Architecture : Oracle AQ with Generic JMS RA (Glassfish)

This page last changed on Mar 07, 2008 by shailesh.kini.

Introduction

This document outlines how to integrate Glassfish with Oracle Streams AQ (10g). It provides a brief introduction regarding the integration options available with glassfish and supported by Oracle jms client implementation. This document also explains the setup requirement on Oracle as well as glassfish for the integration.

Oracle Streams AQ is our MoM

Message Oriented Middleware(MoM) are widely used in Enterprise systems. They are great for processing asynchronous tasks. In this section we will look at Oracle Streams AQ as an enterprise messaging system provider.

Oracle Streams AQ provides a technology independent messaging system. Technology independence enables it to support producers/publishers and consumers/subscribers written in disparate technologies. For example we can write a producer/publisher in .NET or Ruby and the consumer/subscriber in Java, pl/sql or C. All producers/publishers and consumers/subscribers use some sort of api directly or through a generic interface to communicate with Oracle Streams AQ messaging system.

Java provides JMS as a standard api/interface to communicate with the messaging system. All jms compliant messaging systems provide an implementation for the JMS api. The JMS client will use the standard JMS API to send or receive messages. The implementation which is transparent to the JMS client translates the jms method calls to provider specific instructions.

Oracle Streams AQ provides a good messaging system solution integrated with the database. It also gives us the additional flexibility to use pl/sql block as one of the producers. This lets the existing systems offload some of the asynchronous tasks. Other systems can run in parallel and process these offloaded tasks. Oracle Streams AQ supports the latest JMS 1.1 specification.

Connecting Glassfish to a JMS Provider

Introduction

This document tries to explain the approach used by Glassfish application server to connect to a JMS Provider.

Resource Adapters

Java connector architecture was introduced to be able to connect the application server to EAI and JMS Providers. This allowed a standard way for communication between the application server and legacy

systems. Application server would call standard methods on the resource adapter and the resource adapter would know how to translate the standard calls to certain provider specific calls.

Genericjmsra was an attempt to create a resource adapter that would support a standard way to communicate with majority of JMS Providers.
Two approaches were selected for integrating resource adapter with JMS providers

1. Using JavaBean: sets and accesses the properties for the factory through getter and setter methods
2. Using JNDI: uses jndi to lookup directory services for factories

Both these approaches deal with how the resource adapter will get a handle to any one of these factories: ConnectionFactory, QueueConnectionFactory or TopicConnectionFactory or their XA counterparts. Most providers have added support for one or both of these approaches.

Oracle JMS Support

Oracle as of 10g is JMS 1.1 compliant. It support all of JMS 1.1 required features and also some optional features. For glassfish application server to be able to integrate with Oracle AQ jms Provider, it has to your a resource adapter. To use genericjmsra resource adapter Oracle should support atleast one of the integration approaches discussed above.
Oracle JMS client does not allow creation of ConnectionFactory, QueueConnectionFactory or TopicConnectionFactory utilizing JavaBean approach. The factory creation is only possible through AQjmsFactory class provided in the Oracle jms client api. However fortunately, Oracle does support the JNDI lookup approach. We will be focusing on the JNDI approach for Oracle AQ and glassfish integration.

Oracle LDAP server configuration

Introduction

This document assumes that Oralce LDAP server(OID) has already been created and the database instance containing queues has been registered with Oracle LDAP server. Please check references for help in setting up Oracle LDAP server.

Requirements

1. Oracle LDAP server url is required and should be available with your DBA.
2. Oracle LDAP server port. For quick and easy setup use of non SSL port is recommended.
3. DBA should create an LDAP user. The LDAP user should be granted GLOBAL_AQ_USER_ROLE.

Verification

After all of the above requirements have been satisfied, we should test the LDAP connection. Please use JXPlorer to connect to the LDAP server.

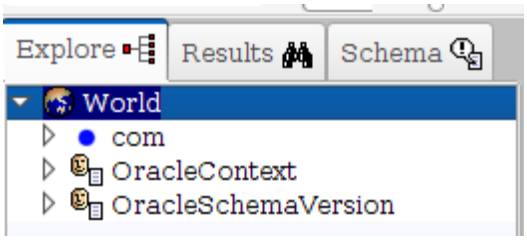
To connect to Oracle LDAP browser please follow the instruction below.

1. Open JXPlorer LDAP browser by executing jxplorer.sh or jxplorer.bat.
2. In Open LDAP/DSML Connection modal dialog enter the following connection parameters
    a. Host: Oracle LDAP server name.
    b. Port: Port where Oracle LDAP server is listening to. default 389
    c. Protocol: Select the default LDAP v3 if your OID support v3 or pick LDAP v2.
    d. DSML Service: Leave it blank
    e. Base DN: If you are using a base distinguished name please add it to this field or else leave it blank.
    f. Security -> Level: Check with your DBA. I use User + Password option (no ssl). Please check with your DBA how OID is setup to accept connection requests.
    g. Security -> User DN: add the LDAP user dn. example cn=userid.
    h. Security -> Password: add the password for the LDAP user. You should be able to get the password from your DBA.

