Skip Headers

Oracle9*i* Application Developer's Guide - Advanced Queuing Release 2 (9.2)
Part Number A96587-01

Home Book List Contents Index Master Index Feedback

Previous Next

A
Oracle Advanced Queuing by Example

In this appendix we provide examples of operations using different programmatic environments:

- Creating Queue Tables and Queues

- Enqueuing and Dequeuing Messages

Creating Queue Tables and Queues

Note:

You may need to set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
```

Creating a Queue Table and Queue of Object Type

```
/* Creating a message type: */
```

```
CREATE type aq.Message_typ as object (
subject     VARCHAR2(30),
text        VARCHAR2(80));

/* Creating a object type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table        => 'aq.objmsgs80_qtab',
queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name         => 'msg_queue',
queue_table        => 'aq.objmsgs80_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'msg_queue');
```

Creating a Queue Table and Queue of Raw Type

```
/* Creating a RAW type queue table and queue: */
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table          => 'aq.RawMsgs_qtab',
queue_payload_type   => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name          => 'raw_msg_queue',
queue_table         => 'aq.RawMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name          => 'raw_msg_queue');
```

Creating a Prioritized Message Queue Table and Queue

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table        => 'aq.priority_msg',
sort_list          => 'PRIORITY,ENQ_TIME',
queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name         => 'priority_msg_queue',
queue_table        => 'aq.priority_msg');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'priority_msg_queue');
```

Creating a Multiconsumer Queue Table and Queue

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
queue_table        => 'aq.MultiConsumerMsgs_qtab',
multiple_consumers => TRUE,
queue_payload_type => 'aq.Message_typ');

EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name         => 'msg_queue_multiple',
queue_table        => 'aq.MultiConsumerMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'msg_queue_multiple');
```

Creating a Queue to Demonstrate Propagation

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (
queue_name        => 'another_msg_queue',
queue_table       => 'aq.MultiConsumerMsgs_qtab');

EXECUTE DBMS_AQADM.START_QUEUE (
queue_name         => 'another_msg_queue');
```

Setting Up Java AQ Examples

```
CONNECT system/manager

DROP USER aqjava CASCADE;
GRANT CONNECT, RESOURCE, AQ_ADMINISTRATOR_ROLE TO aqjava IDENTIFIED BY aqjava;
GRANT EXECUTE ON DBMS_AQADM TO aqjava;
GRANT EXECUTE ON DBMS_AQ TO aqjava;
CONNECT aqjava/aqjava

/* Set up main class from which we will call subsequent examples and handle
 exceptions: */
import java.sql.*;
import oracle.AQ.*;

public class test_aqjava
{
   public static void main(String args[])
   {
      AQSession  aq_sess = null;
      try
      {
         aq_sess = createSession(args);

   /* now run the test: */
        runTest(aq_sess);
      }
      catch (Exception ex)
      {
         System.out.println("Exception-1: " + ex);
         ex.printStackTrace();
      }
   }
}
```

Creating an Java AQ Session

```
/* Creating an Java AQ Session for the 'aqjava' user as shown in the
 AQDriverManager section above: */
 public static AQSession createSession(String args[])
   {
      Connection db_conn;
      AQSession  aq_sess = null;

      try
      {

         Class.forName("oracle.jdbc.driver.OracleDriver");
    /* your actual hostname, port number, and SID will
    vary from what follows. Here we use 'dlsun736,' '5521,'
    and 'test,' respectively: */

         db_conn =
                  DriverManager.getConnection(
                  "jdbc:oracle:thin:@dlsun736:5521:test",
                  "aqjava", "aqjava");
         System.out.println("JDBC Connection opened ");
         db_conn.setAutoCommit(false);

    /* Load the Oracle8i AQ driver: */
         Class.forName("oracle.AQ.AQOracleDriver");

    /* Creating an AQ Session: */
         aq_sess = AQDriverManager.createAQSession(db_conn);
         System.out.println("Successfully created AQSession ");
      }
      catch (Exception ex)
      {
         System.out.println("Exception: " + ex);
         ex.printStackTrace();
      }
      return aq_sess;
   }
```

Creating a Queue Table and Queue Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTableProperty     qtable_prop;
    AQQueueProperty          queue_prop;
    AQQueueTable             q_table;
    AQQueue                  queue;

 /* Creating a AQQueueTableProperty object (payload type - RAW): */
    qtable_prop = new AQQueueTableProperty("RAW");

 /* Creating a queue table called aq_table1 in aqjava schema: */
    q_table = aq_sess.createQueueTable ("aqjava", "aq_table1", qtable_prop);
    System.out.println("Successfully created aq_table1 in aqjava schema");

 /* Creating a new AQQueueProperty object */
    queue_prop = new AQQueueProperty();

 /* Creating a queue called aq_queue1 in aq_table1: */
    queue = aq_sess.createQueue (q_table, "aq_queue1", queue_prop);
    System.out.println("Successfully created aq_queue1 in aq_table1");
}

/* Get a handle to an existing queue table and queue: */
public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable            q_table;
    AQQueue                 queue;

 /* Get a handle to queue table - aq_table1 in aqjava schema: */
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table1");
    System.out.println("Successful getQueueTable");

 /* Get a handle to a queue - aq_queue1 in aqjava schema: */
    queue = aq_sess.getQueue ("aqjava", "aq_queue1");
    System.out.println("Successful getQueue");
}
```

Creating a Queue and Start Enqueue/Dequeue Using Java

```
{
     AQQueueTableProperty    qtable_prop;
     AQQueueProperty         queue_prop;
     AQQueueTable            q_table;
     AQQueue                 queue;

 /* Creating a AQQueueTable property object (payload type - RAW): */
     qtable_prop = new AQQueueTableProperty("RAW");
    qtable_prop.setCompatible("8.1");

 /* Creating a queue table called aq_table3 in aqjava schema: */
     q_table = aq_sess.createQueueTable ("aqjava", "aq_table3", qtable_prop);
     System.out.println("Successful createQueueTable");

 /* Creating a new AQQueueProperty object: */
     queue_prop = new AQQueueProperty();

 /* Creating a queue called aq_queue3 in aq_table3: */
     queue = aq_sess.createQueue (q_table, "aq_queue3", queue_prop);
     System.out.println("Successful createQueue");

 /* Enable enqueue/dequeue on this queue: */
     queue.start();
     System.out.println("Successful start queue");

 /* Grant enqueue_any privilege on this queue to user scott: */
     queue.grantQueuePrivilege("ENQUEUE", "scott");
     System.out.println("Successful grantQueuePrivilege");
}
```

Creating a Multiconsumer Queue and Add Subscribers Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTableProperty    qtable_prop;
     AQQueueProperty         queue_prop;
     AQQueueTable            q_table;
     AQQueue                 queue;
     AQAgent                  subs1, subs2;

 /* Creating a AQQueueTable property object (payload type - RAW): */
     qtable_prop = new AQQueueTableProperty("RAW");
     System.out.println("Successful setCompatible");

 /* Set multiconsumer flag to true: */
     qtable_prop.setMultiConsumer(true);

 /* Creating a queue table called aq_table4 in aqjava schema: */
     q_table = aq_sess.createQueueTable ("aqjava", "aq_table4", qtable_prop);
     System.out.println("Successful createQueueTable");

 /* Creating a new AQQueueProperty object: */
     queue_prop = new AQQueueProperty();
     /* Creating a queue called aq_queue4 in aq_table4 */
     queue = aq_sess.createQueue (q_table, "aq_queue4", queue_prop);
     System.out.println("Successful createQueue");

 /* Enable enqueue/dequeue on this queue: */
     queue.start();
     System.out.println("Successful start queue");

 /* Add subscribers to this queue: */
     subs1 = new AQAgent("GREEN", null, 0);
     subs2 = new AQAgent("BLUE", null, 0);

     queue.addSubscriber(subs1, null); /* no rule   */
     System.out.println("Successful addSubscriber 1");

     queue.addSubscriber(subs2, "priority < 2"); /* with rule */
     System.out.println("Successful addSubscriber 2");
}
```

Enqueuing and Dequeuing Of Messages

Enqueuing and Dequeuing of Object Type Messages Using PL/SQL

To enqueue a single message without any other parameters specify the queue name and the payload.

```
/* Enqueue to msg_queue: */
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   message := message_typ('NORMAL MESSAGE',
   'enqueued to msg_queue first.');

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

   COMMIT;

/* Dequeue from msg_queue: */
DECLARE
   dequeue_options     dbms_aq.dequeue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
           dequeue_options    => dequeue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                  ' ... ' || message.text );
   COMMIT;
END;
```

Enqueuing and Dequeuing of Object Type Messages Using Pro*C/C++

Note:

You may need to set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
```

```
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \
code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq
```

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>
/* The header file generated by processing
object type 'aq.Message_typ': */
#include  "pceg.h"

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
Message_typ     *message = (Message_typ*)0;  /* payload */
message_type_ind *imsg;             /*payload indicator*/
char             user[60]="aq/AQ";  /* user logon password */
char             subject[30];  /* components of the */
char             txt[80];      /* payload type */

/* ENQUEUE and DEQUEUE to an OBJECT QUEUE */

/*  Connect to database: */
EXEC SQL CONNECT :user;

/* On an oracle error print the error number :*/
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* Allocate memory for the host variable from the object cache : */
EXEC SQL ALLOCATE :message;

/* ENQUEUE */

strcpy(subject, "NORMAL ENQUEUE");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* Initialize the components of message : */
EXEC SQL OBJECT SET subject, text OF  :message TO :subject, :txt;

/* Embedded PLSQL call to the AQ enqueue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable 'message' to the payload: */
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message:imsg,    /* indicator has to be specified */
msgid => msgid);
END;
END-EXEC;
/* Commit work */
EXEC SQL COMMIT;

printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* Dequeue */

/* Embedded PLSQL call to the AQ dequeue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Return the payload into the host variable 'message':  */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
 /* Commit work :*/
EXEC SQL COMMIT;

/* Extract the components of message: */
EXEC SQL OBJECT GET SUBJECT,TEXT FROM :message INTO :subject,:txt;

printf("Dequeued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);
}
```

Enqueuing and Dequeuing of Object Type Messages Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv       *envhp;
  OCIServer    *srvhp;
  OCIError     *errhp;
  OCISvcCtx    *svchp;
  dvoid        *tmp;
  OCIType      *mesg_tdo = (OCIType *) 0;
  message      msg;
  null_message nmsg;
  message      *mesg     = &msg;
  null_message *nmesg    = &nmsg;
  message      *deqmesg  = (message *)0;
  null_message *ndeqmesg = (null_message *)0;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit(&envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
      (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

 /* Obtain TDO of message_typ */
  OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
                (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
                (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

 /* Prepare the message payload */
  mesg->subject = (OCIString *)0;
  mesg->data = (OCIString *)0;
  OCIStringAssignText(envhp, errhp,
                      (CONST text *)"NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
                       &mesg->subject);


  OCIStringAssignText(envhp, errhp,
                      (CONST text *)"OCI ENQUEUE", strlen("OCI ENQUEUE"),
                      &mesg->data);
  nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

 /* Enqueue into the msg_queue */
  OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
           mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
  OCITransCommit(svchp, errhp, (ub4) 0);

