Using GenericJMSRA with IBM WebSphere MQ

You can configure GenericJMSRA to enable applications running in GlassFish Server to send messages to, and receive messages from, IBM WebSphere MQ. GlassFish Serveronly supports using GenericJMSRA with WebSphere MQ version 6.0 and WebSphere MQ version 7.0

These instructions assume that the WebSphere MQ broker and GlassFish Server are deployed and running on the same physical host/machine. If you have the WebSphere MQ broker running on a different machine and need to access it remotely, refer to the WebSphere MQ documentation for configuration details. The resource adapter configuration and other application server related configuration remains unchanged.

The following topics are addressed here:

Preliminary Setup Procedures for WebSphere MQ Integration

Before you can configure WebSphere MQ to exchange messages with GlassFish Server, you must complete the following tasks:

- The following permissions must be added to the `server.policy` and the `client.policy` file to deploy GenericJMSRA and to run the client application.
  - Use a text editor to modify the `server.policy` file in the `${appserver-install-dir}/domains/domain1/config/`directory by adding the following line to the default grant block:

    ```
    permission java.util.logging.LoggingPermission "control";
    permission java.util.PropertyPermission "*", "read,write";
    ```

  - If you use an application client in your application, edit the client's `client.policy` file in the `${appserver-install-dir}/lib/appclient/` directory by adding the following permission:

    ```
    permission javax.security.auth.PrivateCredentialPermission
    "javax.resource.spi.security.PasswordCredential * \"*\"","read";
    ```

- To integrate GlassFish Serverwith WebSphere MQ 6.0 or 7.0, copy the necessary JAR files to the *as-install*`/lib` directory:
  - For WebSphere MQ 6.0, copy these JAR files to the *as-install*`/lib` directory:

    ```
    /opt/mqm/java/lib/com.ibm.mq.jar
    /opt/mqm/java/lib/com.ibm.mq.jms.Nojndi.jar
    /opt/mqm/java/lib/com.ibm.mq.soap.jar
    ```

```
/opt/mqm/java/lib/com.ibm.mqjms.jar
/opt/mqm/java/lib/com.ibm.mqetclient.jar
/opt/mqm/java/lib/commonservices.jar
/opt/mqm/java/lib/dhbcore.jar
/opt/mqm/java/lib/rmm.jar
/opt/mqm/java/lib/providerutil.jar
/opt/mqm/java/lib/CL3Export.jar
/opt/mqm/java/lib/CL3Nonexport.jar
```

where `/opt/mqm` is the location of the WebSphere MQ 6.0 installation.

- For WebSphere MQ 7.0, copy these JAR files to the *as-install*`/lib` directory:

```
/opt/mqm/java/lib/com.ibm.mq.jar,
/opt/mqm/java/lib/com.ibm.mq.jms.Nojndi.jar,
/opt/mqm/java/lib/com.ibm.mq.soap.jar,
/opt/mqm/java/lib/com.ibm.mqjms.jar,
/opt/mqm/java/lib/com.ibm.mq.jmqi.jar,
/opt/mqm/java/lib/com.ibm.mq.commonservices.jar,
/opt/mqm/java/lib/dhbcore.jar,
/opt/mqm/java/lib/rmm.jar,
/opt/mqm/java/lib/providerutil.jar,
/opt/mqm/java/lib/CL3Export.jar,
/opt/mqm/java/lib/CL3Nonexport.jar
```

where `/opt/mqm` is the location of the WebSphere MQ 7.0 installation.