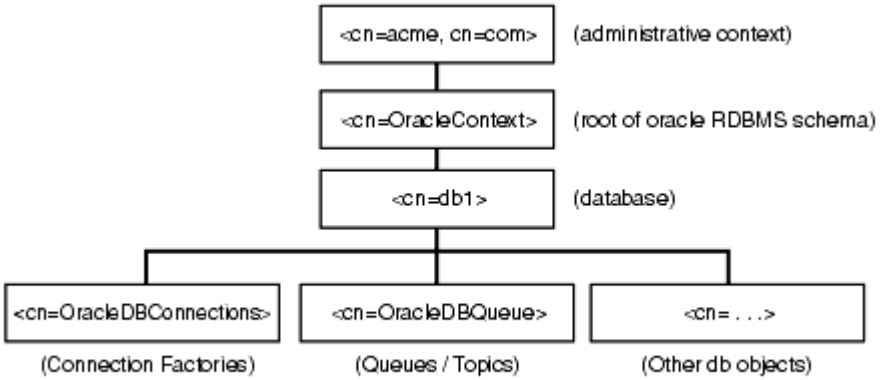
Click on OK to connect. If you get any exceptions or errors you might want to refer to JXPlorer documentation to enable tracing.

Once the LDAP browser successfully connects to Oracle LDAP server it should display all the directory entries visible to this LDAP user.

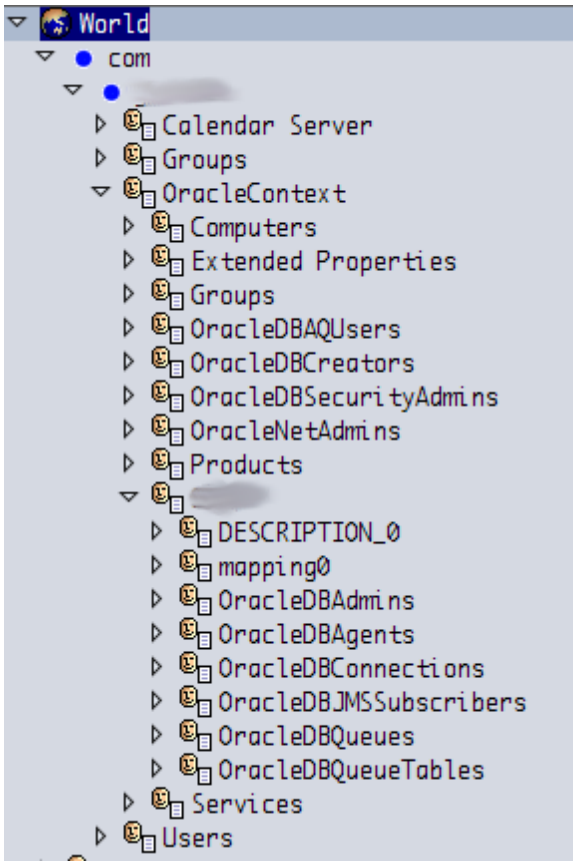
The screen shot below shows the directory structure of Oracle LDAP server.



OracleContext is the parent for all Oracle database registered. Expanding OracleContext node displays all the database servers registered with Oracle LDAP. We are using Oracle LDAP server version 10.x.



Your database server entry should look something like the image below.



You could use stored procedures in Oracle to create queue connections or you could do it through java code. Assuming Oracle LDAP and Oracle Database communicate successfully, all your queues should be added automatically on the OID server once you create them in the database. All the queues should be visible under OracleDBQueues and the queue tables should be visible under OracleDBQueueTables. The connections should be under OracleDBConnections.

You may manually create entries under OracleDBQueues and OracleDBQueueTables for queues and queue tables if they do not get automatically created, however, it is a very involved process.

References

- Step by Step Guide To Troubleshooting
- Example of Setting Up Enterprise User Security
- Oracle LDAP Server setup
- JXPlorer -> LDAP browser

Registering Connection Factories and Queues with OID

Introduction

This article will show how to register Connection Factories and Queues with OID server. We will be using Oracle JMS client api for registration.

Requirements

1. Java Standard Edition 5.0 from sun microsystems is preferred. Some changes are required if you plan to use the JDK from another vendor.
2. Oracle jdbc driver. We are using version 10.x with JDK5.
3. Oracle JMS client jars are bundled with Oracle Client 10.x. Download and install Oracle Client 10.x. Locate aqapi.jar, aqxml.jar and jmscommon.jar files.
4. Download and install netbeans or eclipse IDE. Netbeans is used in this article.

Registering Connection Factory.

Oracle JMS client api can be found at the link here

The most import class in the api used to register factories with the Oracle LDAP server (OID) is oracle.jms.AQjmsFactory. Alternately you could manually register the factory with any LDAP brower. Registering the factory with Oracle JMS client is very easy and is used in this case.

```
package oraclejmsadmin;

import java.util.Hashtable;
import javax.jms.ConnectionMetaData;
import javax.jms.JMSException;
import javax.jms.XAQueueConnection;
import javax.jms.XAQueueConnectionFactory;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import oracle.jms.AQjmsFactory;
import static java.lang.System.out;

public class RegisterFactory {

    Hashtable env = null;
    boolean envSet = false;

    public void setEnv() {
        env = new Hashtable(5, 0.75f);
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://<oid_server>:<oid_port>");
        env.put("searchbase", "cn=<oid_db_entry>, cn=OracleContext");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL, "cn=<oid_user>");
        env.put(Context.SECURITY_CREDENTIALS, "<oid_password>");
        //recommended in oracle metalink knowledgebase.
        env.put("server_dn", "cn=<oid_db_entry>,cn=OracleContext");
        envSet = true;
    }

    public void register() {
        if (!envSet) {
            setEnv();
        } else {
            try {
                String url = "jdbc:oracle:thin:@<db_server_name>:<db_port>:<db_SID>";
                int returnCode = AQjmsFactory.registerConnectionFactory(env,
"<connection_factory_name>", url, null, "queue");
                out.println("Return code from registration is " + returnCode);
            } catch (JMSException jmsEx) {
                jmsEx.printStackTrace();
            }
        }
    }

}
```

Main.java creates and instance of RegisterFactory.

