrieved from a queue. It contains user data (payload) in the form of an object type, and control information (metadata) that can be used to specify message ordering, a time window for execution, message priority and so on. A message can reside in only one queue. Messages are created using the DBMS_AQ.ENQUEUE PL/SQL procedure. Messages can later be read by the same or another application using the DBMS_AQ.DEQUEUE PL/SQL procedure. The application developer is responsible for the application's use of the ENQUEUE and DEQUEUE procedure calls.

Queue

A queue is a repository of messages. There are two types of queues: user queues and exception queues. A user queue is for normal message processing. Messages that cannot be retrieved or processed from user queues are transferred to an exception queue. Queues are created, altered, started, stopped and dropped using the Oracle8 AQ administrative interface package DBMS_AQADM. There is no limit to the number of queues defined in the database. A Queue can have messages inserted (enqueued) by one or more producers (users), and messages retrieved (dequeued) by one or more consumers (users). Messages can be copied from queue to queue via message propagation. The destination queue(s) can be located in the local database or in remote database(s).

Queue Tables

Queues are stored in queue tables. Each queue table is a database table and contains one or more queues. Each queue table contains a default exception queue.

Each column of a queue table represents message queues and rows represent individual messages. Queue tables are created using the DBMS_AQADM.CREATE_QUEUE_TABLE procedure.

Agents

An agent is a queue user. There are 2 types of agents: producers and consumers. A producer places messages in queues; this process is called enqueueing. A consumer retrieves messages; this process is called dequeuing. Any number of producers and consumers may access the queue at a given time. An agent is identified by its name, address and protocol. The address and protocol are reserved for future use. Agents insert messages into and retrieve messages from queues using the Oracle8 AQ operational interface procedures DBMS_AQ.ENQUEUE and DBMS_AQ.DEQUEUE.

Time Manager

The time manager is a background process, 'QMN0', that monitors the messages in the queue. It provides the time management mechanism for message expiration, retry and delay. A parameter called AQ_TM_PROCESSES must be specified in the init<sid>.ora parameter file to perform time monitoring on queue messages. This parameter can only be set to values 0 or 1. If this parameter is set to 1, one time manager background process will be created. Set this to 0 during database upgrades and downgrades.

DBMS_AQADM, DBMS_AQ Packages

Oracle8 AQ operations are performed using the DBMS_AQADM and DBMS_AQ packages. Initially, only user 'SYS' has access to AQ operations and execute privileges for the procedures in the DBMS_AQADM and DBMS_AQ packages. Access to AQ objects and operations is granted to users through the AQ_ADMINISTRATOR_ROLE and AQ_USER_ROLE roles. The AQ_ADMINISTRATOR_ROLE role grants execute privilege to procedures in both the DBMS_AQADM and DBMS_AQ packages. The AQ_USER_ROLE role grants execute privileges to procedures in the DBMS_AQ package. To access AQ object types, the procedure DBMS_AQADM.GRANT_TYPE_ACCESS must be executed for the AQ administrator by the user 'SYS'. The AQ administrator can subsequently execute the DBMS_AQADM.GRANT_TYPE_ACCESS procedure to grant access for AQ object types to other AQ users. This procedure must be executed for any AQ user that wishes to perform any administrative operation involving a multiple consumer queue.

DBMS_AQADM - Package provides for creation and management of queues and queue tables. Procedures include:

- Create_Queue_Table()
- Alter_Queue_Table()
- Drop_Queue_Table()
- Create_Queue()
- Create_NP_Queue()
- Alter_Queue()
- Start_Queue()
- Stop_Queue()
- Drop_Queue()
- Grant_System_Privilege()
- Revoke_System_Privilege()
- Grant_Queue_Privilege()
- Migrate_Queue_Table()
- Disable_Propagation_Schedule()
- Revoke_Queue_Privilege()
- Start_Time_Manager()
- Stop_Time_Manager()
- Add_Subscriber()
- Alter_Subscriber()
- Remove_Subscriber()
- Queue_Subscribers()
- Schedule_Propagation()
- Unschedule_Propagation()
- Verify_Queue_Types()
- Alter_Propagation_Schedule()
- Enable_Propagation_Schedule()

DBMS_AQ - Package provides for enqueue and dequeue of messages to/from message queues. Also, `DBMS_AQ.Listen()` procedure is provided which waits for a message to dequeue but wastes no resources while waiting. Enqueue/dequeue procedures are described next.