 /* Dequeue from the msg_queue */
  OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
           mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Enqueuing and Dequeuing of Object Type Messages (CustomDatum interface) Using Java

To enqueue and dequeue of object type messages follow the lettered steps:

a. Create the SQL type for the Queue Payload

```
connect aquser/aquser
create type ADDRESS as object (street VARCHAR (30), city VARCHAR(30));
create type PERSON as object (name VARCHAR (30), home ADDRESS);
```

b. Generate the java class that maps to the PERSON ADT and implements the CustomDatum interface (using Jpublisher tool)

```
jpub -user=aquser/aquser -sql=ADDRESS,PERSON -case=mixed -usertypes=oracle
-methods=false  -compatible=CustomDatum
This creates two classes - PERSON.java and ADDRESS.java corresponding to the
PERSON and ADDRESS Adt types.
```

c. Create the queue table and queue with ADT payload

d. Enqueue and dequeue messages containing object payloads

```
public static void AQObjectPayloadTest(AQSession aq_sess)
       throws AQException, SQLException, ClassNotFoundException
  {
    Connection           db_conn   = null;
    AQQueue              queue     = null;
    AQMessage            message   = null;
    AQObjectPayload      payload   = null;
    AQEnqueueOption      eq_option = null;
    AQDequeueOption      dq_option = null;
    PERSON           pers = null;
    PERSON           pers2= null;
    ADDRESS          addr = null;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();


    queue = aq_sess.getQueue("aquser", "test_queue2");


    /* Enable enqueue/dequeue on this queue */
    queue.start();

    /* Enqueue a message in test_queue2 */
    message = queue.createMessage();

    pers = new PERSON();
    pers.setName("John");
    addr = new ADDRESS();
    addr.setStreet("500 Easy Street");
    addr.setCity("San Francisco");
    pers.setHome(addr);

    payload = message.getObjectPayload();
    payload.setPayloadData(pers);
    eq_option = new AQEnqueueOption();

    /* Enqueue a message into test_queue2 */
    queue.enqueue(eq_option, message);

    db_conn.commit();

    /* Dequeue a message from test_queue2 */
    dq_option = new AQDequeueOption();
    message = ((AQOracleQueue)queue).dequeue(dq_option, PERSON.getFactory());
    payload = message.getObjectPayload();
    pers2 = (PERSON) payload.getPayloadData();

    System.out.println("Object data retrieved:  [PERSON]");
    System.out.println("Name:   " + pers2.getName());
    System.out.println("Address ");
    System.out.println("Street: " + pers2.getHome().getStreet());
    System.out.println("City:   " + pers2.getHome().getCity());

    db_conn.commit();
  }
```

Enqueuing and Dequeuing of Object Type Messages (using SQLData interface) Using Java

To enqueue and dequeue of object type messages follow the lettered steps:

a. Create the SQL type for the Queue Payload

```
connect aquser/aquser
create type EMPLOYEE as object (empname VARCHAR (50), empno INTEGER);
```

b. Creating a java class that maps to the EMPLOYEE ADT and implements the SQLData interface. This class can also be generated using JPublisher using the following syntax

```
jpub -user=aquser/aquser -sql=EMPLOYEE -case=mixed -usertypes=jdbc
-methods=false

import java.sql.*;
import oracle.jdbc2.*;

public class Employee implements SQLData
{
  private String sql_type;
  public String empName;
  public int empNo;
  public Employee()
  {}
  public Employee (String sql_type, String empName, int empNo)
  {
    this.sql_type = sql_type;
    this.empName = empName;
    this.empNo = empNo;
  }

  ////// implements SQLData //////
  public String getSQLTypeName() throws SQLException
  { return sql_type;
  }
  public void readSQL(SQLInput stream, String typeName)
    throws SQLException
  {
    sql_type = typeName;
    empName = stream.readString();
    empNo = stream.readInt();
  }

  public void writeSQL(SQLOutput stream)
    throws SQLException
  {
    stream.writeString(empName);
    stream.writeInt(empNo);
  }

  public String toString()
  {
String ret_str = "";
    ret_str += "[Employee]\n";
    ret_str += "Name: " + empName + "\n";
    ret_str += "Number: " + empNo + "\n";

    return ret_str;
  }
}
```

c. Create the queue table and queue with ADT payload

```
public static void createEmployeeObjQueue(AQSession aq_sess)
       throws AQException
  {
    AQQueueTableProperty qt_prop   = null;
    AQQueueProperty      q_prop    = null;
    AQQueueTable         q_table   = null;
    AQQueue              queue     = null;

    /* Message payload type is aquser.EMPLOYEE */
    qt_prop = new AQQueueTableProperty("AQUSER.EMPLOYEE");
    qt_prop.setComment("queue-table1");

    /* Creating aQTable1 */
    System.out.println("\nCreate QueueTable: [aqtable1]");
    q_table = aq_sess.createQueueTable("aquser", "aqtable1", qt_prop);

    /* Create test_queue1 */
    q_prop = new AQQueueProperty();
    queue = q_table.createQueue("test_queue1", q_prop);

    /* Enable enqueue/dequeue on this queue */
    queue.start();
  }
```

d. Enqueue and dequeue messages containing object payloads

```
public static void AQObjectPayloadTest2(AQSession aq_sess)
       throws AQException, SQLException, ClassNotFoundException
  {
    Connection           db_conn   = null;
    AQQueue              queue     = null;
    AQMessage            message   = null;
    AQObjectPayload      payload   = null;
    AQEnqueueOption      eq_option = null;
    AQDequeueOption      dq_option = null;
    Employee             emp  = null;
    Employee             emp2 = null;
    Hashtable            map;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

    /* Get the Queue object */
    queue = aq_sess.getQueue("aquser", "test_queue1");

    /* Register Employee class (corresponding to EMPLOYEE Adt)
     * in the connection type map
     */
    try
    {
      map  = (java.util.Hashtable)(((OracleConnection)db_conn).getTypeMap());
      map.put("AQUSER.EMPLOYEE", Class.forName("Employee"));
    }
    catch(Exception ex)
    {
      System.out.println("Error registering type: " + ex);
    }

    /* Enqueue a message in test_queue1 */
    message = queue.createMessage();
    emp = new Employee("AQUSER.EMPLOYEE", "Mark", 1007);

    /* Set the object payload */
    payload = message.getObjectPayload();
    payload.setPayloadData(emp);

    /* Enqueue a message into test_queue1*/
    eq_option = new AQEnqueueOption();
    queue.enqueue(eq_option, message);
    db_conn.commit();


    /* Dequeue a message from test_queue1 */
    dq_option = new AQDequeueOption();

    message = queue.dequeue(dq_option, Class.forName("Employee"));
    payload = message.getObjectPayload();
    emp2 = (Employee) payload.getPayloadData();
    System.out.println("\nObject data retrieved:  [EMPLOYEE]");
    System.out.println("Name  : " + emp2.empName);
    System.out.println("EmpId : " + emp2.empNo);

    db_conn.commit();
  }
```

Enqueuing and Dequeuing of RAW Type Messages Using PL/SQL

```
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             RAW(4096);

BEGIN
   message :=  HEXTORAW(RPAD('FF',4095,'FF'));
   DBMS_AQ.ENQUEUE(queue_name => 'raw_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
                     payload  => message,
                      msgid   => message_handle);

   COMMIT;
END;

/* Dequeue from raw_msg_queue: */
/* Dequeue from raw_msg_queue: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             RAW(4096);

BEGIN
```

```
   DBMS_AQ.DEQUEUE(queue_name => 'raw_msg_queue',
           dequeue_options    => dequeue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   COMMIT;
END;
```

Enqueuing and Dequeuing of RAW Type Messages Using Pro*C/C++

Note:

You may need to set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \
code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq
```

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

main()
{
LNOCIEnv         *oeh;   /* OCI Env handle */
LNOCIError       *err;   /* OCI Err handle */
LNOCIRaw         *message= (OCIRaw*)0;   /* payload */
ub1            message_txt[100];   /* data for payload */
char           user[60]="aq/AQ"; /* user logon password */
int            status;  /* returns status of the OCI call */

/* Enqueue and dequeue to a RAW queue */

/* Connect to database: */
EXEC SQL CONNECT :user;

/* On an oracle error print the error number: */
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");
 /* Get the OCI Env handle: */
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
printf(" error in SQLEnvGet \n");
exit(1);
}
/* Get the OCI Error handle: */
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0))
{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}

/* Enqueue */
/* The bytes to be put into the raw payload:*/
strcpy(message_txt, "Enqueue to a Raw payload queue ");

/* Assign bytes to the OCIRaw pointer :
Memory needs to be allocated explicitly to OCIRaw*: */
if (status=OCIRawAssignBytes(oeh, err, message_txt, 100,
 &message))
{
printf(" error in  OCIRawAssignBytes  %d \n", status);
exit(1);
}

/*  Embedded PLSQL call to the AQ enqueue procedure : */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable message to the raw payload: */
dbms_aq.enqueue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
 /* Dequeue */
/*  Embedded PLSQL call to the AQ dequeue procedure :*/ EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Return the raw payload into the host variable 'message':*/
dbms_aq.dequeue(queue_name => 'raw_msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */ EXEC SQL COMMIT;
}
```

Enqueuing and Dequeuing of RAW Type Messages Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

int main()
```

```
{
  OCIEnv       *envhp;
  OCIServer    *srvhp;
  OCIError     *errhp;
  OCISvcCtx    *svchp;
  dvoid        *tmp;
  OCIType      *mesg_tdo = (OCIType *) 0;
  char         msg_text[100];
  OCIRaw       *mesg = (OCIRaw *)0;
  OCIRaw       *deqmesg = (OCIRaw *)0;
  OCIInd       ind = 0;
  dvoid        *indptr = (dvoid *)&ind;
  int          i;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

 /* Obtain the TDO of the RAW data type */
  OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQADM", strlen("AQADM"),
              (CONST text *)"RAW", strlen("RAW"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

 /* Prepare the message payload */
  strcpy(msg_text, "Enqueue to a RAW queue");
  OCIRawAssignBytes(envhp, errhp, msg_text, strlen(msg_text), &mesg);

 /* Enqueue the message into raw_msg_queue */
  OCIAQEnq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
        mesg_tdo, (dvoid **)&mesg, (dvoid **)&indptr, 0, 0);
  OCITransCommit(svchp, errhp, (ub4) 0);

 /* Dequeue the same message into C variable deqmesg */
  OCIAQDeq(svchp, errhp, (CONST text *)"raw_msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&indptr, 0, 0);
  for (i = 0; i < OCIRawSize(envhp, deqmesg); i++)
    printf("%c", *(OCIRawPtr(envhp, deqmesg) + i));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Enqueue of RAW Messages using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTable             q_table;
     AQQueue                  queue;
```

```
    AQMessage                message;
    AQRawPayload             raw_payload;
    AQEnqueueOption          enq_option;
    String                   test_data = "new message";
    byte[]                   b_array;
    Connection               db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

/* Get a handle to queue table - aq_table4 in aqjava schema: */
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
    System.out.println("Successful getQueueTable");

/* Get a handle to a queue - aq_queue4 in aquser schema: */
    queue = aq_sess.getQueue ("aqjava", "aq_queue4");
    System.out.println("Successful getQueue");