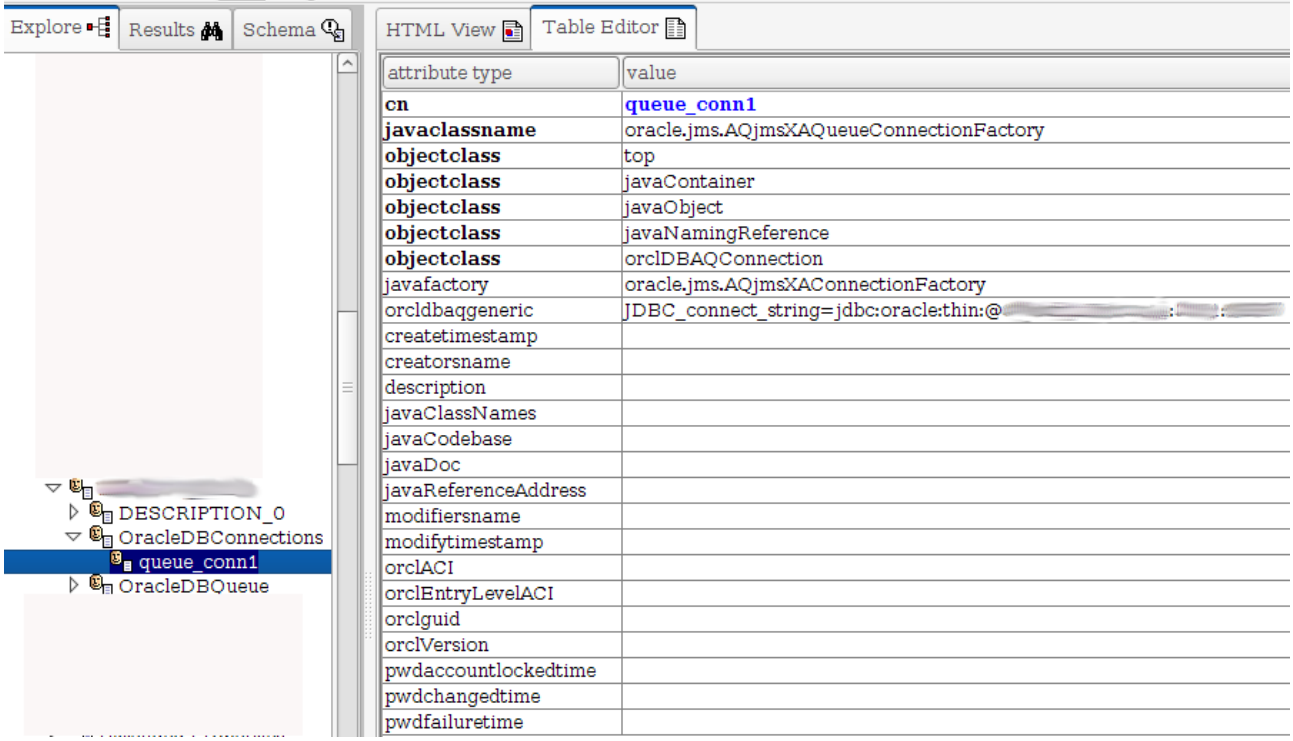
```
package oraclejmsadmin;

public class Main {
    public static void main(String[] args) {
        RegisterFactory rf = new RegisterFactory();
        rf.setEnv();
        rf.register();
        rf = null;
    }
}
```

After running this example code the output is as follows

Output

```
init:
deps-jar:
compile-single:
run-single:
Return code from registration is 0
BUILD SUCCESSFUL (total time: 1 second)
```

Once this code is run the connection factory should be visible under OracleDBConnections node of the OID. Use JXPlorer to confirm this.



By default the connection factory does not support distributed transactions. The javaclassname property is set to oracle.jms.AQjmsQueueConnectionFactory and javafactory is set to oracle.jms.AQjmsConnectionFactory. To enable support for distributed transactions change these property values to the following
javaclassname oracle.jms.AQjmsXAQueueConnectionFactory

javafactory oracle.jms.AQjmsXAConnectionFactory

Submit your changes and run the test program below to verify your changes.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package oraclejmsadmin;

import java.util.Hashtable;
import javax.jms.ConnectionMetaData;
import javax.jms.JMSException;
import javax.jms.XAQueueConnection;
import javax.jms.XAQueueConnectionFactory;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import static java.lang.System.out;
/**
 *
 * @author shailesh
 */
public class TestRegistration {