DBMS_AQ.ENQUEUE - Parameters include (queue_name, enqueue_options, message_properties, payload, msg_id) where:

1. Queue_name - name of the queue; varchar2 type.
2. Enqueue_options - `DBMS_AQ.ENQUEUE_OPTIONS_T` record type consisting of:
 - Visibility determines when the process enqueueing is permitted to see the message (`ON_COMMIT` or `IMMEDIATE`)
 - Relative_msgid (relevant only when `sequence_deviation=BEFORE`)
 - Sequence_deviation (`BEFORE`, `TOP` or `NULL`)
3. Message_properties - `DBMS_AQ.MESSAGE_PROPERTIES_T` type consisting of:
 - Priority of the message (any number, where lower numbers have higher priority)
 - Delay in seconds before a message can be dequeued (`NO_DELAY` or a number)
 - Expiration in seconds before message can no longer be dequeued (`NEVER` or a number)
 - Correlation which identifies the message with a name.
 - Attempts represents the number of times consumers attempted to dequeue the message; not set at enqueue time.
 - Recipient_list is the list of consumers permitted to dequeue the message; defined as a table of `sys.AQ$AGENT` type. Used for multiconsumer queues.
 - Exception_queue is the queue where messages are moved after exceeding Expiration time or Retry limit.
 - Enqueue_time is set internally by the system time-stamping when the message was enqueued.
 - State is the current message availability (`WAITING`, `READY`, `PROCESSED`, or `EXPIRED`).
 - Sender_id specifies sender information.

- Original_msgid is used by AQ for propagating messages.
4. Payload - the actual message content which can be either of RAW or Object data type. PL/SQL allows RAW messages up to 32K.
 5. Msgid is a RAW datatype which returns the unique, generated handle which can be used to refer to the message by enqueueing or dequeuing processes.

DBMS_AQ.DEQUEUE - Parameters include (queue_name, dequeue_options, message_properties, payload, msg_id) where:

1. Queue_name - name of the queue; varchar2 type.
2. Dequeue_options - DBMS_AQ.DEQUEUE_OPTIONS_T record type consisting of:
 - Consumer_name is the name of the application, process or user receiving the message, or NULL if not a multiuser queue.
 - Dequeue_mode specifies lock required and disposition of the message (BROWSE for read-only, LOCKED for ability to update, REMOVE for ability to read/update/delete, and REMOVE_NODATA to mark the message as updated/deleted).
 - Navigation determines the position of the message to retrieve (NEXT_MESSAGE for next available matching message, FIRST_MESSAGE to reset queue position to first message that fits criteria, or NEXT_TRANSACTION to skip to the next transaction group of messages).
 - Visibility determines when the process dequeuing is permitted to see the message (ON_COMMIT or IMMEDIATE).
 - Wait tells how long to wait for a message for dequeue (FOREVER which blocks until a message arrives, NO_WAIT which returns immediately whether a message is ready or not, or number of seconds).
 - Msgid is the specific message handle to be dequeued.
 - Correlation which identifies the message by name. Wildcards are allowed.
3. Message_properties - Refer to DBMS_AQ.ENQUEUE above for description of Message_Properties.
4. Payload - the variable which will be loaded with the actual message content. Must be the same type definition as when enqueued.
5. Msgid - message handle to be dequeued.

Major AQ Features include:

Advanced queuing has the following capabilities.

- Persistent queuing - so that messages are never lost
- Message propagation - ability to store and forward messages from queue to queue, including remote databases
- Single/Multiple queue producers/consumers - flexibility in sharing queues
- Priority, Order, Scheduling, Retention, Expiration - message control information
- Scalability - integrated with Oracle Server and comprising regular Oracle tables; therefore, AQ reaps all the advantages of the Oracle database
- Complex routing and 1-way replication - replication with control
- History - audit capability
- Exception handling - error handling capability

Major New AQ Features in 8.1.x include:

With Oracle 8.1.x, advanced queuing gains the following capabilities.

- Faster propagation by 3 times
- Concurrent dequeuing - parallel dequeuing by different consumers
- Transaction grouping - mark messages as a single work unit
- Rule-based subscribers - filter messages individual subscribers receive based on rules
- Nonblocking listen for dequeue - similar to polling or waiting, but doesn't waste resources
- Nonblocking notification for dequeue - register with dequeue to be notified when a message matching criteria is available; doesn't waste resources
- Queue level and System level privileges - fine-grained security

Security

Granting execute on DBMS_AQ and DBMS_AQADM packages allows user to create/manage/use queues in user's own schema. Granting Manage Any Queue to a user allows him/her to create/manage other queues in other schemas. Difference between 8i and 8.0 security based upon whether queue is 8.0 or 8.1 compatible, a parameter in the DBMS_AQADM.Create_Queue_Table procedure.