/* Creating a message to contain raw payload: */
    message = queue.createMessage();

/* Get handle to the AQRawPayload object and populate it with raw data: */
    b_array = test_data.getBytes();

    raw_payload = message.getRawPayload();

    raw_payload.setStream(b_array, b_array.length);

/* Creating a AQEnqueueOption object with default options: */
    enq_option = new AQEnqueueOption();
/* Enqueue the message: */
    queue.enqueue(enq_option, message);

    db_conn.commit();
}
```

Dequeue of Messages Using Java

```
   public static void runTest(AQSession aq_sess) throws AQException
{
    AQQueueTable             q_table;
    AQQueue                  queue;
    AQMessage                message;
    AQRawPayload             raw_payload;
    AQEnqueueOption          enq_option;
    String                   test_data = "new message";
    AQDequeueOption          deq_option;
    byte[]                   b_array;
    Connection               db_conn;

    db_conn = ((AQOracleSession)aq_sess).getDBConnection();

/* Get a handle to queue table - aq_table4 in aqjava schema: */
    q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
    System.out.println("Successful getQueueTable");

/* Get a handle to a queue - aq_queue4 in aquser schema: */
    queue = aq_sess.getQueue ("aqjava", "aq_queue4");
    System.out.println("Successful getQueue");

/* Creating a message to contain raw payload: */
    message = queue.createMessage();

/* Get handle to the AQRawPayload object and populate it with raw data: */
    b_array = test_data.getBytes();
```

```
        raw_payload = message.getRawPayload();

        raw_payload.setStream(b_array, b_array.length);

    /* Creating a AQEnqueueOption object with default options: */
        enq_option = new AQEnqueueOption();

    /* Enqueue the message: */
        queue.enqueue(enq_option, message);
        System.out.println("Successful enqueue");

        db_conn.commit();

    /* Creating a AQDequeueOption object with default options: */
        deq_option = new AQDequeueOption();

    /* Dequeue a message: */
        message = queue.dequeue(deq_option);
        System.out.println("Successful dequeue");

    /* Retrieve raw data from the message: */
        raw_payload = message.getRawPayload();

        b_array = raw_payload.getBytes();

        db_conn.commit();
}
```

Dequeue of Messages in Browse Mode Using Java

```
   public static void runTest(AQSession aq_sess) throws AQException
{
      AQQueueTable            q_table;
      AQQueueTable            q_table;
      AQQueue                 queue;
      AQMessage               message;
      AQRawPayload            raw_payload;
      AQEnqueueOption         enq_option;
      String                  test_data = "new message";
      AQDequeueOption         deq_option;
      byte[]                   b_array;
      Connection              db_conn;

      db_conn = ((AQOracleSession)aq_sess).getDBConnection();

 /* Get a handle to queue table - aq_table4 in aqjava schema: */
      q_table = aq_sess.getQueueTable ("aqjava", "aq_table4");
      System.out.println("Successful getQueueTable");

 /* Get a handle to a queue - aq_queue4 in aquser schema: */
      queue = aq_sess.getQueue ("aqjava", "aq_queue4");
      System.out.println("Successful getQueue");

 /* Creating a message to contain raw payload: */
      message = queue.createMessage();

 /* Get handle to the AQRawPayload object and populate it with raw data: */
      b_array = test_data.getBytes();

      raw_payload = message.getRawPayload();

      raw_payload.setStream(b_array, b_array.length);

 /* Creating a AQEnqueueOption object with default options: */
      enq_option = new AQEnqueueOption();
 /* Enqueue the message: */
      queue.enqueue(enq_option, message);
      System.out.println("Successful enqueue");

      db_conn.commit();

 /* Creating a AQDequeueOption object with default options: */
      deq_option = new AQDequeueOption();

 /* Set dequeue mode to BROWSE: */
      deq_option.setDequeueMode(AQDequeueOption.DEQUEUE_BROWSE);

 /* Set wait time to 10 seconds: */
      deq_option.setWaitTime(10);

 /* Dequeue a message: */
      message = queue.dequeue(deq_option);

 /* Retrieve raw data from the message: */
      raw_payload = message.getRawPayload();
      b_array = raw_payload.getBytes();

      String ret_value = new String(b_array);
      System.out.println("Dequeued message: " + ret_value);

      db_conn.commit();
}
```

Enqueuing and Dequeuing of Messages by Priority Using PL/SQL

When two messages are enqued with the same priority, the message which was enqued earlier will be dequeued first. However, if two messages are of different priorities, the message with the lower value (higher priority) will be dequeued first.

```
/* Enqueue two messages with priority 30 and 5: */
DECLARE
   enqueue_options     dbms_aq.enqueue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   message := message_typ('PRIORITY MESSAGE',
   'enqued at priority 30.');

   message_properties.priority := 30;

   DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   message := message_typ('PRIORITY MESSAGE',
   'Enqueued at priority 5.');

   message_properties.priority := 5;

   DBMS_AQ.ENQUEUE(queue_name => 'priority_msg_queue',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);
END;

/* Dequeue from priority queue: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   DBMS_AQ.DEQUEUE(queue_name  => 'priority_msg_queue',
         dequeue_options       => dequeue_options,
         message_properties    => message_properties,
         payload               => message,
         msgid                 => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );

   COMMIT;

   DBMS_AQ.DEQUEUE(queue_name => 'priority_msg_queue',
         dequeue_options      => dequeue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );
   COMMIT;
END;

/* On return, the second message with priority set to 5 will be retrieved before
the message with priority set to 30 since priority takes precedence over enqueue
time. */
```

Enqueue of Messages with Priority Using Java

```
public static void runTest(AQSession aq_sess) throws AQException
{
     AQQueueTable             q_table;
     AQQueue                  queue;
     AQMessage                message;
     AQMessageProperty        m_property;
     AQRawPayload             raw_payload;
     AQEnqueueOption          enq_option;
     String                   test_data;
     byte[]                   b_array;
     Connection               db_conn;

     db_conn = ((AQOracleSession)aq_sess).getDBConnection();

 /* Get a handle to queue table - aq_table4 in aqjava schema: */
     qtable = aq_sess.getQueueTable ("aqjava", "aq_table4");
     System.out.println("Successful getQueueTable");

 /* Get a handle to a queue - aq_queue4 in aqjava schema: */
     queue = aq_sess.getQueue ("aqjava", "aq_queue4");
     System.out.println("Successful getQueue");

 /* Enqueue 5 messages with priorities with different priorities: */
     for (int i = 0; i < 5; i++ )
       {
     /* Creating a message to contain raw payload: */
```

```
          message = queue.createMessage();

          test_data = "Small_message_" + (i+1);      /* some test data */

         /* Get a handle to the AQRawPayload object and
        populate it with raw data: */
          b_array = test_data.getBytes();

          raw_payload = message.getRawPayload();

          raw_payload.setStream(b_array, b_array.length);

     /* Set message priority: */
         m_property = message.getMessageProperty();

         if( i < 2)
            m_property.setPriority(2);
          else
             m_property.setPriority(3);

     /* Creating a AQEnqueueOption object with default options: */
          enq_option = new AQEnqueueOption();

     /* Enqueue the message: */
         queue.enqueue(enq_option, message);
         System.out.println("Successful enqueue");
     }

     db_conn.commit();
}
```

Dequeue of Messages after Preview by Criterion Using PL/SQL

An application can preview messages in browse mode or locked mode without deleting the message. The message of interest can then be removed from the queue.

```
/* Enqueue 6 messages to msg_queue
-- GREEN, GREEN, YELLOW, VIOLET, BLUE, RED */

DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN
   message := message_typ('GREEN',
   'GREEN enqueued to msg_queue first.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);

   message := message_typ('GREEN',
   'GREEN also enqueued to msg_queue second.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
         enqueue_options      => enqueue_options,
         message_properties   => message_properties,
         payload              => message,
         msgid                => message_handle);
```

```
      message := message_typ('YELLOW',
      'YELLOW enqueued to msg_queue third.');

      DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
            enqueue_options      => enqueue_options,
            message_properties   => message_properties,
            payload              => message,
            msgid                => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message handle: ' || message_handle);

      message := message_typ('VIOLET',
      'VIOLET enqueued to msg_queue fourth.');

      DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
            enqueue_options      => enqueue_options,
            message_properties   => message_properties,
            payload              => message,
            msgid                => message_handle);

      message := message_typ('BLUE',
      'BLUE enqueued to msg_queue fifth.');

      DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
            enqueue_options      => enqueue_options,
            message_properties   => message_properties,
            payload              => message,
            msgid                => message_handle);

      message := message_typ('RED',
      'RED enqueued to msg_queue sixth.');

      DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
            enqueue_options      => enqueue_options,
            message_properties   => message_properties,
            payload              => message,
            msgid                => message_handle);

      COMMIT;
   END;

   /* Dequeue in BROWSE mode until RED is found,
   and remove RED from queue: */
   DECLARE
      dequeue_options     DBMS_AQ.dequeue_options_t;
      message_properties  DBMS_AQ.message_properties_t;
      message_handle      RAW(16);
      message             aq.message_typ;

   BEGIN
      dequeue_options.dequeue_mode := DBMS_AQ.BROWSE;

      LOOP
         DBMS_AQ.DEQUEUE(queue_name          => 'msg_queue',
                         dequeue_options    => dequeue_options,
                         message_properties => message_properties,
                         payload            => message,
                         msgid              => message_handle);

         DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                         ' ... ' || message.text );

         EXIT WHEN message.subject = 'RED';

      END LOOP;

      dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
      dequeue_options.msgid        := message_handle;

      DBMS_AQ.DEQUEUE(queue_name => 'msg_queue',
              dequeue_options    => dequeue_options,
              message_properties => message_properties,
              payload            => message,
              msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
      ' ... ' || message.text );

      COMMIT;
   END;

   /* Dequeue in LOCKED mode until BLUE is found,
   and remove BLUE from queue: */
   DECLARE
   dequeue_options     dbms_aq.dequeue_options_t;
   message_properties  dbms_aq.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

   BEGIN
   dequeue_options.dequeue_mode := dbms_aq.LOCKED;

         LOOP

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

   dbms_output.put_line ('Message: ' || message.subject ||
             ' ... ' || message.text );

   EXIT WHEN message.subject = 'BLUE';
         END LOOP;

   dequeue_options.dequeue_mode := dbms_aq.REMOVE;
   dequeue_options.msgid        := message_handle;

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

   DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
   ' ... ' || message.text );

         COMMIT;
   END;
```

Enqueuing and Dequeuing of Messages with Time Delay and Expiration Using PL/SQL

Note:

Expiration is calculated from the earliest dequeue time. So, if an application wants a message to be dequeued no earlier than a week from now, but no later than 3 weeks from now, this requires setting the expiration time for 2 weeks. This scenario is described in the following code segment.