    public void runTest() {
        Hashtable env = null;
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://<oid_server>:<oid_port>");
        env.put("searchbase", "cn=<oid_db_entry>, cn=OracleContext");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL, "cn=<oid_user>");
        env.put(Context.SECURITY_CREDENTIALS, "<oid_password>");
        //recommended in oracle metalink knowledgebase.
        env.put("server_dn", "cn=<oid_db_entry>,cn=OracleContext");
        XAQueueConnection queueConnection = null;
        InitialContext dirContext = null;
        try {
            dirContext = new InitialContext(env);
            XAQueueConnectionFactory qcFactory = (XAQueueConnectionFactory)
dirContext.lookup("cn=<connection_factory_name>,cn=OracleDBConnections, cn=<oid_db_entry>,
cn=OracleContext");
            if (qcFactory != null) {
                queueConnection = qcFactory.createXAQueueConnection("<db_user>",
"<db_password>");
                ConnectionMetaData connMetaData = queueConnection.getMetaData();
                out.println("JMS Version = " + connMetaData.getJMSVersion());
                out.println("JMS Major Version = " + connMetaData.getJMSMajorVersion());
                out.println("JMS Minor Version = " + connMetaData.getJMSMinorVersion());
                out.println("JMS Provider Name = " + connMetaData.getJMSProviderName());
                out.println("Provider Version = " + connMetaData.getProviderVersion());
                out.println("Provider Major Version = " +
connMetaData.getProviderMajorVersion());
                out.println("Provider Minor Version = " +
connMetaData.getProviderMinorVersion());
                connMetaData = null;
                queueConnection.close();
                queueConnection = null;
                dirContext.close();
                dirContext = null;
            }
        }catch(NamingException ne) {
            ne.printStackTrace();
        }catch(JMSException jmsEx) {
            jmsEx.printStackTrace();
        }finally {
            try {
                if(queueConnection != null) {
                    queueConnection.close();
                    queueConnection = null;
                }
                if(dirContext != null) {
                    dirContext.close();
```

```
                    dirContext = null;
                }
            }catch(JMSException jmsEx) {
                jmsEx.printStackTrace();
            }catch(NamingException ne) {
                ne.printStackTrace();
            }
        }
    }

    public static void main(String args[]) {
        TestRegistration testRegistration = new TestRegistration();

    }
}
```

Oracle AQ Setup and Testing.

Warning

Before proceeding to creating queues please make sure the parameter GLOBAL_TOPIC_ENABLED has been set for the database. This ensures that all queues and topics created in Oracle Streams AQ are automatically registered with the LDAP server.

Useful Information

To set GLOBAL_TOPIC_ENABLED for your database run the script below
ALTER SYSTEM SET GLOBAL_TOPIC_ENABLED=TRUE

Oracle AQ Setup

This is the most easiest step but if not setup correctly could lead to a debugging nightmare.

Setting up Oracle AQ requires SYSDBA access. If you do not have the required access level, please contact your DBA. For the proof of concept I created a queue "SAMPLE_Q". Please check the sample script below. It uses default for most attributes available with Oracle AQ. Additional Advanced attributes should be available on the Oracle site.

```
//Use only to delete and recreate queues
EXECUTE DBMS_AQADM.STOP_QUEUE(queue_name  => 'sample_schema.sample_q');
EXECUTE DBMS_AQADM.DROP_QUEUE (queue_name  => 'sample_schema.sample_q');
EXECUTE DBMS_AQADM.DROP_QUEUE_TABLE (queue_table => 'sample_schema.sample_q', force=>true);


EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (queue_table => 'sample_schema.sample_q',
    queue_payload_type      => 'SYS.AQ$_JMS_TEXT_MESSAGE',
    multiple_consumers      => FALSE
);

EXECUTE DBMS_AQADM.CREATE_QUEUE (
    queue_name              => 'sample_schema.sample_q',
    queue_table             => 'sample_schema.sample_q_tb',
    queue_type              => DBMS_AQADM.NORMAL_QUEUE,
    retention_time          => DBMS_AQADM.INFINITE,
    max_retries             => 5,
    retry_delay             => 60
);
```

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

Be Careful with queue_payload_type

The queue_payload_type should be set to a proper JMS message type or a user defined Oracle type. JMS message payload types available in oracle are as follows:

- SYS.AQ$_JMS_TEXT_MESSAGE
- SYS.AQ$_JMS_MAP_MESSAGE
- SYS.AQ$_JMS_BYTES_MESSAGE
- SYS.AQ$_JMS_OBJECT_MESSAGE
- SYS.AQ$_JMS_STREAM_MESSAGE

JMS client does not work with Oracle AQ when setup with a default message payload type (RAW). JMS client throws a NullPointerException during deployment.

Testing

It is very important that we test our queue before moving on to the configuration of the Resource Adapter in Glassfish. Testing includes the following

- Verifying that queue is registered with Oracle LDAP server(OID).
- Enqueueing a message to our sample_q queue.
- Writing a small java application to test connectivity of JMS client and Oracle AQ.
- Modifying the sample java application to be able to dequeue the message.

Oracle LDAP server verification

In this step we use any LDAP brower to connect to the Oracle LDAP server and browse the directory structure to locate our database service name. If done correctly it should be under WORLD->com->sub-domain->OracleContext. If the database service entry does not exist you will have to go back to the OID setup and database registration section of this document. If the database service entry does exist then the queues should be registered under OracleDBQueues directory and a corresponding queue table should also be under OracleDBQueueTables.
Small example to test Queues

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package oraclejmsadmin;

import java.util.Hashtable;

import javax.jms.JMSException;
import javax.jms.Queue;
import javax.jms.XAQueueConnection;
import javax.naming.Context;
import javax.naming.InitialContext;
```

```
import javax.naming.NamingException;
import oracle.jms.AQjmsOracleDebug;
import static java.lang.System.out;

/**
 *
 * @author shailesh
 */
public class QueueFactory {

    Hashtable env = null;
    boolean envSet = false;

    public void setEnv() {
        env = new Hashtable(5, 0.75f);
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://<iod_server>:<oid_nonssl_port>");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL, "<OID_user>");
        env.put(Context.SECURITY_CREDENTIALS, "<OID_PASSWORD>");
        envSet = true;
    }

    public void testRegistration() {
        AQjmsOracleDebug.setLogStream(out);
        AQjmsOracleDebug.setTraceLevel(AQjmsOracleDebug.AQ_ORA_TR6);
        AQjmsOracleDebug.setDebug(true);
        XAQueueConnection queueConnection = null;
        InitialContext dirContext = null;
        try {
            dirContext = new InitialContext(env);
            Queue queue = (Queue) dirContext.lookup("cn=<queue_name>, cn=OracleDBQueues,
cn=SERV, cn=OracleContext, dc=<sub-domain>, dc=com");
            if (queue != null) {
                out.println("got queue object " + queue.getQueueName());
            }

        } catch (NamingException ne) {
            ne.printStackTrace();
        } catch (JMSException jmsEx) {
            jmsEx.printStackTrace();
        } finally {
            try {
                if (queueConnection != null) {
                    queueConnection.close();
                    queueConnection = null;
                }
                if (dirContext != null) {
                    dirContext.close();
                    dirContext = null;
                }
            } catch (JMSException jmsEx) {
                jmsEx.printStackTrace();
            } catch (NamingException ne) {
                ne.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        QueueFactory qf = new QueueFactory();
        qf.setEnv();
        qf.testRegistration();
        qf=null;
    }
}
```