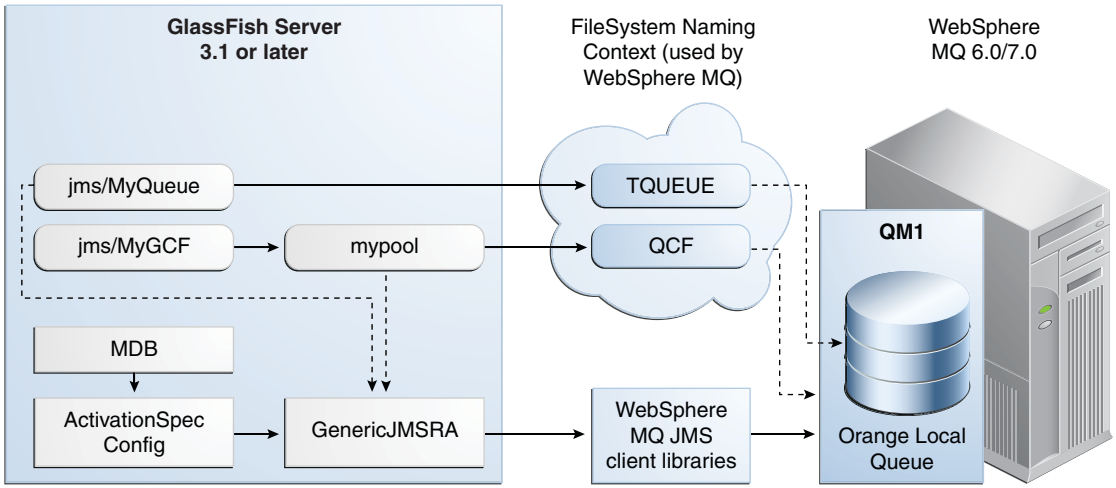
- Set the `LD_LIBRARY_PATH` environment variable to the `java/lib` directory, and then restart GlassFish Server. For example, in a UNIX—based system, with WebSphere MQ installed under `/opt/mqm`, you would enter:

```
$ export LD_LIBRARY_PATH=/opt/mqm/java/lib
```

▼ Configure the WebSphere MQ Administered Objects

This section provides an example of how you could configure the necessary administered objects, such as destinations and connection factories, on the WebSphere MQ instance from which you want to access messages using GlassFish Server. Therefore, you will need to change the administered object names to suit your installation.

Before You Begin If WebSphere MQ created a user and a group named `mqm` during the installation, then you must specify a password for the `mqm` user using the `$ passwd mqm` command.



1. **Switch to the `mqm` user:**

   ```
   $ su mqm
   ```

2. **For Linux, set the following kernel version:**

   ```
   $ export LD_ASSUME_KERNEL=2.2.5
   ```

3. **Create a new MQ queue manager named "QM1":**

   ```
   $ crtmqm QM1
   ```

4. **Start the new MQ queue manager.**

   In the image above, `QM1` is associated with the IBM WebSphere MQ broker.

   ```
   $ strmqm QM1
   ```

5. **Start the MQ listener:**

   ```
   $ runmqlsr -t tcp -m QM1 -p 1414 &
   ```

6. **Modify the default JMSAdmin console configuration as follows:**

   a. **Edit the JMSAdmin script in the `/opt/mqm/java/bin` directory to change the JVM to a location of a valid JVM your system.**

   b. **Set the relevant environment variable required for JMSAdmin by sourcing the `setjmsenv` script located in the `/opt/mqm/java/bin` directory.**

   ```
   $ cd /opt/mqm/java/bin
   $ source setjmsenv
   ```

where `/opt/mqm` is the location of the WebSphere MQ installation.

c. Change the JMSAdmin.config file to indicate the Initial Context Factory you will be using by setting the following name-value pairs and commenting out the rest:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
       PROVIDER_URL=file:/opt/tmp
```

7 Create WebSphere MQ queues using the runmqsc console and MQJMS_PSQ.mqsc script.

```
$ runmqsc QM1 < MQJMS_PSQ.mqsc
```

8 Create user defined physical queue for your application using runmqsc console and an appropriate physical queue name. An example of how this could be done is shown below.

In the image above, `ORANGE.LOCAL.QUEUE` is associated with `QM1`.

```
$ runmqsc QM1
         > DEFINE QLOCAL(ORANGE.LOCAL.QUEUE)
         > end
```

9 Start the WebSphere MQ Broker:

```
$ strmqbrk -m QM1
```

10 In the WebSphere MQ JMSAdmin console, use the following commands to create the connection factories, XA connection factories, and destinations for your application, as shown in the following sample, which lists each of the various JMS administered objects.

In the image above, `QCF` (for `QM1`) and `TQueue` (associated with `ORANGE.LOCAL.QUEUE`) are defined in the `FileSystem Naming Context`.