```
/* Enqueue message for delayed availability: */
DECLARE
enqueue_options     dbms_aq.enqueue_options_t;
message_properties  dbms_aq.message_properties_t;
message_handle      RAW(16);
message             aq.Message_typ;

BEGIN
message := Message_typ('DELAYED',
'This message is delayed one week.');
message_properties.delay := 7*24*60*60;
message_properties.expiration := 2*7*24*60*60;

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

      COMMIT;
END;
```

Enqueuing and Dequeuing of Messages by Correlation and Message ID Using Pro*C/C++

Note:

You may need to set up data structures similar to the following for certain examples to work:

```
$ cat >> message.typ
case=lower
type aq.message_typ
$
$ ott userid=aq/aq intyp=message.typ outtyp=message_o.typ \
code=c hfile=demo.h
$
$ proc intyp=message_o.typ iname=<program name> \
config=<config file> SQLCHECK=SEMANTICS userid=aq/aq
```

```
#include  <stdio.h>
#include  <string.h>
#include  <sqlca.h>
#include  <sql2oci.h>
/* The header file generated by processing
object type 'aq.Message_typ': */
#include  "pceg.h"

void sql_error(msg)
char *msg;
{
EXEC SQL WHENEVER SQLERROR CONTINUE;
printf("%s\n", msg);
printf("\n% .800s \n", sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}
```

```
main()
{
LNOCIEnv            *oeh;  /* OCI Env Handle */
LNOCIError          *err;  /* OCI Error Handle */
Message_typ      *message = (Message_typ*)0; /* queue  payload */
message_type_ind *imsg;              /*payload indicator*/
LNOCIRaw            *msgid = (OCIRaw*)0; /* message id */
ub1               msgmem[16]="";  /* memory for msgid */
char              user[60]="aq/AQ";  /* user login password */
char              subject[30];  /* components of */
char              txt[80];  /* Message_typ */
char              correlation1[30];  /* message correlation  */
char              correlation2[30];
int               status; /* code returned by the OCI calls */

/* Dequeue by correlation and msgid */

/* Connect to the database: */
EXEC SQL CONNECT :user;
EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle Error :");

/* Allocate space in the object cache for the host variable: */
EXEC SQL ALLOCATE :message;

/* Get the OCI Env handle: */
if (SQLEnvGet(SQL_SINGLE_RCTX, &oeh) != OCI_SUCCESS)
{
 printf(" error in SQLEnvGet \n");
 exit(1);
}
/* Get the OCI Error handle: */
if (status = OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
(ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0))
{
printf(" error in OCIHandleAlloc %d \n", status);
exit(1);
}


/* Assign memory for msgid:
Memory needs to be allocated explicitly to OCIRaw*: */
if (status=OCIRawAssignBytes(oeh, err, msgmem, 16, &msgid))
{
printf(" error in  OCIRawAssignBytes  %d \n", status);
exit(1);
}

/* First enqueue */

strcpy(correlation1, "1st message");
strcpy(subject, "NORMAL ENQUEUE1");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");
 /* Initialize the components of message: */
EXEC SQL OJECT SET subject, text OF :message TO :subject, :txt;

/* Embedded PLSQL call to the AQ enqueue procedure: */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
BEGIN
/* Bind the host variable 'correlation1': to message correlation*/
message_properties.correlation := :correlation1;

/* Bind the host variable 'message' to payload and
 return message id into host variable 'msgid': */
```

```
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message:imsg,    /* indicator has to be specified */
msgid => :msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;

printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* Second enqueue */

strcpy(correlation2, "2nd message");
strcpy(subject, "NORMAL ENQUEUE2");
strcpy(txt, "The Enqueue was done through PLSQL embedded in PROC");

/* Initialize the components of message: */
EXEC SQL OBJECT SET subject, text OF :messsage TO :subject,:txt;

/* Embedded PLSQL call to the AQ enqueue procedure: */
EXEC SQL EXECUTE
DECLARE
message_properties   dbms_aq.message_properties_t;
enqueue_options      dbms_aq.enqueue_options_t;
msgid                RAW(16);
BEGIN
/* Bind the host variable 'correlation2':  to message correlaiton */
message_properties.correlation := :correlation2;

/* Bind the host variable 'message': to payload */
dbms_aq.enqueue(queue_name => 'msg_queue',
message_properties => message_properties,
enqueue_options => enqueue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
printf("Enqueued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* First dequeue - by  correlation */

EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Dequeue by  correlation in host variable 'correlation2': */
dequeue_options.correlation := :correlation2;

/* Return the payload into host variable 'message': */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work : */
```

```
EXEC SQL COMMIT;

/* Extract the values of the components of message: */
EXEC SQL OBJECT GET subject, text FROM :message INTO :subject,:txt;

printf("Dequeued Message \n");
printf("Subject  :%s\n",subject);
printf("Text     :%s\n",txt);

/* SECOND DEQUEUE - by MSGID  */

EXEC SQL EXECUTE
DECLARE
message_properties  dbms_aq.message_properties_t;
dequeue_options     dbms_aq.dequeue_options_t;
msgid               RAW(16);
BEGIN
/* Dequeue by msgid in host variable 'msgid': */
dequeue_options.msgid := :msgid;

/* Return the payload into host variable 'message':  */
dbms_aq.dequeue(queue_name => 'msg_queue',
message_properties => message_properties,
dequeue_options => dequeue_options,
payload => :message,
msgid => msgid);
END;
END-EXEC;
/* Commit work: */
EXEC SQL COMMIT;
}
```

Enqueuing and Dequeuing of Messages by Correlation and Message ID Using OCI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv       *envhp;
  OCIServer    *srvhp;
  OCIError     *errhp;
  OCISvcCtx    *svchp;
  dvoid        *tmp;
  OCIType      *mesg_tdo = (OCIType *) 0;
  message      msg;
  null_message nmsg;
  message      *mesg     = &msg;
  null_message *nmesg    = &nmsg;
  message      *deqmesg  = (message *)0;
  null_message *ndeqmesg = (null_message *)0;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
               52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
               52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
               52, (dvoid **) &tmp);

  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
           (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

  OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

 /* Obtain TDO of message_typ */
  OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

 /* Prepare the message payload */
  mesg->subject = (OCIString *)0;
  mesg->data = (OCIString *)0;
  OCIStringAssignText(envhp, errhp,
                    (CONST text *)"NORMAL MESSAGE", strlen("NORMAL MESSAGE"),
                    &mesg->subject);
  OCIStringAssignText(envhp, errhp,
                    (CONST text *)"OCI ENQUEUE", strlen("OCI ENQUEUE"),
                    &mesg->data);
  nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

 /* Enqueue into the msg_queue */
  OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);
  OCITransCommit(svchp, errhp, (ub4) 0);

 /* Dequeue from the msg_queue */
  OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue", 0, 0,
         mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0);
  printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
  printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Enqueuing and Dequeuing of Messages to/from a Multiconsumer Queue Using PL/SQL

```
/* Create subscriber list: */
DECLARE
   subscriber aq$_agent;
   /* Add subscribers RED and GREEN to the suscriber list: */
BEGIN
   subscriber := aq$_agent('RED', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
   subscriber => subscriber);

   subscriber := aq$_agent('GREEN', NULL, NULL);
   DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
   subscriber => subscriber);
END;

DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   recipients          DBMS_AQ.aq$_recipient_list_t;
   message_handle      RAW(16);
   message             aq.message_typ;

   /* Enqueue MESSAGE 1 for subscribers to the queue
   i.e. for RED and GREEN: */
BEGIN
   message := message_typ('MESSAGE 1',
   'This message is queued for queue subscribers.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
   enqueue_options    => enqueue_options,
   message_properties => message_properties,
   payload            => message,
   msgid              => message_handle);

   /* Enqueue MESSAGE 2 for specified recipients i.e. for RED and BLUE.*/
   message := message_typ('MESSAGE 2',
   'This message is queued for two recipients.');
   recipients(1) := aq$_agent('RED', NULL, NULL);
   recipients(2) := aq$_agent('BLUE', NULL, NULL);
   message_properties.recipient_list := recipients;

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   COMMIT;
END;
```

Note that RED is both a subscriber to the queue, as well as being a specified recipient of MESSAGE 2. By contrast, GREEN is only a subscriber to those messages in the queue (in this case, MESSAGE) for which no recipients have been specified. BLUE, while not a subscriber to the queue, is nevertheless specified to receive MESSAGE 2.

```
/* Dequeue messages from msg_queue_multiple: */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;
   no_messages         exception;
   pragma exception_init (no_messages, -25228);

BEGIN

   dequeue_options.wait := DBMS_AQ.NO_WAIT;
   BEGIN
   /* Consumer BLUE will get MESSAGE 2: */
   dequeue_options.consumer_name := 'BLUE';
   dequeue_options.navigation := FIRST_MESSAGE;

   LOOP

   DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
             dequeue_options    => dequeue_options,
             message_properties => message_properties,
             payload            => message,
             msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
            ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;

   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
   DBMS_OUTPUT.PUT_LINE ('No more messages for BLUE');
   COMMIT;
END;

BEGIN
/* Consumer RED will get MESSAGE 1 and MESSAGE 2: */
   dequeue_options.consumer_name := 'RED';
dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE
   LOOP
      DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
                dequeue_options    => dequeue_options,
                message_properties => message_properties,
                payload            => message,
                msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
                                         ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;
   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
      DBMS_OUTPUT.PUT_LINE ('No more messages for RED');
   COMMIT;
END;

BEGIN
   /* Consumer GREEN will get MESSAGE 1: */
   dequeue_options.consumer_name := 'GREEN';
   dequeue_options.navigation := FIRST_MESSAGE;
   LOOP
      DBMS_AQ.DEQUEUE(queue_name   => 'msg_queue_multiple',
                dequeue_options    => dequeue_options,
                message_properties => message_properties,
                payload            => message,
                msgid              => message_handle);

      DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
          ' ... ' || message.text );
      dequeue_options.navigation := NEXT_MESSAGE;
   END LOOP;
   EXCEPTION
   WHEN no_messages THEN
      DBMS_OUTPUT.PUT_LINE ('No more messages for GREEN');
   COMMIT;
END;
```

Enqueuing and Dequeuing of Messages to/from a Multiconsumer Queue using OCI

Note:

You may need to set up the following data structures for certain examples to work:

```
CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'aq.qtable_multi',
   multiple_consumers => true,
   queue_payload_type => 'aq.message_typ');
EXECUTE DBMS_AQADM.START_QUEUE('aq.msg_queue_multiple');
CONNECT aq/aq
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

struct message
{
  OCIString   *subject;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_subject;
  OCIInd    null_data;
};
typedef struct null_message null_message;

int main()
{
  OCIEnv                 *envhp;
  OCIServer              *srvhp;
  OCIError               *errhp;
  OCISvcCtx              *svchp;
  dvoid                  *tmp;
  OCIType                *mesg_tdo = (OCIType *) 0;
  message                msg;
  null_message           nmsg;
  message                *mesg = &msg;
  null_message           *nmesg = &nmsg;
  message                *deqmesg = (message *)0;
  null_message           *ndeqmesg = (null_message *)0;
  OCIAQMsgProperties     *msgprop = (OCIAQMsgProperties *)0;
  OCIAQAgent             *agents[2];
  OCIAQDeqOptions        *deqopt = (OCIAQDeqOptions *)0;
  ub4                    wait = OCI_DEQ_NO_WAIT;
  ub4                    navigation = OCI_DEQ_FIRST_MSG;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
              (dvoid * (*)()) 0,  (void (*)()) 0 );