This should print the queue name. If it throws an exception please drop the queues and queuetables from the database and create them again. Do not change the information on the directory browser once they are created. This may cause the queue to not work as expected.

Enqueueing a message

Assuming that sample_q is setup correctly and the schema used by your application can access it we proceed with creating a sample script to enqueue text messages to our queue.

```
DECLARE
    message sys.AQ$_JMS_TEXT_MESSAGE;
    enqueue_options dbms_aq.enqueue_options_t;
    message_properties dbms_aq.message_properties_t;
    msgid raw(16);
BEGIN

    -- Consturct a empty test message object
    message := sys.AQ$_JMS_TEXT_MESSAGE.CONSTRUCT;

    -- Pass some custom properties that could be used by the Java JMS Client.
    message.set_string_property('red-color', 'RED');
    message.set_string_property('yellow-color', 'YELLOW');
    message.set_string_property('green-color', 'GREEN');

    -- Set the actualy text message. You may also send xml data as text.
    message.set_text('This is a sample message. Please process it.  sdfsdfd');

    -- Enqueue your message
    dbms_aq.enqueue(queue_name => 'schema_name.sample_q',
        enqueue_options => enqueue_options,
        message_properties => message_properties,
        payload => message,
        msgid => msgid);

COMMIT;
END;
```

To verify if the messages were successfully queued we can query the sample_q_tb table created with our queue. Please run this query below.

```
select * from schema_name.sample_q_tb;
```

Every message enqueued corresponds to a row in sample_q_tb table.
We have just finished the Oracle Setup and testing part. We will move on the the Glassfish Configuration now.

Oracle AQ Retry hint

If you have setup Oracle AQ to retry dequeuing messages from the queue and the messages get stuck at state 1 (retry required) and never go to state 3 (max retry count elapsed) then the Oracle background process QMN may not be running on the database server.
QMN- Queue Monitor Process (QMNn) - Used to manage Oracle Streams Advanced Queuing.

Glassfish configuration for Oracle AQ

If already not done so please download Glassfish 9.0 V2 from java.net. You can refer Glassfish documentation for the installation instructions.

Download Generic JMS Resource Adapter

You will need to download Generic JMS Resource adapter from java.net. For debugging purposes, it is strongly recommended that you also download the source from java.net. You will be required to register with java.net to be able to download the source files.

Configuring Glassfish

Glassfish application server has to be configured to use the generic jms resource adapter. The default installation comes bundled with a generic jms resource adapter which could be used for the configuration. However, we will using V2.0 of the generic jms resource adapter so it is better to install a separate resource adapter than the one pre-bundled with glassfish.

Generic jms resource adapter can be deployed to Glassfish application server through the web based admin console or the asadmin command line utility.

Configuring generic jms resource adapter through asadmin command line utility comprises of following steps.

1. Deploying the resource adapter
   Before you run the script below please change the : and \ per your environment. For simplicity the script is divided into multiple lines but it should be run in a single line with not "\". For windows environment please replace : with ; as the property seperator.

create resource adapter config

```
./asadmin create-resource-adapter-config --user admin --passwordfile <USER_HOME>/.asadminpass
--property
ProviderIntegrationMode=jndi:\
LogLevel=FINEST:\
RMPolicy=ProviderManaged:\
SupportsXA=true:\
UserName=dbuser:\
Password=pwd:\
DeliveryType=Synchronous:\
JndiProperties=java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://<ldap_server>:<port>,
java.naming.security.authentication=simple,
java.naming.security.principal=cn=<LDAP_USER>,
java.naming.security.credentails=<LDAP_USER_PWD>
```

You should be able to set these properties through the admin console web application.

Once the resource adapter configuration has been created you will now have to deploy the resource adapter as shown below.

deploy connector rar

```
./asadmin deploy --user admin --passwordfile <USER_HOME>/.asadminpass <your_path>/oracleaq.rar
```

Please note that I renamed the genericjmsra.rar file to oracleaq.rar.