With 8.1, can grant/revoke privileges at the object level on 8.1-compatible queues.

Security for 8.0-compatible in 8.1.x Database:

- a. AQ_USER_ROLE - User can enqueue/dequeue to/from any queue in the system
- b. AQ_ADMINISTRATOR_ROLE - Create/Manage any queue or queue table
- c. execute on DBMS_AQ - can perform enqueue/dequeue/listen functions

Security for 8.1-compatible in 8.1.x Database:

- a. execute on DBMS_AQ, Enqueue Any or Dequeue Any Queue system privilege; DBMS_AQADM.Grant_System_Privilege / Revoke_System_Privilege - available security for users
- d. AQ_ADMINISTRATOR_ROLE - Create/Manage any queue or queue table
- e. execute on DBMS_AQ - can perform enqueue/dequeue/listen functions

- b. Propagation user must have rights to enqueue to destination queue or Enqueue Any Queue system privilege (Unless user already owns the objects in the destination).

Other 8.1-Compatible Features include:

Other features which are available when Compatible is set to 8.1 in DBMS_AQADM.Create_Queue_Table() are:

8.1-compatible allows:

- Queue level access control
- NonPersistent Queues
- Rule-based subscribers for publish/subscribe
- Sender identification
- Separate storage for history information

8.1-compatible provides additional objects:

- Subscriber table - for subscribers
- History table - for auditing
- Rules table - for filtering

AQ Requirements

The following are requirements for configuring Advanced Queuing.

- DBMS_AQ and DBMS_AQADM packages- created by default install, else run \$ORACLE_HOME/rdbms/admin/dbmsaqad.sql as connect internal
- Net8
- Preferably Oracle 8.1.6+ to gain latest features; also, see Bugs section at end of document
- Objects option if using object type messages
- init.ora parameters
 - ◆ GLOBAL_NAMES = TRUE
 - ◆ JOB_QUEUE_INTERVAL = 10
 - ◆ JOB_QUEUE_PROCESSES = 5
 - ◆ AQ_TM_PROCESSES = 1
 - ◆ OPEN_LINKS = 255
 - ◆ OPEN_CURSORS = 500

Note: Except for AQ_TM_PROCESSES, numeric values listed are suggestions.

Application Scenarios for AQ

Typical examples of applications for AQ include:

- Message passing
- Workflow
- Data replication plus control
- Time-sensitive messaging
- Priority-based messaging

- Publish/subscribe model
- Others ...

General Implementation Steps

The following general steps are needed to implement Oracle Advanced Queuing.

1. Design the flow of messages and message content
2. Configure the database for advanced queuing; create AQ administrator and users
3. Create message type - raw or object type
4. Create queue table
5. Create queue
6. Start queue and associated exception queue
7. Add queue subscribers
8. Develop an enqueue process
9. Develop a dequeue process
10. Develop exception-handler for queue failures
11. Maintain and monitor

Note: Propagation requires similar steps in destination database, plus scheduling propagation.

Supported Interfaces

Oracle advanced queuing supports a number of standard and third party interfaces/gateways including:

Standard Interfaces:

- PL/SQL
- OCI (Pro*C/C++/others)
- Java
- JMS
- Visual Basic 0040
- XA

Third Party:

- IBM MQ Series
- MicroSoft MSMQ
- Vitria Velociti
- TIBCO TIB/Rendezvous
- Mercator

Maintenance

Maintenance and monitoring is important because advanced queuing is complex and problems can affect processing both upstream at point of enqueue, and downstream at point of dequeue or propagation. Keep the following in mind.

- Monitor exception queue and handle exception messages
- Queue table and index fragmentation
- Monitor Queue table counts, locks, broken jobs
- Pay attention to rollback and redo log size and performance
- Stopping outbound queue for maintenance requires coordination with enqueueing process(es) to ensure messages are not lost
- Mechanism to resynchronize if a failure occurs
- OEM provides monitoring and statistics for AQ

Oracle Enterprise Manager Support

Oracle Enterprise Manager supports several AQ administrative functions including:

- View queues and properties
- Create, start, stop, drop queues
- Schedule/unschedule propagation
- Add/remove subscribers
- View propagation schedule
- Grant/revoke privileges
- View queue and propagation statistics

Lessons Learned

The following issues surfaced during Advanced Queuing evaluation while getting a good example to work.