  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
               52, (dvoid **) &tmp);
```

```
 OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

 OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                52, (dvoid **) &tmp);
 OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                52, (dvoid **) &tmp);

 OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

 OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                52, (dvoid **) &tmp);

 OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
            (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

 OCILogon(envhp, errhp, &svchp, "AQ", strlen("AQ"), "AQ", strlen("AQ"), 0, 0);

/* Obtain TDO of message_typ */
 OCITypeByName(envhp, errhp, svchp, (CONST text *)"AQ", strlen("AQ"),
              (CONST text *)"MESSAGE_TYP", strlen("MESSAGE_TYP"),
              (text *)0, 0, OCI_DURATION_SESSION, OCI_TYPEGET_ALL, &mesg_tdo);

/* Prepare the message payload */
 mesg->subject = (OCIString *)0;
 mesg->data = (OCIString *)0;
 OCIStringAssignText(envhp, errhp,
                     (CONST text *)"MESSAGE 1", strlen("MESSAGE 1"),
                     &mesg->subject);
 OCIStringAssignText(envhp, errhp,
                    (CONST text *)"mesg for queue subscribers",
                    strlen("mesg for queue subscribers"), &mesg->data);
 nmesg->null_adt = nmesg->null_subject = nmesg->null_data = OCI_IND_NOTNULL;

/* Enqueue MESSAGE 1 for subscribers to the queue i.e. for RED and GREEN */
 OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, 0,
          mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

/* Enqueue MESSAGE 2 for specified recipients i.e. for RED and BLUE */
/* prepare message payload */
OCIStringAssignText(envhp, errhp,
                     (CONST text *)"MESSAGE 2", strlen("MESSAGE 2"),
                     &mesg->subject);
 OCIStringAssignText(envhp, errhp,
      (CONST text *)"mesg for two recipients",
      strlen("mesg for two recipients"), &mesg->data);

/* Allocate AQ message properties and agent descriptors */
 OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
                    OCI_DTYPE_AQMSG_PROPERTIES, 0, (dvoid **)0);
 OCIDescriptorAlloc(envhp, (dvoid **)&agents[0],
                    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);
 OCIDescriptorAlloc(envhp, (dvoid **)&agents[1],
                    OCI_DTYPE_AQAGENT, 0, (dvoid **)0);

/* Prepare the recipient list, RED and BLUE */
 OCIAttrSet(agents[0], OCI_DTYPE_AQAGENT, "RED", strlen("RED"),
            OCI_ATTR_AGENT_NAME, errhp);
 OCIAttrSet(agents[1], OCI_DTYPE_AQAGENT, "BLUE", strlen("BLUE"),
            OCI_ATTR_AGENT_NAME, errhp);
 OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES, (dvoid *)agents, 2,
            OCI_ATTR_RECIPIENT_LIST, errhp);

 OCIAQEnq(svchp, errhp, (CONST text *)"msg_queue_multiple", 0, msgprop,
          mesg_tdo, (dvoid **)&mesg, (dvoid **)&nmesg, 0, 0);

 OCITransCommit(svchp, errhp, (ub4) 0);
 /* Now dequeue the messages using different consumer names */
 /* Allocate dequeue options descriptor to set the dequeue options */
  OCIDescriptorAlloc(envhp, (dvoid **)&deqopt, OCI_DTYPE_AQDEQ_OPTIONS, 0,
                     (dvoid **)0);

 /* Set wait parameter to NO_WAIT so that the dequeue returns immediately */
  OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&wait, 0,
             OCI_ATTR_WAIT, errhp);

 /* Set navigation to FIRST_MESSAGE so that the dequeue resets the position */
 /* after a new consumer_name is set in the dequeue options           */
  OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)&navigation, 0,
             OCI_ATTR_NAVIGATION, errhp);

 /* Dequeue from the msg_queue_multiple as consumer BLUE */
  OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"BLUE", strlen("BLUE"),
             OCI_ATTR_CONSUMER_NAME, errhp);

  while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
                  mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
                  == OCI_SUCCESS)
  {
    printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  }
  OCITransCommit(svchp, errhp, (ub4) 0);

 /* Dequeue from the msg_queue_multiple as consumer RED */
  OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS, (dvoid *)"RED", strlen("RED"),
        OCI_ATTR_CONSUMER_NAME, errhp);
  while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
    mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
    == OCI_SUCCESS)
  {
    printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  }
  OCITransCommit(svchp, errhp, (ub4) 0);

 /* Dequeue from the msg_queue_multiple as consumer GREEN */
  OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,(dvoid *)"GREEN",strlen("GREEN"),
        OCI_ATTR_CONSUMER_NAME, errhp);
  while (OCIAQDeq(svchp, errhp, (CONST text *)"msg_queue_multiple", deqopt, 0,
    mesg_tdo, (dvoid **)&deqmesg, (dvoid **)&ndeqmesg, 0, 0)
    == OCI_SUCCESS)
  {
    printf("Subject: %s\n", OCIStringPtr(envhp, deqmesg->subject));
    printf("Text: %s\n", OCIStringPtr(envhp, deqmesg->data));
  }
  OCITransCommit(svchp, errhp, (ub4) 0);
}
```

Enqueuing and Dequeuing of Messages Using Message Grouping Using PL/SQL

```
CONNECT aq/aq

EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table          => 'aq.msggroup',
   queue_payload_type   => 'aq.message_typ',
   message_grouping     => DBMS_AQADM.TRANSACTIONAL);

EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name      => 'msggroup_queue',
   queue_table     => 'aq.msggroup');

EXECUTE DBMS_AQADM.START_QUEUE(
   queue_name => 'msggroup_queue');

/* Enqueue three messages in each transaction */
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

BEGIN

 /* Loop through three times, committing after every iteration */
  FOR txnno in 1..3 LOOP

 /* Loop through three times, enqueuing each iteration */
    FOR mesgno in 1..3 LOOP
      message := message_typ('GROUP#' || txnno,
               'Message#' || mesgno ||  ' in group' || txnno);

      DBMS_AQ.ENQUEUE(queue_name         => 'msggroup_queue',
                  enqueue_options        => enqueue_options,
                  message_properties     => message_properties,
                  payload                => message,
                  msgid                  => message_handle);
    END LOOP;
  /* Commit the transaction */
    COMMIT;
  END LOOP;
END;

/* Now dequeue the messages as groups */
DECLARE
   dequeue_options     DBMS_AQ.dequeue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
   message             aq.message_typ;

   no_messages   exception;
   end_of_group  exception;

   PRAGMA EXCEPTION_INIT (no_messages, -25228);
   PRAGMA EXCEPTION_INIT (end_of_group, -25235);

BEGIN
   dequeue_options.wait       := DBMS_AQ.NO_WAIT;
   dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;

   LOOP
      BEGIN
      DBMS_AQ.DEQUEUE(queue_name   => 'msggroup_queue',
                dequeue_options    => dequeue_options,
                message_properties => message_properties,
                payload            => message,
                msgid              => message_handle);

     DBMS_OUTPUT.PUT_LINE ('Message: ' || message.subject ||
           ' ... ' || message.text );

     dequeue_options.navigation := DBMS_AQ.NEXT_MESSAGE;

     EXCEPTION
       WHEN end_of_group THEN
         DBMS_OUTPUT.PUT_LINE ('Finished processing a group of messages');
         COMMIT;
         dequeue_options.navigation := DBMS_AQ.NEXT_TRANSACTION;
     END;
   END LOOP;
   EXCEPTION
     WHEN no_messages THEN
       DBMS_OUTPUT.PUT_LINE ('No more messages');
END;
```

Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using PL/SQL

```
/* Create the message payload object type with one or more LOB attributes. On
 enqueue, set the LOB attribute to EMPTY_BLOB. After the enqueue completes,
 before you commit your transaction. Select the LOB attribute from the
  user_data column of the queue table or queue table view. You can now
 use the LOB interfaces (which are available through both OCI and PL/SQL) to
 write the LOB data to the queue.  On dequeue, the message payload
 will contain the LOB locator. You can use this LOB locator after
 the dequeue, but before you commit your transaction, to read the LOB data.
 */
/* Setup the accounts: */

connect system/manager

CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT CONNECT, RESOURCE TO aqadm;
GRANT aq_administrator_role TO aqadm;

CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON DBMS_AQ TO aq;
CREATE TYPE aq.message AS OBJECT(id      NUMBER,
                                 subject VARCHAR2(100),
                                 data    BLOB,
                                 trailer NUMBER);
CREATE TABLESPACE aq_tbs DATAFILE 'aq.dbs' SIZE 2M REUSE;


/* create the queue table, queues and start the queue: */

CONNECT aqadm/aqadm
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'aq.qt1',
   queue_payload_type => 'aq.message');
EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name  => 'aq.queue1',
   queue_table =>  'aq.qt1');
EXECUTE DBMS_AQADM.START_QUEUE(queue_name => 'aq.queue1');

/* End set up: */

/* Enqueue of Large data types: */

CONNECT aq/aq
CREATE OR REPLACE PROCEDURE blobenqueue(msgno IN NUMBER) AS
enq_userdata aq.message;
enq_msgid    RAW(16);
enqopt       DBMS_AQ.enqueue_options_t;
msgprop      DBMS_AQ.message_properties_t;
lob_loc      BLOB;
buffer       RAW(4096);

BEGIN
```

```
   buffer := HEXTORAW(RPAD('FF', 4096, 'FF'));
   enq_userdata := aq.message(msgno, 'Large Lob data', EMPTY_BLOB(), msgno);
   DBMS_AQ.ENQUEUE('aq.queue1', enqopt, msgprop, enq_userdata, enq_msgid);

 --select the lob locator for the queue table
   SELECT t.user_data.data INTO lob_loc
      FROM qt1 t
      WHERE t.msgid = enq_msgid;

   DBMS_LOB.WRITE(lob_loc, 2000, 1, buffer );
   COMMIT;
END;

/* Dequeue lob data: */

CREATE OR REPLACE PROCEDURE blobdequeue AS
   dequeue_options    DBMS_AQ.dequeue_options_t;
   message_properties DBMS_AQ.message_properties_t;
   mid                RAW(16);
   pload              aq.message;
   lob_loc            BLOB;
   amount             BINARY_INTEGER;
   buffer             RAW(4096);

BEGIN
   DBMS_AQ.DEQUEUE('aq.queue1', dequeue_options, message_properties,
                pload, mid);
   lob_loc := pload.data;

 -- read the lob data info buffer
   amount := 2000;
   DBMS_LOB.READ(lob_loc, amount, 1, buffer);
   DBMS_OUTPUT.PUT_LINE('Amount of data read: '||amount);
   COMMIT;
END;

/* Do the enqueues and dequeues: */
 SET SERVEROUTPUT ON

BEGIN
   FOR i IN 1..5 LOOP
      blobenqueue(i);
   END LOOP;
END;

BEGIN
   FOR i IN 1..5 LOOP
     blobdequeue();
   END LOOP;
END;
```

Enqueuing and Dequeuing Object Type Messages That Contain LOB Attributes Using Java

1. Create the message type (ADT with CLOB and blob)

```
connect aquser/aquser

create type LobMessage as object(id        NUMBER,
                              subject   varchar2(100),
                              data      blob,
                              cdata     clob,
                              trailer   number);
```

2. Create the queue table and queue