1. Configuration for Inbound Messages
   Inbound Messages are the one's that the container listens to and forwards it to the appropriate MDB for processing. If your MDB does not require to send any messages then you are not required to setup the configuration for Outbound messages.
   To understand the configuration steps we will now look at our sample MDB

```
package com.mdb;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import static java.lang.System.out;

public class SampleQMDBBean implements MessageListener {

    private static Logger logger = Logger.getLogger(SampleQMDBBean.class.getName());

    public void onMessage(Message message) {
        logger.fine("Executing onMessage - SampleQMDBBean");
        if(logger.isLoggable(Level.FINEST)) {
            printMessageMetadata(message);
        }
        TextMessage sampleMessage = (TextMessage) message;
        try {
            if(sampleMessage != null) {
                logger.fine("TextMessage Details :");
                logger.fine("TextMessage ID  : " + sampleMessage.getJMSMessageID());
                logger.fine("Text Message Properties");
                logger.fine("red-color property value " +
sampleMessage.getStringProperty("red-color"));
                logger.fine("yellow-color property value " +
sampleMessage.getStringProperty("yellow-color"));
                logger.fine("green-color property value " +
sampleMessage.getStringProperty("green-color"));
                logger.fine("Text Message Body");
                logger.fine(sampleMessage.getText());

            }
        }catch(JMSException ex) {
            logger.severe(ex);
        } finally {
            logger.fine("Finished executing onMessage");
        }

    }

    private void printMessageMetadata(Message arg0) {
        try {
            logger.finest("Message ID = " + arg0.getJMSMessageID());
            logger.finest("JMS Type = " + arg0.getJMSType());
            logger.finest("Message Priority = " + arg0.getJMSPriority());
            logger.finest("Message Delivery Mode = " + arg0.getJMSDeliveryMode());
            logger.finest("Message Redelivered = " + arg0.getJMSRedelivered());
            logger.finest("Message Timestamp = " + arg0.getJMSTimestamp());
        }catch(JMSException jmsEx) {
            logger.warning("Exception while priting message properties " + jmsEx);
        }
    }

}
```

   Glassfish application server reads the Activation Configuration to map this MDB to listen to a particular destination. With EJB 3.0 we can specify activation configuration properties either through annotations or sun-ejb-jar.xml metadata file. However, the annotation apply only to the default provider from Glassfish (SJMQ). We have to use metadata file to setup AQ. Here's the sun-ejb-jar.xml for SampleQMDBBean MDB.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 9.0 EJB
3.0//EN"
"http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-0.dtd">
<sun-ejb-jar>
   <enterprise-beans>
       <ejb>
           <ejb-name>SampleQMDBBean</ejb-name>
           <jndi-name>jms/SampleQMDBBean</jndi-name>
           <mdb-resource-adapter>
               <resource-adapter-mid>oracleaq</resource-adapter-mid>
               <activation-config>
                   <activation-config-property>
                       <activation-config-property-name>
                          ProviderIntegrationMode
                       </activation-config-property-name>
                       <activation-config-property-value>jndi</activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          ConnectionFactoryJndiName
                       </activation-config-property-name>
                       <activation-config-property-value>
                          cn=<factory_name>,cn=OracleDBConnections,cn=<DB_NAME>,cn=OracleContext,dc=xyz,dc=
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          DestinationJndiName
                       </activation-config-property-name>
                       <activation-config-property-value>
                         cn=<Queue_name>,cn=OracleDBQueues,cn=<DB_NAME>,cn=OracleContext,dc=xyz,dc=com>
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          DestinationType
                       </activation-config-property-name>
                       <activation-config-property-value>
                          javax.jms.Queue
                      </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          MaxPoolSize
                       </activation-config-property-name>
                       <activation-config-property-value>
                          1
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          RedeliveryAttempts
                       </activation-config-property-name>
                       <activation-config-property-value>
                           0
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          ReconnectAttempts
                       </activation-config-property-name>
                       <activation-config-property-value>
                         4
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                          ReconnectInterval
                       </activation-config-property-name>
                       <activation-config-property-value>
                           60
                       </activation-config-property-value>
                   </activation-config-property>
                   <activation-config-property>
                       <activation-config-property-name>
                           RedeliveryInterval
```

```
                    </activation-config-property-name>
                    <activation-config-property-value>
                         0
                    </activation-config-property-value>
                </activation-config-property>
                <activation-config-property>
                    <activation-config-property-name>
                         SendBadMessagesToDMD
                    </activation-config-property-name>
                    <activation-config-property-value>
                         false
                    </activation-config-property-value>
                </activation-config-property>
            </activation-config>
        </mdb-resource-adapter>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns = "http://java.sun.com/xml/ns/javaee"
         version = "3.0"
         xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd">
   <enterprise-beans>
       <message-driven>
           <ejb-name>SampleQMDBBean</ejb-name>
           <ejb-class>com.xyz.mdb.SampleQMDBBean</ejb-class>
           <transaction-type>Container</transaction-type>
           <message-destination-type>javax.jms.Queue</message-destination-type>
       </message-driven>
   </enterprise-beans>
   <assembly-descriptor>
       <container-transaction>
           <method>
               <ejb-name>SampleQMDBBean</ejb-name>
               <method-name>onMessage</method-name>
               <method-params>
                   <method-param>javax.jms.Message</method-param>
               </method-params>
           </method>
           <trans-attribute>Required</trans-attribute>
       </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