1. Needed to resize SYSTEM and USERS tablespaces to accommodate growth.
2. Decided to using Minimum Extent of 1M for AQADM default tablespace 'USERS' to control growth of associated objects.
3. Tweaked JOB_QUEUE_INTERVAL, DELAY, RETRY, EXPIRATION, LISTEN WAIT_TIME, Oracle Job NEXT_DATE, Enqueue interval in order to balance queue table growth, dequeue vs. enqueue lag time, and propagation lag time.
4. Determined that Propagated messages could only be dequeued by subscribers/recipients declared when the message was originally enqueued in the outbound queue.
5. Needed to size tables/tablespaces large enough to accommodate long propagation failure times.

For More Information

- Oracle documentation
- White papers including:
 1. [Oracle8 Advanced Queuing capabilities and an introduction to terms and concepts](#) - Metalink, Doc Id: 52748.1 dated 5/2000
 2. [Taking Advantage of the Oracle 8i Object Layer, Today.pdf](#) by Joe Hudicka, IOUG-A 2000
 3. [JMS Overview in 8.1.6.txt](#) - Metalink, dated 6/2000
 4. [Oracle AQ - Why Use It.pdf](#) by Mandip Bhuller, Openworld 2000
 5. [Oracle Applications Interconnect \(OAI\).pdf](#) by Robert Gaydos, Openworld 2000
 6. [E-Business Integration Using AQ 9i.pdf](#) by Brajesh Goyal, Openworld 2000
 7. [Advanced Queuing: What Everyone Should Know](#) by Doug Dosman, RMOUG 2001.

Bugs

Experienced two major bugs during evaluation.

1. Message propagation failed with ORA-00600 between HP-UX 8.0.5 and 8.1.6 databases. Work-around was to use 8.1.6 databases. This seems limited to HP-UX and may go away with latest patches. No specific Bug number.
2. Sys.aq\$ Pending_Messages table in system tablespace is not cleaned up after messages are successfully sent. Occurred between HP-UX 8.1.6.2 databases where

DBMS_AQADM.create_queue_table used COMPATIBLE='8.1' setting. Work-around is to use COMPATIBLE='8.0'. Bug #1099084.

Keep in mind that Advanced Queuing is a complex product so it is prudent to thoroughly test implemented AQ features when testing upgrades.

AQ Example

The following example shows the steps and code involved in sending messages from one database (TELNO) to a remote database (BILLING), both running Oracle 8.1.6.2. It is assumed that Advanced Queuing is available, recommended `init.ora` parameter changes have been made, and databases restarted. All steps are performed as AQADM user unless otherwise noted. Code scripts are listed beginning on page 13.

The goal of this example is to send data from a local table to a remote table along with control information.

This is accomplished by:

- A procedure reads three columns from a table in TELNO database, TELNO, and enqueues the data plus control information as a message into TELNO_OUT_QUEUE.
- Message propagation retrieves the message from TELNO_OUT_QUEUE in TELNO and enqueues it into BILLING_IN_QUEUE on the BILLING database.
- An oracle job runs on BILLING which listens for messages arriving in BILLING_IN_QUEUE and dequeues arriving messages and inserts the three columns into TELNO_TGT on BILLING database.

Test results for both 3 million and 4 million enqueued messages are also reported.

The steps and scripts used to implement the example are:

1. TELNO & BILLING: Create aqadm AQ administrator with privileges by running @cre_aqadm.sql as system. AQADM will also be the queue user in this example.
2. TELNO: Create database link to BILLING via @cre_aqadm_BILLINGb_link.sql
3. BILLING: Create database link to TELNO via @cre_aqadm_TELNO_db_link.sql
4. BILLING: Create a table, TELNO_TGT, which will ultimately receive dequeued data.
5. TELNO: Establish message type, queue table, outbound queue, and start queue with @TELNOdemo.sql
6. BILLING: Establish message type, queue table, inbound queue, and start queue with @BILLINGdemo.sql
7. TELNO: Add queue subscriber for outbound queue by running @TELNOsub.sql
8. TELNO: Schedule (start) propagation for the outbound queue to remote queue with @TELNOprop.sql
9. TELNO: Install enqproc procedure which reads data from TELNO table and performs enqueue into outbound queue by running @TELNOenqproc.sql
10. BILLING: Install monitor_queue procedure which listens for messages in the inbound queue, dequeues messages, and performs data inserts into TELNO_TGT table by running @BILLINGdeqmon.sql
11. TELNO: Invoke enqproc to enqueue 4,000,000 messages in batches of 4000 messages with batch interval delay of 5 seconds by running @runenq.sql
12. BILLING: Start oracle job to invoke monitor_queue periodically via rundeq.sql.