```
connect aquser/aquser
execute dbms_aqadm.create_queue_table(
          queue_table => 'qt_adt',
          queue_payload_type => 'LOBMESSAGE',
          comment => 'single-consumer, default sort ordering, ADT Message',
          compatible => '8.1.0'
        );

execute dbms_aqadm.create_queue(
          queue_name  => 'q1_adt',
          queue_table => 'qt_adt'
        );

execute dbms_aqadm.start_queue(queue_name => 'q1_adt');
```

3. Run jpublisher to generate the java class that maps to the LobMessage

```
   Oracle object type

   jpub -user=aquser/aquser -sql=LobMessage -case=mixed -methods=false
-usertypes=oracle -compatible=CustomDatum
```

4. Enqueuing and Dequeuing Messages

```
  public static void runTest(AQSession aq_sess)
  {
    Connection                db_conn   = null;
    AQEnqueueOption           eq_option = null;
    AQDequeueOption           dq_option = null;
    AQQueue                   queue1    = null;
    AQMessage                 adt_msg   = null;
    AQMessage                 adt_msg2  = null;
    AQObjectPayload           sPayload  = null;
    AQObjectPayload           sPayload2 = null;
    LobMessage                sPayl     = null;
    LobMessage                sPayl2    = null;
    AQObjectPayload           rPayload  = null;
    LobMessage                rPayl     = null;
    byte[]                    smsgid;
    AQMessage                 rMessage  = null;
    int                       i         = 0;
    int                       j         = 0;
    int                       id        = 0;
    boolean                   more      = false;
    byte[]                    b_array;
    char[]                    c_array;
    String                    mStr      = null;
    BLOB                      b1        = null;
    CLOB                      c1        = null;
    BLOB                      b2        = null;
    CLOB                      c2        = null;
    BLOB                      b3        = null;
    CLOB                      c3        = null;
    int                       b_len     = 0;
    int                       c_len     = 0;
```

```
OracleCallableStatement   blob_stmt0= null;
OracleCallableStatement   clob_stmt0= null;
OracleResultSet           rset0     = null;
OracleResultSet           rset1     = null;
OracleCallableStatement   blob_stmt = null;
OracleResultSet           rset2     = null;
OracleCallableStatement   clob_stmt = null;
OracleResultSet           rset3     = null;

try
{

 db_conn = ((AQOracleSession)aq_sess).getDBConnection();

 queue1  = aq_sess.getQueue("aquser", "q1_adt");


  b_array = new byte[5000];
  c_array = new char[5000];
  for (i = 0; i < 5000; i++)
  {
     b_array[i] = 67;
     c_array[i] = 'c';
  }
  sPayl = new LobMessage();


  System.out.println("Enqueue Long messages");

  eq_option = new AQEnqueueOption();

  /* Enqueue messages with LOB attributes */
  for ( i = 0; i < 10; i++)
  {
    adt_msg = queue1.createMessage();

    sPayload = adt_msg.getObjectPayload();

    /*  Get Empty BLOB handle */
    blob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
                  "select empty_blob() from dual");
    rset0 = (OracleResultSet) blob_stmt0.executeQuery ();
    try
    {
      if (rset0.next())
      {
        b1 = (oracle.sql.BLOB)rset0.getBlob(1);
      }
      if (b1 == null)
      {
        System.out.println("select empty_blob() from dual failed");
      }
    }
    catch (Exception ex)
    {
      System.out.println("Exception during select from dual " + ex);
      ex.printStackTrace();
    }

  /*  Get Empty CLOB handle */
    clob_stmt0 = (OracleCallableStatement)db_conn.prepareCall(
                  "select empty_clob() from dual");
    rset1 = (OracleResultSet) clob_stmt0.executeQuery ();
    try
    {
      if (rset1.next())
      {
```

```
    c1 = (oracle.sql.CLOB)rset1.getClob(1);
  }
  if (c1 == null)
  {
    System.out.println("select empty_clob() from dual failed");
  }
}
catch (Exception ex)
{
  System.out.println("Exception2 during select from dual " + ex);
  ex.printStackTrace();
}
id = i+1;
mStr = "Message #" + id;
sPayl.setId(new BigDecimal(id));
sPayl.setTrailer(new BigDecimal(id));
sPayl.setSubject(mStr);
sPayl.setData(b1);
sPayl.setCdata(c1);

/* Set Object Payload data */
sPayload.setPayloadData(sPayl);

/* Enqueue the message */
queue1.enqueue(eq_option, adt_msg);
System.out.println("Enqueued Message: " + id );
smsgid = adt_msg.getMessageId();

/*
 * Note: The message is initially enqueued with an EMPTY BLOB and CLOB
 * After enqueuing the message, we need to get the lob locators and
 * then populate the LOBs
 */
blob_stmt = (OracleCallableStatement)db_conn.prepareCall(
      "SELECT user_data FROM qt_adt where msgid = ?");
blob_stmt.setBytes(1,smsgid);
rset2 = (OracleResultSet) blob_stmt.executeQuery ();
try
{
  if (rset2.next())
  {
    /* Get message contents */
    sPayl2 = (LobMessage)rset2.getCustomDatum(1,
                    ((CustomDatumFactory)LobMessage.getFactory()));

    /* Get BLOB locator */
    b2 = sPayl2.getData();

    /* Popuate the BLOB */
    if (b2 == null)
    {
       System.out.println("Blob select null");
    }
    if ((i % 3) == 0)
    {
      b_len = b2.putBytes(1000,b_array);
    }
    else
    {
      b_len = b2.putBytes(1,b_array);
    }

    /* Get CLOB locator */
    c2 = sPayl2.getCdata();

    /* Populate the CLOB */
    if (c2 == null)
```

```
          {
             System.out.println("Clob select null");
          }
          if ((i % 4) == 0)
          {
            c_len = c2.putChars(2500,c_array);
          }
          else
          {
            c_len = c2.putChars(1,c_array);
          }
        }
      }
      catch (Exception ex)
      {
        System.out.println("Blob or Clob exception: " + ex);
      }

    }

    Thread.sleep(30000);


   // dequeue messages
   dq_option = new AQDequeueOption();
   dq_option.setWaitTime(AQDequeueOption.WAIT_NONE);


   for (i = 0 ; i < 10 ; i++)
   {
     /* Dequeue the message */
      adt_msg2 = ((AQOracleQueue)queue1).dequeue(dq_option,
                                          LobMessage.getFactory());

      /* Get payload containing LOB data */
      rPayload = adt_msg2.getObjectPayload();
      rPayl = (LobMessage) rPayload.getPayloadData();

      System.out.println("\n  Message: #" + (i+1));
      System.out.println("    Id: " + rPayl.getId());
      System.out.println("    Subject: " + rPayl.getSubject());

      /* Get BLOB data */
      b3 = rPayl.getData();
      System.out.println("    " + b3.length() + " bytes of data");

      /* Get CLOB data */
      c3 = rPayl.getCdata();
      System.out.println("    " + c3.length() + " chars of data");
      System.out.println("    Trailer: " + rPayl.getTrailer());
      db_conn.commit();
   }


  }
  catch (java.sql.SQLException sql_ex)
  {
    System.out.println("SQL Exception: " + sql_ex);
    sql_ex.printStackTrace();
  }
  catch (Exception ex)
  {
    System.out.println("Exception-2: " + ex);
    ex.printStackTrace();
  }

}
```

Propagation

> **Caution:**
>
> You may need to create queues or queue tables, or start or enable queues, for certain examples to work:

Enqueue of Messages for remote subscribers/recipients to a Multiconsumer Queue and Propagation Scheduling Using PL/SQL

```
/* Create subscriber list: */
DECLARE
   subscriber aq$_agent;

   /* Add subscribers RED and GREEN with different addresses  to the suscriber
   list: */
BEGIN
   BEGIN
      /* Add subscriber RED that will dequeue messages from another_msg_queue
 queue in the same datatbase */
      subscriber := aq$_agent('RED', 'another_msg_queue', NULL);
      DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
      subscriber => subscriber);

      /* Schedule propagation from msg_queue_multiple to other queues in the
 same
      database: */
      DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple');

      /* Add subscriber GREEN that will dequeue messages from the msg_queue
 queue
      in another database reached by the database link another_db.world */
      subscriber := aq$_agent('GREEN', 'msg_queue@another_db.world', NULL);
      DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'msg_queue_multiple',
      subscriber => subscriber);

      /* Schedule propagation from msg_queue_multiple to other queues in the
      database "another_database": */
   END;
   BEGIN
      DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'msg_queue_multiple',
      destination  => 'another_db.world');
   END;
END;

DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   recipients          DBMS_AQ.aq$_recipient_list_t;
   message_handle      RAW(16);
   message             aq.message_typ;

/* Enqueue MESSAGE 1 for subscribers to the queue
i.e. for RED at address another_msg_queue and GREEN at address msg_
queue@another_db.world: */
BEGIN
   message := message_typ('MESSAGE 1',
   'This message is queued for queue subscribers.');

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
        message_properties => message_properties,
        payload            => message,
        msgid              => message_handle);

   /* Enqueue MESSAGE 2 for specified recipients i.e. for RED at address
   another_msg_queue and BLUE.*/
   message := message_typ('MESSAGE 2',
   'This message is queued for two recipients.');
   recipients(1) := aq$_agent('RED', 'another_msg_queue', NULL);
   recipients(2) := aq$_agent('BLUE', NULL, NULL);
   message_properties.recipient_list := recipients;

   DBMS_AQ.ENQUEUE(queue_name => 'msg_queue_multiple',
           enqueue_options    => enqueue_options,
           message_properties => message_properties,
           payload            => message,
           msgid              => message_handle);

   COMMIT;
END;
```

Note:

RED at address another_msg_queue is both a subscriber to the queue, as well as being a specified recipient of MESSAGE 2. By contrast, GREEN at address msg_queue@another_db.world is only a subscriber to those messages in the queue (in this case, MESSAGE 1) for which no recipients have been specified. BLUE, while not a subscriber to the queue, is nevertheless specified to receive MESSAGE 2.

Managing Propagation From One Queue To Other Queues In The Same Database Using PL/SQL

```
/* Schedule propagation from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'q1def');

/* Disable propagation from queue q1def to other queues in the same
database */
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   queue_name => 'q1def');

/* Alter schedule from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name => 'q1def',
   duration   => '2000',
   next_time  => 'SYSDATE + 3600/86400',
   latency    => '32');

/* Enable propagation from queue q1def to other queues in the same database */
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   queue_name => 'q1def');

/* Unschedule propagation from queue q1def to other queues in the same database
*/
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name => 'q1def');
```

Manage Propagation From One Queue To Other Queues In Another Database Using PL/SQL

```
/* Schedule propagation from queue q1def to other queues in another  database
reached by the database link another_db.world  */
EXECUTE DBMS_AQADM.SCHEDULE_PROPAGATION(
   queue_name   => 'q1def',
   destination  => 'another_db.world');

/* Disable propagation from queue q1def to other queues in another database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world');

/* Alter schedule from queue q1def to other queues in another database  reached
by the database link another_db.world  */
EXECUTE DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world',
   duration    => '2000',
   next_time   => 'SYSDATE +  3600/86400',
   latency     => '32');