2. Configuration for Outbound Messages
   If the MDB or Sesion beans are required to send messages to a jms Destination (Oracle AQ) then we need to configure glassfish application server for Outbound Messages. MDB's and Session beans will use ConnectionFactories and Administered Destination to send messages.
   Once both the scripts run successfully we move to the next step of Creating a ConnectionFactory.
   a. Creating a ConnectionFactory
      We start with creating a connector connection pool. We will be creating a ConnectionFactory of type QueueConnectionFactory.

create connector connection pool

```
./asadmin create-connector-connection-pool
--raname oracleaq
--connectiondefinition javax.jms.QueueConnectionFactory
--transactionsupport XATransaction
--isconnectvalidatereq true
--description Connector pool for oracle AQ
--property ConnectionFactoryJndiName=cn=<cf_name>,cn=OracleDBConnections,
cn=<db_name>, cn=OracleContext, dc=xyz, dc=com://ConnectionValidationEnabled=true
```

jms/qcpool

Once the pool has been created we create a JNDI entry to the connection factory by running the script below.

create connector resource

./asadmin create-connector-resource --poolname jms/qcpool jms/QCFactory

Once the connector resource has been successfully created we move on to the next step of creating a destination.

b. Creating a Destination
We will be creating a Queue as our destination.

create admin object

./asadmin create-admin-object --raname oracleaq --restype javax.jms.Queue --property DestinationJndiName=cn=<queue_name>,cn=OracleDBQueues, cn=<db_name>, cn=OracleContext, dc=xyz, dc=com jms/sample_aq

Modified version of SampleQMDBBean to send messages.

```
package com.mdb;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import static java.lang.System.out;

public class SampleQMDBBean implements MessageListener {

    @Resource(name="jms/QCFactory")
    QueueConnectionFactory connectionFactory;

    @Resource(name="jms/sample_aq")
    Queue queue;
    private static Logger logger = Logger.getLogger(SampleQMDBBean.class.getName());

    public void onMessage(Message message) {
        logger.fine("Executing onMessage - SampleQMDBBean");
        if(logger.isLoggable(Level.FINEST)) {
            printMessageMetadata(message);
        }
        TextMessage sampleMessage = (TextMessage) message;
        try {
            if(sampleMessage != null) {
                logger.fine("TextMessage Details :");
                logger.fine("TextMessage ID  : " + sampleMessage.getJMSMessageID());
                logger.fine("Text Message Properties");
                logger.fine("red-color property value " +
sampleMessage.getStringProperty("red-color"));
                logger.fine("yellow-color property value " +
sampleMessage.getStringProperty("yellow-color"));
                logger.fine("green-color property value " +
```

```
"green-color"));
                logger.fine("Text Message Body");
                logger.fine(sampleMessage.getText());

            }
            XAQueueConnection queueConn =
(XAQueueConnection)connectionFactory.createXAQueueConnection();
            XAQueueSession session = queueConnection.createXAQueueSession();
            TextMessage message = session.createTextMessage();
            message.setText("Testing Outbound communication");
            MessageProducer producer = session.createProducer(queue);
            producer.send(message);
            producer.close();
            session.close();
          }catch(JMSException ex) {
            logger.severe(ex);
          } finally {
            logger.fine("Finished executing onMessage");
          }

    }

    private void printMessageMetadata(Message arg0) {
        try {
            logger.finest("Message ID = " + arg0.getJMSMessageID());
            logger.finest("JMS Type = " + arg0.getJMSType());
            logger.finest("Message Priority = " + arg0.getJMSPriority());
            logger.finest("Message Delivery Mode = " + arg0.getJMSDeliveryMode());
            logger.finest("Message Redelivered = " + arg0.getJMSRedelivered());
            logger.finest("Message Timestamp = " + arg0.getJMSTimestamp());
        }catch(JMSException jmsEx) {
            logger.warning("Exception while priting message properties " + jmsEx);
        }
    }

}
```

Add references to Queue/Topic connection factory and destinations to the ejb-jar.xml and sun-ejb-jar.xml metadata files.