Note: The example given is simplified. The final implementation uses a trigger to perform enqueues based upon specific changes to data in the 3 columns of interest in TELNO. The dequeue procedure will apply the appropriate DML Insert/Update/Delete to the target table as needed. Queue users will be created to perform the enqueues/dequeues. Exception handling and monitoring will also be addressed.

Enqueue Test Results

For 3 million messages:

Total Enqueue Time for 3,000,000 messages = 4:03:25

Gross message rate => 3,000,000msgs/1,4605s = 205 msgs/sec

Net message rate w/o delays => 3,000,000/10,855s = 276 msgs/sec

For 4 million messages:

Total Enqueue Time for 4,000,000 messages = 5:24:48

Gross message rate => 4000000msgs/19488s = 205 msgs/sec

Net message rate w/o delays => 4000000/14488s = 276 msgs/sec

Notes: Both databases reside on the same host server. Message propagation and the dequeue process lagged behind only slightly given the built-in enqueue delays. Message propagation will likely take more time using remote hosts. Enqueue delays of 5 seconds between batches of 4000 messages provided ample time for message propagation in these cases.

Housekeeping items included

6. Resizing SYSTEM and USERS tablespaces to accommodate growth.
7. Using Minimum extent of 1M for AQADM default tablespace 'USERS' to control growth of associated objects.
8. Tweaking JOB_QUEUE_INTERVAL, DELAY, RETRY, EXPIRATION, LISTEN WAIT_TIME, Oracle Job NEXT_DATE, Enqueue interval.
9. Determining that Propagated messages could only be dequeued by subscribers/recipients declared when the message was originally enqueued in the outbound queue.
10. Sizing tables/tablespaces large enough to accommodate long propagation failure times.

Useful views/tables in this example

DBA_QUEUES - defined queues

DBA_QUEUE_SCHEDULES - defined AQ Propagation

DBA_JOBS - oracle jobs

TELNO_OUT_QUEUE_T - queue table for TELNO

BILLING_IN_QUEUE_T - queue table for BILLING

GV\$AQ - queue statistics

This section lists the scripts to carry out the example. First the scripts run on TELNO are listed. Beginning on page 18, the BILLING scripts are listed in *italic*.

**** cre_aqadm.sql - Create AQ administrator user for TELNO. *******

```
create user aqadm identified by xxxxxx
default tablespace users
temporary tablespace temp;
```

```
grant aq_administrator_role to aqadm;
```

```
grant connect, resource to aqadm;
grant execute on dbms_aq to aqadm;
grant execute on dbms_aqadm to aqadm;
grant execute on dbms_job to aqadm;
```

**** cre_aqadm BILLING_db_link.sql - Create database link to target database. ******

```
create database link BILLING
connect to aqadm identified by xxxxxx
using 'BILLING';
```

--Note: Database link must be non-anonymous

**** initTELNO.ora - AQ-related changes. *******

```
GLOBAL_NAMES = TRUE
OPEN_CURSORS = 100
OPEN_LINKS = 255
JOB_QUEUE_PROCESSES = 5      ##Creates snp0-snp4
JOB_QUEUE_INTERVAL = 10
AQ_TM_PROCESSES = 1         ##Creates qmn0
```

**** TELNOdemo.sql - Create Object Message Type, Queue Table, Queue, Start Queues.****

```
-- Create object message type
create type message_type as object (
    table_name          varchar2(40),
    inserted_date       date,
    dml_code            varchar2(1),
    telephone_no       varchar2(10),
    region              varchar2(10),
    history_ind         varchar2(1),
    activate_dt         date )
/
-- Create queue table
begin
    dbms_aqadm.create_queue_table (
        queue_table => 'aqadm.TELNO_out_queue_t',
        queue_payload_type => 'message_type',
        sort_list => 'PRIORITY,ENQ_TIME',
        multiple_consumers => TRUE,
        comment => 'Creating queue with priority and enq_time sort order',
        compatible => '8.0',
        storage_clause =>
            'storage(initial 5M next 5M pctincrease 0 maxextents 1000)
            tablespace USERS');
end;
/
-- Create queue
begin
```

```

    dbms_aqadm.create_queue (
        queue_table => 'aqadm.TELNO_out_queue_t',
        queue_name => 'aqadm.TELNO_out_queue');
end;
/
-- Start queue
begin
    dbms_aqadm.start_queue(
        queue_name => 'aqadm.TELNO_out_queue',
        dequeue => TRUE,
        enqueue => TRUE);
end;
/
-- Start exception queue
begin
    dbms_aqadm.start_queue(
        queue_name => 'AQ$_TELNO_OUT_QUEUE_T_E',
        enqueue => FALSE,
        dequeue => TRUE);
end;
/