/* Enable propagation from queue q1def to other queues in another  database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
   queue_name  => 'q1def',
   destination => 'another_db.world');

/* Unschedule propagation from queue q1def to other queues in another  database
reached by the database link another_db.world */
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name  => 'q1def',
   destination => 'another_db.world');
```

Unscheduling Propagation Using PL/SQL

```
/* Unschedule propagation from msg_queue_multiple to the destination another_
db.world */
EXECUTE DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name => 'msg_queue_multiple',
   destination => 'another_db.world');
```

For additional examples of Alter Propagation, Enable Propagation and Disable Propagation, see:

- "Example: Alter a Propagation Schedule Using PL/SQL (DBMS_AQADM)" on page 9-76
- "Example: Enable a Propagation Using PL/SQL (DBMS_AQADM)" on page 9-79
- "Example: Disable a Propagation Using PL/SQL (DBMS_AQADM)" on page 82

Dropping AQ Objects

Caution:

You may need to create queues or queue tables, or start, stop, or enable queues, for certain examples to work:

```
/* Cleans up all objects related to the object type: */
CONNECT aq/aq

EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name => 'msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.objmsgs80_qtab');

/* Cleans up all objects related to the RAW type: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name      => 'raw_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.RawMsgs_qtab');

/* Cleans up all objects related to the priority queue: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name     => 'priority_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name     => 'priority_msg_queue');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table   => 'aq.priority_msg');

/* Cleans up all objects related to the multiple-consumer queue: */
EXECUTE DBMS_AQADM.STOP_QUEUE (
   queue_name  => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE (
   queue_name  => 'msg_queue_multiple');

EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table => 'aq.MultiConsumerMsgs_qtab');

DROP TYPE aq.message_typ;
```

Revoking Roles and Privileges

```
CONNECT sys/change_on_install
DROP USER aq;
```

Deploying AQ with XA

Note:

You may need to set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER aqadm CASCADE;
```

```
GRANT CONNECT, RESOURCE TO aqadm;
CREATE USER aqadm IDENTIFIED BY aqadm;
GRANT EXECUTE ON DBMS_AQADM TO aqadm;
GRANT Aq_administrator_role TO aqadm;
DROP USER aq CASCADE;
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table => 'aq.qtable',
   queue_payload_type => 'RAW');

EXECUTE DBMS_AQADM.CREATE_QUEUE(
   queue_name => 'aq.aqsqueue',
   queue_table => 'aq.qtable');

EXECUTE DBMS_AQADM.START_QUEUE(queue_name =>
'aq.aqsqueue');
```

```
/*
 * The program uses the XA interface to enqueue 100 messages and then
 * dequeue them.
 * Login: aq/aq
 * Requires: AQ_USER_ROLE to be granted to aq
 *        a RAW queue called "aqsqueue" to be created in aqs schema
 *        (above steps can be performed by running aqaq.sql)
 * Message Format: Msgno: [0-1000] HELLO, WORLD!
 * Author: schandra@us.oracle.com
 */

#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <xa.h>

/* XA open string */
char xaoinfo[] =  "oracle_xa+ACC=P/AQ/AQ+SESTM=30+Objects=T";

/* template for generating XA XIDs */
XID  xidtempl = { 0x1e0a0a1e, 12, 8, "GTRID001BQual001" };

/* Pointer to Oracle XA function table */
extern struct xa_switch_t xaosw;                        /* Oracle XA switch */
static struct xa_switch_t *xafunc = &xaosw;

/* dummy stubs for ax_reg and ax_unreg */
int ax_reg(rmid, xid, flags)
int  rmid;
XID *xid;
long flags;
{
  xid->formatID = -1;
  return 0;
}

int ax_unreg(rmid, flags)
int     rmid;
long    flags;
{
  return 0;
}

/* generate an XID */
```

```
void xidgen(xid, serialno)
XID *xid;
int  serialno;
{
  char seq [11];

  sprintf(seq, "%d", serialno);
  memcpy((void *)xid, (void *)&xidtempl, sizeof(XID));
  strncpy((&xid->data[5]), seq, 3);
}

/* check if XA operation succeeded */
#define checkXAerr(action, funcname)    \
    if ((action) != XA_OK)        \
    {                  \
      printf("%s failed!\n", funcname);   \
      exit(-1);            \
    } else

/* check if OCI operation succeeded */
static void checkOCIerr(errhp, status)
LNOCIError *errhp;
sword      status;
{
  text errbuf[512];
  ub4 buflen;
  sb4 errcode;

  if (status == OCI_SUCCESS) return;

  if (status == OCI_ERROR)
  {
    OCIErrorGet((dvoid *) errhp, 1, (text *)0, &errcode, errbuf,
      (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
  }
  else
    printf("Error - %d\n", status);
  exit (-1);
}


void main(argc, argv)
int    argc;
char **argv;
{
  int         msgno = 0;            /* message being enqueued */
  OCIEnv       *envhp;               /* OCI environment handle */
  OCIError     *errhp;               /* OCI Error handle */
  OCISvcCtx    *svchp;               /* OCI Service handle */
  char      message[128];               /* message buffer */
  ub4      mesglen;             /* length of message */
  OCIRaw       *rawmesg = (OCIRaw *)0;     /* message in OCI RAW format */
  OCIInd      ind = 0;               /* OCI null indicator */
  dvoid          *indptr = (dvoid *)&ind;      /* null indicator pointer */
  OCIType      *mesg_tdo = (OCIType *) 0;        /* TDO for RAW datatype */
  XID      xid;              /* XA's global transaction id */
  ub4      i;                   /* array index */

  checkXAerr(xafunc->xa_open_entry(xaoinfo, 1, TMNOFLAGS), "xaoopen");

  svchp = xaoSvcCtx((text *)0);           /* get service handle from XA */
  envhp = xaoEnv((text *)0);          /* get enviornment handle from XA */

  if (!svchp || !envhp)
  {
    printf("Unable to obtain OCI Handles from XA!\n");
    exit (-1);
  }

  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&errhp,
       OCI_HTYPE_ERROR, 0, (dvoid **)0);  /* allocate error handle */

 /* enqueue 1000 messages, 1 message per XA transaction */
  for (msgno = 0; msgno < 1000; msgno++)
  {
    sprintf((const char *)message, "Msgno: %d, Hello, World!", msgno);
    mesglen = (ub4)strlen((const char *)message);
    xidgen(&xid, msgno);                 /* generate an XA xid */

    checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");

    checkOCIerr(errhp, OCIRawAssignBytes(envhp, errhp, (ub1 *)message, mesglen,
                &rawmesg));

    if (!mesg_tdo)          /* get Type descriptor (TDO) for RAW type */
      checkOCIerr(errhp, OCITypeByName(envhp, errhp, svchp,
                      (CONST text *)"AQADM", strlen("AQADM"),
                       (CONST text *)"RAW", strlen("RAW"),
                   (text *)0, 0, OCI_DURATION_SESSION,
                   OCI_TYPEGET_ALL, &mesg_tdo));

    checkOCIerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"aqsqueue",
               0, 0, mesg_tdo, (dvoid **)&rawmesg, &indptr,
            0, 0));

    checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaoend");
    checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
    printf("%s Enqueued\n", message);
  }

  /* dequeue 1000 messages within one XA transaction */
  xidgen(&xid, msgno);                           /* generate an XA xid */
  checkXAerr(xafunc->xa_start_entry(&xid, 1, TMNOFLAGS), "xaostart");
  for (msgno = 0; msgno < 1000; msgno++)
  {
    checkOCIerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"aqsqueue",
             0, 0, mesg_tdo, (dvoid **)&rawmesg, &indptr,
            0, 0));
    if (ind)
      printf("Null Raw Message");
    else
      for (i = 0; i < OCIRawSize(envhp, rawmesg); i++)
   printf("%c", *(OCIRawPtr(envhp, rawmesg) + i));
    printf("\n");

  }
  checkXAerr(xafunc->xa_end_entry(&xid, 1, TMSUCCESS), "xaoend");
  checkXAerr(xafunc->xa_commit_entry(&xid, 1, TMONEPHASE), "xaocommit");
}
```

AQ and Memory Usage

Create_types.sql: Create Payload Types and Queues in Scott's Schema

Note:

You may need to set up data structures for certain examples to work, such as:

```
/* Create_types.sql */
```

```
CONNECT system/manager
GRANT AQ_ADMINISTRATOR_ROLE, AQ_USER_ROLE TO scott;
CONNECT scott/tiger
CREATE TYPE MESSAGE AS OBJECT (id NUMBER, data VARCHAR2(80));
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE(
   queue_table        => 'qt',
   queue_payload_type => 'message');
EXECUTE DBMS_AQADM.CREATE_QUEUE('msgqueue', 'qt');
EXECUTE DBMS_AQADM.START_QUEUE('msgqueue');
```

Enqueuing Messages (Free Memory After Every Call) Using OCI

This program, `enqnoreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in scott's schema using *create_types.sql*. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c` (see below). If there are no messages, it waits for 60 seconds before timing out. In this program, the dequeue subroutine does not reuse client side objects' memory. It allocates the required memory before dequeue and frees it after the dequeue is complete.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

LNOCIEnv       *envhp;
LNOCIError     *errhp;
LNOCISvcCtx    *svchp;

struct message
{
  OCINumber    id;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text  *buf;
ub4   *buflen;
{
  OCIType          *mesgtdo = (OCIType *)0;   /* type descr of SCOTT.MESSAGE */
  message          *mesg    = (dvoid *)0;    /* instance of SCOTT.MESSAGE */
  null_message     *mesgind = (dvoid *)0;    /* null indicator */
  OCIAQDeqOptions *deqopt  = (OCIAQDeqOptions *)0;
  ub4               wait    = 60;            /* timeout after 60 seconds */
  ub4               navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */


 /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

 /* Allocate an instance of SCOTT.MESSAGE, and get its null indicator: */
  checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
           mesgtdo, (dvoid *)0, OCI_DURATION_SESSION,
           TRUE, (dvoid **)&mesg));
  checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *)mesg,
           (dvoid **)&mesgind));

 /* Allocate a descriptor for dequeue options and set wait time, navigation: */
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
           OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp));

 /* Dequeue the message and commit: */
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
           deqopt, 0, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

 /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
           (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));

 /* Free the dequeue options descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

 /* Free the memory for the objects: */
  Checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *)mesg,
           OCI_OBJECTFREE_FORCE));
}                           /* end deqmesg */

void main()
{
  OCIServer     *srvhp;
  OCISession    *usrhp;
  dvoid         *tmp;
  text           buf[80];                /* payload text */
  ub4            buflen;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

 /* Set attribute server context in the service context: */
```

```
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
             (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