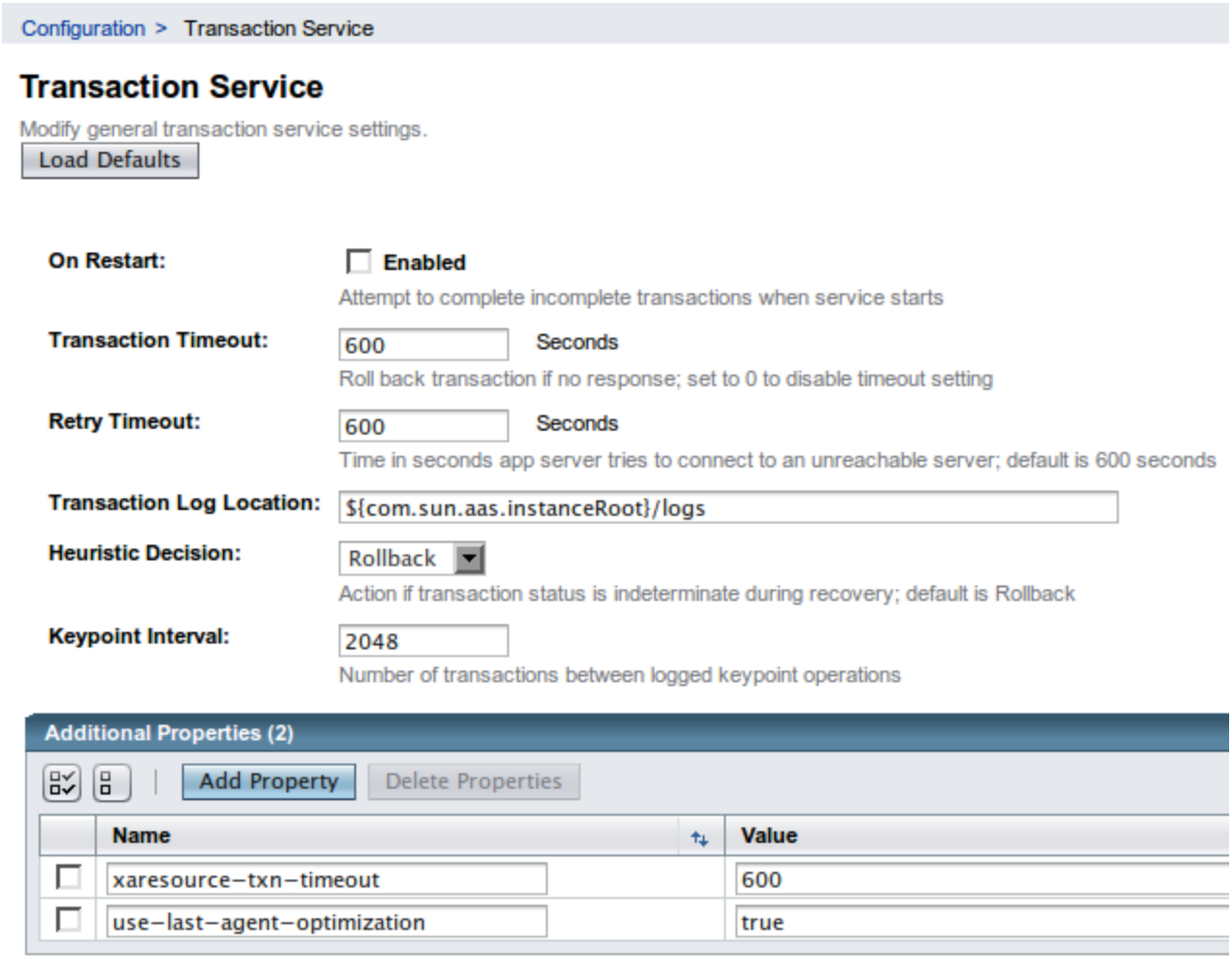
Glassfish Transaction Manager Setup

Introduction

We have to make a few changes to the Transaction Manager Service to support Oracle Advanced Queuing. However, these setting could apply of any other application that require transactions across multiple resources.

Details

To get to the Transaction Service log in to Glassfish Web Administration Console. Click Configuration -> Transaction Service.



Change the default properties values to the one shown in the screen capture above.

Property	Description
On Restart	Enable this option if you want Glassfish to recover all the failed/suspended transactions and try to run them on restart. This is not completely stable yet. We have not enabled this option yet.
Transaction Timeout	Set a non zero value for this option for transaction that run for a long time. Long running transactions are usually not required in CRUD applications so non zero value will ensure a transaction will be marked for rollback after the timeout has elapsed. Rollback transaction if no response; set to 0 seconds to disable timeout setting. We have this value set to 600 seconds.
Retry Timeout	Time in seconds the application server tries to connect to an unreachable server. Default is 600 seconds and that's what we have selected.
Transaction Log Location	${com.sun.aas.installRoot}/logs is the default location. We use the default location.
Heuristic Decision	Action if transaction status is indeterminate during

	recovery; default is Rollback which is what we have selected. Selecting Commit could be unpredictable and dangerous.
Keypoint Interval	Number of transactions between logged keypoint operations. An appropriate value has to be selected. A low value could cause performance issues as the file has to be cleared often. A huge value could improve performance but increases the file size. We have selected 2048, twice the default which is 1024
xaresource-txn-timeout	This property when set to a non zero value enables the transaction timeout for an XAResoruce participating in a global transaction. In our case we set this property to 600 seconds
use-last-agent-optimization	All the resources participating in the global transactions are required to be XA or the transaction is marked for rollback immediately. In scenarios where you have one of the participating resources to be non XA, you should set this option and the application server will consider it to be an XA resource. However, only one participating resource could be non XA and it should be the last resource in the transaction. We have set this value to true. We use global transaction for Oracle AQ. The messages are dequeued and business logic is executed withing one global transaction. The message dequeue is handled by a XA resource and the business logic uses a non XA resource (datasource)

Useful Reference Links

- Introduction to Oracle Advanced Queuing
- GlassFish V2 and ActiveMQ 4.1
- Oracle Internet Directory Administrator's Guide
- Forum: Advanced Queueing
- Using Oracle Java Message Service (OJMS) to Access Oracle Streams AQ
- Oracle JMS Basic Operations
- Sun- Configuring Java Message Service Resources
- Nabble-Development of Generic Resource Adapter for JMS
- User Guide to Resource Adapter for Oracle Advanced Queueing
- Middleware- Simple MDB with Oracle Database JMS Provider
- Oracle-Diagnosability in JDBC
- Oracle JMS Types
- Akadia-Oracle Advanced Queueing
- Oracle - Creating Oracle Streams AQ Applications Using JMS
- Oracle - Oracle Streams AQ JMS Types Examples
- Glassfish Documentation
- The JMS API Programming Model