```

**** TELNOsub.sql - Add subscriber for message propagation to BILLING target database. *******

```

DECLARE
    subscriber sys.aq$_agent;
BEGIN
    subscriber := sys.aq$_agent('TELNO_agent','BILLING_in_queue@BILLING',0);
    dbms_aqadm.add_subscriber(queue_name => 'TELNO_out_queue',
        subscriber => subscriber);
    commit;
END;
/

```

**** TELNOprop.sql - Schedule message propagation to target database. *******
 -- Propagate (copy) from TELNO_out_queue to a remote queue BILLING_in_queue.

```

begin
    dbms_aqadm.schedule_propagation(queue_name => 'aqadm.TELNO_out_queue',
        destination => 'BILLING',
        start_time => sysdate,
        duration => 120,
        next_time => 'sysdate + 1 / 24 / 60',
        latency => 10);
end;
/

```

```

** TELNOenqproc.sql - Create procedure to enqueue messages. *****
create or replace procedure enqproc (
TOTAL_MSGS  IN    NUMBER,
MSG_CHUNK   IN    NUMBER,
DELAY_SEC   IN    NUMBER
)
)
is
  enqueue_options      dbms_aq.enqueue_options_t;
  message_properties    dbms_aq.message_properties_t;
  message_handle        RAW(16);
  message              aqadm.message_type;
  table_name            varchar2(40);
  inserted_date         date;
  dml_code              varchar2(1);
  telephone_no         varchar2(10);
  region               varchar2(10);
  history_ind          varchar2(1);
  activate_dt          date;
  CTR                  number;
  TIMENOW              varchar2(11);

  CURSOR TELNOtype_cur IS
    SELECT
      A.telephone_no,
      A.region,
      A.activate_dt
    FROM TELNO A;

  TELNOrec TELNOtype_cur%ROWTYPE;

BEGIN
  -- Assign Enqueue Options
  enqueue_options.VISIBILITY := DBMS_AQ.ON_COMMIT; --else IMMEDIATE
  enqueue_options.RELATIVE_MSGID := '0'; --irrelevant w/o BEFORE
  --enqueue_options.SEQUENCE_DEVIATION := DBMS_AQ.NULL; --else BEFORE or TOP;

  -- Assign Message Properties
  message_properties.PRIORITY := 1;
  message_properties.DELAY := DBMS_AQ.NO_DELAY;
  message_properties.EXPIRATION := DBMS_AQ.NEVER;
  -- message_properties.CORRELATION := 'TELNO';
  -- message_properties.ATTEMPTS not set at time of enqueue
  -- subscriber(1) := SYS.AQ$_AGENT('DINAH',NULL,NULL);
  -- subscriber(2) := SYS.AQ$_AGENT('SPANKY',NULL,NULL);
  -- subscriber(3) := SYS.AQ$_AGENT('ATHENA',NULL,NULL);
  -- message_properties.RECIPIENT_LIST := subscriber;
  -- message_properties.EXCEPTION_QUEUE := 'NOT_HUNGRY_EXCEPTION_Q';
  -- message_properties.ENQUEUE_TIME not set by user
  -- message_properties.STATE not set by user

  -- Assign Message Type fields
  table_name := 'CORP.TELNO';
  inserted_date := sysdate;
  dml_code := 'I';
  history_ind := 'N';

  CTR := 1;

```

```

IF NOT TELNotype_cur%ISOPEN
THEN
  OPEN TELNotype_cur;
END IF;

FETCH TELNotype_cur INTO TELNorec;

TIMENOW := to_char(sysdate, 'HH24:MI:SS');
dbms_output.put_line('Starting Enqueue at time = '||TIMENOW);

WHILE (CTR < (TOTAL_MSGS +1)) and (TELNotype_cur%FOUND)
LOOP

if ( MOD(CTR, MSG_CHUNK) = 0) then
  TIMENOW := to_char(sysdate, 'HH24:MI:SS');
  dbms_output.put_line('Sleeping with CTR= '||CTR||' at time = '||TIMENOW);
  dbms_lock.sleep(DELAY_SEC);
end if;

  telephone_no := TELNorec.telephone_no;
  region := TELNorec.region;
  activate_dt := TELNorec.activate_dt;

-- Actual assignment of object record into payload
message := aqadm.message_type(table_name, inserted_date,dml_code,
                             telephone_no, region, history_ind, activate_dt);

dbms_aq.enqueue(queue_name => 'aqadm.TELNO_out_queue',
enqueue_options => enqueue_options,
message_properties => message_properties,
payload => message,
msgid => message_handle);

commit;
FETCH TELNotype_cur INTO TELNorec;
CTR := CTR + 1;
END LOOP;

CLOSE TELNotype_cur;

TIMENOW := to_char(sysdate, 'HH24:MI:SS');
dbms_output.put_line('Finishing Enqueue at time = '||TIMENOW);
END;
/

** runenq.sql - Call procedure to enqueue 4 million msgs in batches of 4000 with 5
sec delay. *****
execute enqproc(4000000,4000,5);