 /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);
}                      /* end main */

static void checkerr(errhp, status)
LNOCIError  *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4 errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                         /* end checkerr */
```

Enqueuing Messages (Reuse Memory) Using OCI

This program, `enqreuse.c`, enqueues each line of text into a queue 'msgqueue' that has been created in scott's schema by executing `create_types.sql`. Each line of text entered by the user is stored in the queue until user enters `EOF`. In this program the enqueue subroutine reuses the memory for the message payload, as well as the AQ message properties descriptor.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>
```

```
static void checkerr(OCIError *errhp, sword status);
static void enqmesg(ub4 msgno, text *buf);

struct message
{
  OCINumber     id;
  OCIString    *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

/* Global data reused on calls to enqueue: */
LNOCIEnv             *envhp;
LNOCIError           *errhp;
LNOCISvcCtx          *svchp;
message             msg;
null_message        nmsg;
LNOCIAQMsgProperties *msgprop;

static void enqmesg(msgno, buf)
ub4     msgno;
text   *buf;
{
  OCIType          *mesgtdo = (OCIType *)0; /* type descr of SCOTT.MESSAGE */
  message          *mesg = &msg;           /* instance of SCOTT.MESSAGE */
  null_message     *mesgind = &nmsg;       /* null indicator */
  text              corrid[128];           /* correlation identifier */

 /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

 /* Fill in the attributes of SCOTT.MESSAGE: */
  checkerr(errhp, OCINumberFromInt(errhp, &msgno, sizeof(ub4), 0, &mesg->id));
  checkerr(errhp, OCIStringAssignText(envhp, errhp, buf, strlen(buf),
           &mesg->data));
  mesgind->null_adt = mesgind->null_id = mesgind->null_data = 0;

 /* Set the correlation id in the message properties descriptor: */
  sprintf((char *)corrid, "Msg#: %d", msgno);
  checkerr(errhp, OCIAttrSet(msgprop, OCI_DTYPE_AQMSG_PROPERTIES,
           (dvoid *)&corrid, strlen(corrid),
           OCI_ATTR_CORRELATION, errhp));

 /* Enqueue the message and commit: */
  checkerr(errhp, OCIAQEnq(svchp, errhp, (CONST text *)"msgqueue",
           0, msgprop, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));
}                     /* end enqmesg */

void main()
{
  OCIServer    *srvhp;
  OCISession   *usrhp;
  dvoid        *tmp;
  text          buf[80];             /* user supplied text */
  int           msgno = 0;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

 /* Set attribute server context in the service context: */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
             (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

 /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

 /* Allocate a message properties descriptor to fill in correlation id :*/
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&msgprop,
           OCI_DTYPE_AQMSG_PROPERTIES,
           0, (dvoid **)0));
  do {
    printf("Enter a line of text (max 80 chars):");
    if (!gets((char *)buf))
      break;
    enqmesg((ub4)msgno++, buf);
  } while(1);

 /* Free the message properties descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)msgprop,
           OCI_DTYPE_AQMSG_PROPERTIES));

}                        /* end main */

static void checkerr(errhp, status)
LNOCIError   *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;
  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                      /* end checkerr */
```

Dequeuing Messages (Free Memory After Every Call) Using OCI

This program, `deqnoreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in scott's schema by executing *`create_types.sql`*. Messages are enqueued using `enqnoreuse` or `enqreuse`. If there are no messages, it waits for 60 seconds before timing out. In this program the dequeue subroutine does not reuse client side objects' memory. It allocates the required memory before dequeue and frees it after the dequeue is complete.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

LNOCIEnv       *envhp;
LNOCIError     *errhp;
LNOCISvcCtx    *svchp;

struct message
{
  OCINumber     id;
  OCIString    *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

static void deqmesg(buf, buflen)
text      *buf;
ub4       *buflen;
{
  OCIType          *mesgtdo = (OCIType *)0;  /* type descr of SCOTT.MESSAGE */
  message          *mesg = (dvoid *)0;       /* instance of SCOTT.MESSAGE */
  null_message     *mesgind   = (dvoid *)0;      /* null indicator */
  OCIAQDeqOptions  *deqopt    = (OCIAQDeqOptions *)0;
  ub4              wait      = 60;             /* timeout after 60 seconds */
  ub4             navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */

 /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

 /* Allocate an instance of SCOTT.MESSAGE, and get its null indicator: */
  checkerr(errhp, OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT,
           mesgtdo, (dvoid *)0, OCI_DURATION_SESSION,
           TRUE, (dvoid **)&mesg));
  checkerr(errhp, OCIObjectGetInd(envhp, errhp, (dvoid *)mesg,
           (dvoid **)&mesgind));

 /* Allocate a descriptor for dequeue options and set wait time, navigation: */
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
           OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp));

 /* Dequeue the message and commit: */
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
           deqopt, 0, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

 /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
           (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));

 /* Free the dequeue options descriptor: */
  checkerr(errhp, OCIDescriptorFree((dvoid *)deqopt, OCI_DTYPE_AQDEQ_OPTIONS));

 /* Free the memory for the objects: */
  checkerr(errhp, OCIObjectFree(envhp, errhp, (dvoid *)mesg,
           OCI_OBJECTFREE_FORCE));
}                                      /* end deqmesg */

void main()
{
  OCIServer    *srvhp;
  OCISession   *usrhp;
  dvoid        *tmp;
  text          buf[80];                /* payload text */
  ub4           buflen;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

 /* Set attribute server context in the service context: */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
             (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

 /* Allocate a user context handle: */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);
}                        /* end main */

static void checkerr(errhp, status)
LNOCIError   *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                         /* end checkerr */
```

Dequeuing Messages (Reuse Memory) Using OCI

This program, `deqreuse.c`, dequeues each line of text from a queue '`msgqueue`' that has been created in scott's schema by executing `create_types.sql`. Messages are enqueued using `enqnoreuse.c` or `enqreuse.c`. If there are no messages, it waits for 60 seconds before timing out. In this program, the dequeue subroutine reuses client side objects' memory between invocation of `LNOCIAQDeq`. During the first call to `LNOCIAQDeq`, OCI automatically allocates the memory for the message payload. During subsequent calls to `LNOCIAQDeq`, the same payload pointers are passed and OCI will automatically resize the payload memory if necessary.

```
#ifndef OCI_ORACLE
#include <oci.h>
#endif

#include <stdio.h>

static void checkerr(OCIError *errhp, sword status);
static void deqmesg(text *buf, ub4 *buflen);

struct message
{
  OCINumber    id;
  OCIString   *data;
};
typedef struct message message;

struct null_message
{
  OCIInd    null_adt;
  OCIInd    null_id;
  OCIInd    null_data;
};
typedef struct null_message null_message;

/* Global data reused on calls to enqueue: */
LNOCIEnv          *envhp;
LNOCIError        *errhp;
LNOCISvcCtx       *svchp;
LNOCIAQDeqOptions *deqopt;
message         *mesg = (message *)0;
null_message    *mesgind = (null_message *)0;

static void deqmesg(buf, buflen)
text       *buf;
ub4        *buflen;
{

  OCIType        *mesgtdo    = (OCIType *)0; /* type descr of SCOTT.MESSAGE */
  ub4             wait       = 60;         /* timeout after 60 seconds */
  ub4             navigation = OCI_DEQ_FIRST_MSG;/* always get head of q */

 /* Get the type descriptor object for the type SCOTT.MESSAGE: */
  checkerr(errhp, OCITypeByName(envhp, errhp, svchp,
           (CONST text *)"SCOTT", strlen("SCOTT"),
           (CONST text *)"MESSAGE", strlen("MESSAGE"),
           (text *)0, 0, OCI_DURATION_SESSION,
           OCI_TYPEGET_ALL, &mesgtdo));

 /* Set wait time, navigation in dequeue options: */
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&wait, 0, OCI_ATTR_WAIT, errhp));
  checkerr(errhp, OCIAttrSet(deqopt, OCI_DTYPE_AQDEQ_OPTIONS,
           (dvoid *)&navigation, 0,
           OCI_ATTR_NAVIGATION, errhp));

 /*
 * Dequeue the message and commit. The memory for the payload will be
 * automatically allocated/resized by OCI:
 */
```

```
  checkerr(errhp, OCIAQDeq(svchp, errhp, (CONST text *)"msgqueue",
           deqopt, 0, mesgtdo, (dvoid **)&mesg,
           (dvoid **)&mesgind, 0, 0));

  checkerr(errhp, OCITransCommit(svchp, errhp, (ub4) 0));

 /* Copy the message payload text into the user buffer: */
  if (mesgind->null_data)
    *buflen = 0;
  else
    memcpy((dvoid *)buf, (dvoid *)OCIStringPtr(envhp, mesg->data),
           (size_t)(*buflen = OCIStringSize(envhp, mesg->data)));
}                           /* end deqmesg */

void main()
{
  OCIServer    *srvhp;
  OCISession   *usrhp;
  dvoid        *tmp;
  text          buf[80];               /* payload text */
  ub4           buflen;

  OCIInitialize((ub4) OCI_OBJECT, (dvoid *)0,  (dvoid * (*)()) 0,
                (dvoid * (*)()) 0,  (void (*)()) 0 );

  OCIHandleAlloc((dvoid *) NULL, (dvoid **) &envhp, (ub4) OCI_HTYPE_ENV,
                 52, (dvoid **) &tmp);

  OCIEnvInit( &envhp, (ub4) OCI_DEFAULT, 21, (dvoid **) &tmp  );

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, (ub4) OCI_HTYPE_ERROR,
                 52, (dvoid **) &tmp);
  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, (ub4) OCI_HTYPE_SERVER,
                 52, (dvoid **) &tmp);

  OCIServerAttach(srvhp, errhp, (text *) 0, (sb4) 0, (ub4) OCI_DEFAULT);

  OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                 52, (dvoid **) &tmp);

 /* set attribute server context in the service context */
  OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp, (ub4) 0,
             (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

 /* allocate a user context handle */
  OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                 (size_t) 0, (dvoid **) 0);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"scott", (ub4)strlen("scott"), OCI_ATTR_USERNAME, errhp);

  OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
             (dvoid *)"tiger", (ub4)strlen("tiger"), OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
           OCI_DEFAULT));

  OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
             (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

 /* allocate the dequeue options descriptor */
  checkerr(errhp, OCIDescriptorAlloc(envhp, (dvoid **)&deqopt,
           OCI_DTYPE_AQDEQ_OPTIONS, 0, (dvoid **)0));

  do {
    deqmesg(buf, &buflen);
    printf("%.*s\n", buflen, buf);
  } while(1);

 /*
 * This program never reaches this point as the dequeue timesout & exits.
 * If it does reach here, it will be a good place to free the dequeue
 * options descriptor using OCIDescriptorFree and free the memory allocated
 * by OCI for the payload using OCIObjectFree
 */
}                       /* end main */

static void checkerr(errhp, status)
LNOCIError *errhp;
sword      status;
{
  text errbuf[512];
  ub4  buflen;
  sb4  errcode;

  if (status == OCI_SUCCESS) return;

  switch (status)
  {
  case OCI_ERROR:
    OCIErrorGet ((dvoid *) errhp, (ub4) 1, (text *) NULL, &errcode,
                 errbuf, (ub4) sizeof(errbuf), (ub4) OCI_HTYPE_ERROR);
    printf("Error - %s\n", errbuf);
    break;
  case OCI_INVALID_HANDLE:
    printf("Error - OCI_INVALID_HANDLE\n");
    break;
  default:
    printf("Error - %d\n", status);
    break;
  }
  exit(-1);
}                        /* end checkerr */
```

Previous Next

Home Book List Contents Index Master Index Feedback