```
$ ./JMSAdmin

   InitCtx>def qcf<JNDI name to be given to the Queue Connection Factory>
     hostname<IBM MQ server hostname> port(1414) channel(SYSTEM.DEF.SVRCONN)
     transport(CLIENT) qmanager<name of queue manager defined>

     For example:
     def qcf(QCF) hostname(localhost) port(1414) channel(SYSTEM.DEF.SVRCONN)
      transport(CLIENT) qmanager(QM1)

   InitCtx%def xaqcf<JNDI name to be given to the XA Queue Connection Factory>
     hostname<IBM MQ server hostname> port(1414) channel(SYSTEM.DEF.SVRCONN)
     transport(CLIENT) qmanager<name of queue manager defined>

     For example:
     def xaqcf(XAQCF) hostname(localhost) port(1414) channel(SYSTEM.DEF.SVRCONN)
     transport(CLIENT) qmanager(QM1)

   InitCtx%def q<JNDI Name to be given to the Queue> queue<physical queue name>
     qmanager(name of queue manager defined )

     For example: def q(TQueue) queue(ORANGE.LOCAL.QUEUE) qmanager(QM1)

   InitCtx%def tcf<JNDI Name to be given to the Topic Connection Factory>
     qmanager(name of queue manager defined )
```

```
  For example: def tcf(TCF) qmanager(QM1)

InitCtx%def xatcf<JNDI Name to be given to the  XA Topic Connection Factory>
  qmanager(name of queue manager defined )

  For example: def xatcf(XATCF) qmanager(QM1)

InitCtx%def t<JNDI Name to be given to the Topic> topic<sample topic name>

  For example: def t(TTopic) topic(topic)
```

Create a Resource Adapter Configuration for GenericJMSRA to Work With WebSphere MQ

Before deploying GenericJMSRA, you need to create a resource adapter configuration in GlassFish Server. You can do this using either the Administration Console or the `asadmin` command. Use the following `asadmin` command to create a resource adapter configuration for `genericra` to configure it to work with WebSphere MQ.