```

** chkusrobj2TELNO.lst - List of Aqadm-owned objects in TELNO. *****

OBJECT_NAME	OBJECT_TYPE
AQ\$TELNO_OUT_QUEUE_T	VIEW
MESSAGE_TYPE	TYPE
AQ\$ TELNO_OUT_QUEUE_T_I	TABLE
TELNO_OUT_QUEUE_T	TABLE
SYS_IOT_OVER_5515	TABLE
AQ\$ TELNO_OUT_QUEUE_T_E	QUEUE
TELNO_OUT_QUEUE	QUEUE
ENQPROC	PROCEDURE
SYS_LOB0000005510C00026\$\$	LOB
AQ\$ TELNO_OUT_QUEUE_T_T	INDEX
SYS_C001631	INDEX
SYS_IOT_TOP_5515	INDEX
BILLING.CORP.LITEL.COM	DATABASE LINK

```
** cre_aqadm.sql - Create AQ administrator user on BILLING *****
Note: Same as for TELNO database listed previously.
```

```
** cre_aqadm_TELNO_db_link.sql - Create db link to TELNO source database.
*****
create database link TELNO
connect to aqadm identified by xxxxxx
using 'TELNO';
```

```
** initBILLING.ora - AQ-related init.ora parameters. *****
GLOBAL_NAMES = TRUE
OPEN_CURSORS = 50
OPEN_LINKS = 4
JOB_QUEUE_INTERVAL = 10
JOB_QUEUE_PROCESSES = 5
AQ_TM_PROCESSES = 1
```

```
** BILLINGdemo.sql - Create Object message type, Queue Table, Queue, and Start
Queues. ***
```

```
-- Create object message type
create type message_type as object (
    table_name          varchar2(40),
    inserted_date       date,
    dml_code            varchar2(1),
    telephone_no       varchar2(10),
    region              varchar2(10),
    history_ind        varchar2(1),
    activate_dt         date )
/
-- Create queue table
begin
    dbms_aqadm.create_queue_table (
        queue_table => 'aqadm.BILLING_in_queue_t',
        queue_payload_type => 'message_type',
        sort_list => 'PRIORITY,ENQ_TIME',
        multiple_consumers => TRUE,
        comment => 'Creating queue with priority and enq_time sort order',
        compatible => '8.0',
        storage_clause =>
            'storage(initial 1M next 1M pctincrease 0 maxextents 1000)
            tablespace users');
end;
/
-- Create queue
begin
    dbms_aqadm.create_queue (
        queue_table => 'aqadm.BILLING_in_queue_t',
        queue_name => 'aqadm.BILLING_in_queue');
end;
/
-- Start queue
begin
    dbms_aqadm.start_queue(
        queue_name => 'aqadm.BILLING_in_queue',
        dequeue => TRUE,
        enqueue => TRUE);
end;
```

```

/
-- Start exception queue
begin
dbms_aqadm.start_queue(
  queue_name => 'AQ$_BILLING_IN_QUEUE_T_E',
  enqueue => FALSE,
  dequeue => TRUE);
end;
/

** BILLINGdeqmon.sql - Create procedure to listen for and dequeue messages. ***
create or replace procedure MONITOR_QUEUE(
fetch_duration IN NUMBER,
msg_chunk IN NUMBER
)
IS
  agent_w_message sys.aq$_agent;
  agent_list DBMS_AQ.aq$_agent_list_t;
  wait_time INTEGER := 180;
  listen_timeout EXCEPTION;
  pragma EXCEPTION_INIT(listen_timeout, -25254);
  fetch_timeout EXCEPTION;
  pragma EXCEPTION_INIT(fetch_timeout, -25228);
  monitor BOOLEAN := TRUE;
  begin_time NUMBER;
  end_time NUMBER;

-- Dequeue a message
dequeue_options dbms_aq.dequeue_options_t;
message_properties dbms_aq.message_properties_t;
message_handle RAW(16);
message aqadm.message_type;
table_name varchar2(40);
inserted_date date;
dml_code varchar2(1);
telephone_no varchar2(10);
region varchar2(10);
history_ind varchar2(1);
activate_dt date;
CTR number;
recips dbms_aq.aq$_recipient_list_t;
aqagent sys.aq$_agent;

BEGIN
  begin_time := DBMS_UTILITY.get_time;

  WHILE (monitor)
  LOOP
  BEGIN
    agent_list(1) := sys.aq$_agent('TELNO_AGENT', 'BILLING_IN_QUEUE', NULL);

    DBMS_AQ.listen(agent_list, wait_time, agent_w_message);

-- DBMS_OUTPUT.put_line('Agent ' || agent_w_message.name || ' Address ' ||
--agent_w_message.address);

    dequeue_options.consumer_name := 'TELNO_AGENT';

```

```

dequeue_options.WAIT := DBMS_AQ.NO_WAIT;
dequeue_options.NAVIGATION := DBMS_AQ.FIRST_MESSAGE;

WHILE (TRUE)
  LOOP

    dbms_aq.dequeue(queue_name => 'BILLING_IN_QUEUE',
                   dequeue_options => dequeue_options,
                   message_properties => message_properties,
                   payload => message,
                   msgid => message_handle);

    table_name          := message.table_name;
    inserted_date       := message.inserted_date;
    dml_code            := message.dml_code;
    telephone_no       := message.telephone_no;
    region              := message.region;
    activate_dt         := message.activate_dt;
    history_ind        := message.history_ind;

    insert into aqadm.TELNO_tgt(telephone_no, region, activate_dt)
    values (telephone_no, region, activate_dt);

  COMMIT;

END LOOP;

END;
end_time := DBMS_UTILITY.get_time;
IF ((end_time - begin_time) > fetch_duration) THEN
  monitor := FALSE;
--   DBMS_OUTPUT.put_line('Timing out after:' ||(end_time-begin_time)/100||
--'s');
  EXIT;
END IF;
END LOOP;
EXCEPTION
  WHEN listen_timeout THEN
    end_time := DBMS_UTILITY.get_time;
    IF ((end_time - begin_time) > fetch_duration) THEN
null;
--   DBMS_OUTPUT.put_line('Timing out after:' ||(end_time-begin_time)/100||
--'s');
    END IF;
  WHEN fetch_timeout THEN
    end_time := DBMS_UTILITY.get_time;
--   DBMS_OUTPUT.put_line('Fetch timeout after ' ||
--(end_time-begin_time)/100||'s');
  WHEN OTHERS
  THEN
    DECLARE
      error_code NUMBER := SQLCODE;
      error_msg VARCHAR2(300) := SQLERRM;
    BEGIN
NULL;
--   dbms_output.put_line('SQLCODE IS: ' ||error_code);
--   dbms_output.put_line('ERROR MSG IS: ' ||error_msg);

```

```
--      dbms_output.put_line('Done. ');
      END;
END;
/
```

**** chkusrobj2BILLING.lst - List of Aqadm-owned objects in BILLING.**

OBJECT_NAME	OBJECT_TYPE
-----	-----
AQ\$BILLING_IN_QUEUE_T	VIEW
MESSAGE_TYPE	TYPE
AQ\$_BILLING_IN_QUEUE_T_I	TABLE
BILLING_IN_QUEUE_T	TABLE
SYS_IOT_OVER_26956	TABLE
AQ\$_BILLING_IN_QUEUE_T_E	QUEUE
BILLING_IN_QUEUE	QUEUE
MONITOR_QUEUE	PROCEDURE
SYS_LOB0000026951C00026\$\$	LOB
AQ\$_BILLING_IN_QUEUE_T_T	INDEX
SYS_C001665	INDEX
SYS_IOT_TOP_26956	INDEX
TELNO.CORP.LITEL.COM	DATABASE LINK

**** rundeq.sql - Initiate Oracle Job to monitor and dequeue messages. ******

```
declare
dbms_job_no  binary_integer;
begin
dbms_job.submit(job=> dbms_job_no,
what => 'aqadm.monitor_queue(12000,5000);',
next_date => SYSDATE,
interval => 'SYSDATE + 1/24/12',
no_parse => FALSE );
end;
/
```

Developed by Robert Bond, Qwest Communications Corp., Database Architecture Team,
7/2001. Email: Robert.Bond@qwest.com
Note: Examples are not warranted to be free of defects.