```
asadmin> create-resource-adapter-config
  --user <adminname> --password <admin password>
  --property SupportsXA=true:ProviderIntegrationMode
  =jndi:UserName=mqm:Password=###:RMPolicy
  =OnePerPhysicalConnection:JndiProperties
  =java.naming.factory.url.pkgs\\
  =com.ibm.mq.jms.naming,java.naming.factory.initial\\
  =com.sun.jndi.fscontext.RefFSContextFactory,java.naming.provider.url\\
  =file\\:\\/\\/opt\\/tmp:LogLevel=finest genericra
```

Note – When using `asadmin` you need to escape each `=` and any `:` characters by prepending a backward slash `\`. The escape sequence is not necessary if the configuration is performed through the Administration Console. Also , ensure that the provider URL is configured correctly depending on the platform. For example, on Windows systems it should be `file:/C:/opt/tmp` and on UNIX—based systems it is `file://opt/tmp`.

This creates a resource adapter configuration with the name `genericra`, and Oracle recommends not changing the default name. The resource adapter configuration is configured with the properties specified using the `--properties` argument; multiple properties are configured as a colon-separated list of name-value pairs that are entered as a single line.

In this example, the following properties are configured:

Note – The tables in this section describe the GenericJMSRA properties that are relevant only when integrating with WebSphere MQ. For a complete list of properties, see the comprehensive table in "GenericJMSRA Configuration Properties" on page 40.

Property Name	*Required Value*	*Description*
`SupportsXA`	`true`	Set the supports distributed transactions attribute to `true`. The level of transactional support the adapter provides -- none, local, or XA -- depends on the capabilities of the Enterprise Information System (EIS) being adapted. If an adapter supports XA transactions and this attribute is XA, the application can use distributed transactions to coordinate the EIS resource with JDBC and JMS resources.
`ProviderIntegrationMode`	`jndi`	Specifies that connection factories and destinations in GlassFish's JNDI store are configured to refer to connection factories and destinations in WebSphere MQ's JNDI store.

`JndiProperties`	`JndiProperties=`
	`java.naming.factory.url.pkgs\\`
	`=com.ibm.mq.jms.naming,java.naming.`
	`factory.initial\\=com.sun.jndi.fscontext.`
	`RefFSContextFactory,java.naming.`
	`provider.url\\`
	`=file\\:\\/\\/opt\\/tmp:`
	`LogLevel=finest genericra`
		JNDI properties for connecting to WebSphere MQ's JNDI, specified as comma-separated list of name=value pairs without spaces.

`UserName`	`Name of the WebSphere MQ user`	User name to connect to WebSphere MQ. The user name can be overridden in `ActivationSpec` and `ManagedConnection`. If no user name is specified anonymous connections will be used, if permitted.
`Password`	`Password for the WebSphere MQ user`	Password to connect to WebSphere MQ. The password can be overridden in `ActivationSpec` and `ManagedConnection`.

Property Name	*Required Value*	*Description*
`RMIPolicy`	`OnePerPhysicalConnection`	Some XAResource implementations, such as WebSphere MQ, rely on a Resource Manager per Physical Connection, and this causes issues when there is inbound and outbound communication to the same queue manager in a single transaction (for example, when an MDB sends a response to a destination). When *RMPolicy* is set to `OnePerPhysicalConnection`, the XAResource wrapper implementation's `isSameRM` in GenericJMSRA would check if both the XAResources use the same physical connection, before delegating to the wrapped objects. Therefore, ensure that this attribute is set to `OnePerPhysicalConnection` if the application uses XA.
`LogLevel`	`Desired log level of JDK logger`	Used to specify the level of logging.

Note – You must use the values for `SupportsXA`, `RMPolicy` and `ProviderIntegrationMode` as the required values that are used in this table.

Deploy the GenericJMSRA Archive

The GenericJMSRA archive is available as an Add-On in the Administration Console's Update Tool.

For instructions on downloading and deploying GenericJMSRA, see "Deploy the GenericJMSRA Resource Archive" on page 48.

Create the Connection Factories and Administered Objects in GlassFish Server

In order to configure a JMS Connection Factory using GenericJMSRA, a Connector Connection Pool and resource needs to be created in GlassFish Server, as described in this section.

Using the example WebSphere MQ configuration in "Configure the WebSphere MQ Administered Objects" on page 60, you will see `mypool` (pointing to `GenericJMSRA` and `QCF`) and `jms/MyQCF` (for `mypool`) created in GlassFish Server.

Note – If you want configure connections and destination resources using the Administration Console, this is explained in the Administration Console online help. When using Administration Console, following the, instructions for creating a new **Connector Connection Pool** and **Admin Object Resources**, and not the instructions for creating a JMS Connection Pool and Destination Resources. For more information about using `asadmin` to create these resources, see "To Create a Connector Connection Pool" on page and "To Create a Connector Resource" on page .

Creating Connections and Destinations

In order to configure a JMS Connection Factory, using GenericJMSRA, a Connector Connection Pool and Destination resources need to be created in GlassFish Server using names that map to the corresponding connection and destination resources in WebSphere MQ. The connections and destination name in these steps map to the example WebSphere MQ configuration in "Configure the WebSphere MQ Administered Objects" on page 60.

1 Create connection pools that point to the connection pools in WebSphere MQ.

The following `asadmin` command creates a Connection Pool called `mypool` and points to the `XAQCF` created in WebSphere MQ:

```
asadmin create-connector-connection-pool -- raname genericra connectiondefinition
       javax.jms.QueueConnectionFactory --transactionsupport  XATransaction
       --property ConnectionFactoryJndiName=QCF mypool
```

The following `asadmin` command creates a Connection Pool called `mypool2` and points to the `XATCF` created in WebSphere MQ:

```
asadmin create-connector-connection-pool
       -- raname genericra connectiondefinition javax.jms.TopicConnectionFactory
       --transactionsupport  XATransaction
       --property ConnectionFactoryJndiName=XATCF mypool2
```

2 Create the connector resources.

The following `asadmin` command creates a connector resource named `jms/MyQCF` and binds this resource to JNDI for applications to use:

```
asadmin create-connector-resource --poolname mypool jms/MyQCF
```

The following `asadmin` command creates a connector resource named `jms/MyTCF` and binds this resource to JNDI for applications to use:

```
asadmin create-connector-resource --poolname mypool2 jms/MyTCF
```

3. **Create the JMS destination resources as administered objects.**

   In the image above, `jms/MyQueue` (pointing to `GenericJMSRA` and `TQueue`) is created in GlassFish Server.

   The following `asadmin` command creates a `javax.jms.Queue` administered object and binds it to the GlassFish Server JNDI tree at `jms/MyQueue` and points to the `jms/TQueue` created in WebSphere MQ.

   ```
   asadmin create-admin-object --raname genericra --restype javax.jms.Queue
          --property DestinationJndiName=TQueue jms/MyQueue
   ```

   The following `asadmin` command creates a `javax.jms.Topic` administered object and binds it to the GlassFish Server JNDI tree at `jms/MyTopic` and points to the `jms/TTopic` created in WebSphere MQ.

   ```
   asadmin create-admin-object --raname genericra --restype javax.jms.Topic
          --property DestinationJndiName=TTopic jms/MyTopic
   ```

Configuring an MDB to Receive Messages from WebSphere MQ

The administered object names in the sample deployment descriptor below map to the example WebSphere MQ configuration in "Configure the WebSphere MQ Administered Objects" on page 60. The deployment descriptors need to take into account the resource adapter and the connection resources that have been created. A sample `sun-ejb-jar.xml` for a Message Driven Bean that listens to a destination called `TQueue` in WebSphere MQ, and publishes back reply messages to a destination resource named `jms/replyQueue` in GlassFish Server, as shown below.

```
<sun-ejb-jar>
  <enterprise-beans>
  <unique-id.1</unique-id>
  <ejb>
     <ejb-name>SimpleMessageEJB</ejb-name>
     <jndi-name>jms/SampleQueue</jndi-name>
      <!-- QCF used to publish reply messages -->
     <resource-ref>
        <res-ref-name>jms/MyQueueConnectionFactory</res-ref-name>
        <jndi-name>jms/MyQCF</jndi-name>
        <default-resource-principal>
           <name>mqm</name>
           <password>mqm</password>
        </default-resource-principal>
     </resource-ref>
     <!-- reply destination resource> Creating of this replyQueue destination resource is not
          shown above, but the steps are similar to creating the  "jms/MyQueue" resource -->
     <resource-env-ref>
        <resource-env-ref-name>jms/replyQueue</resource-env-ref-name>
        <jndi-name>jms/replyQueue</jndi-name>
     </resource-env-ref>

     <!-- Activation related RA specific configuration for this MDB -->
     <mdb-resource-adapter>
     <!-- resource-adapter-mid points to the Generic Resource Adapter for JMS -->
```

```
      <resource-adapter-mid>genericra</resource-adapter-mid>
      <activation-config>
         <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>ConnectionFactoryJndiName</activation-config-property-name>
            <activation-config-property-value>QCF</activation-config-property-value>
         </activation-config-property>
         <activation-config-property>
            <activation-config-property-name>DestinationJndiName</activation-config-property-name>
            <activation-config-property-value>TQueue</activation-config-property-value>
         </activation-config-property>
         <activation-config-property>
            <activation-config-property-name>MaxPoolSize</activation-config-property-name>
            <activation-config-property-value>32</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>10</activation-config-property-value>
         </activation-config-property>
         <activation-config-property>
            <activation-config-property-name>RedeliveryInterval</activation-config-property-name>
            <activation-config-property-value>1</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>
```

The business logic encoded in Message Driven Bean could then lookup the configured `QueueConnectionFactory/Destination` resource to create a connection as shown below.

```
Context context = null;
ConnectionFactory connectionFactory = null;
logger>info("In PublisherBean>ejbCreate()");
try {
    context = new InitialContext();
    queue = (javax>jms>Queue) context>lookup ("java:comp/env/jms/replyQueue");
    connectionFactory = (ConnectionFactory) context>lookup("java:comp/env/jms/MyQueueConnectionFactory");
    connection = connectionFactory>createConnection();
} catch (Throwable t) {
    logger>severe("PublisherBean>ejbCreate:" + "Exception: " +
    t>toString());
}
```