- If all Message Queue brokers are down, it can take up to 30 minutes for GlassFish Server to go down or up when you are using the default values in JMS. You can change the default values for this timeout. For example:

  ```
  asadmin set domain1.jms-service.reconnect-interval-in-seconds=5
  ```

Using the Generic Resource Adapter for JMS to Integrate Supported External JMS Providers

GlassFish Server supports the integration and use of Oracle WebLogic JMS and IBM WebSphere MQ JMS providers through the use of the Generic Resource Adapter for JMS (GenericJMSRA), which is available as an Add-On in the Administration Console's Update Tool. This Java EE connector 1.5 resource adapter can wrap the JMS client library of Oracle WebLogic JMS and IBM WebSphere MQ and make it available for use by GlassFish. The adapter is a `.rar` archive that can be deployed and configured using GlassFish Server administration tools.

The following topics are addressed here:

Configuring GenericJMSRA for Supported External JMS Providers

The GenericJMSRA can be configured to indicate whether the JMS provider supports XA or not. It is also possible to indicate what mode of integration is possible with the JMS provider. Two modes of integration are supported by GenericJMSRA. The first one uses JNDI as the means of integration. In this situation, administered objects are set up in the JMS provider's JNDI tree and will be looked up for use by GenericJMSRA. Depending on the JMS provider being used, you may need to use either JNDI or JavaBean mode or have the choice of both. If that mode is not suitable for integration, it is also possible to use the Java reflection of JMS administered object JavaBean classes as the mode of integration.

▼ To Deploy and Configure GenericJMSRA

Before deploying GenericJMSRA, JMS client libraries must be made available to GlassFish Server. For some JMS providers, client libraries might also include native libraries. In such cases, these native libraries must be made available to any GlassFish Server JVMs.

1 Download the `genericra.rar` archive as an Add-On in the Administration Console's Update Tool.

2 **Deploy GenericJMSRA the same way you would deploy a connector module.**

See "Deploying a Connector Module" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*

3 **Create a connector connection pool.**

See "To Create a Connector Connection Pool" on page .

4 **Create a connector resource.**

See "To Create a Connector Resource" on page .

5 **Create an administered object resource.**

See "To Create an Administered Object" on page .

GenericJMSRA Configuration Properties

The following table describes the properties that can be set to when configuring the resource adapter.

Property Name	*Valid Values*	*Default Value*	*Description*
`ProviderIntegrationMode`	`javabean/jndi`	`javabean`	Decides the mode of integration between the resource adapter and the JMS client. If `jndi` is specified, then the resource adapter will obtain JMS connection factories and destinations from the JMS provider's JNDI repository. If `javabean` is specified then the resource adapter will obtain JMS connection factories and destinations by instantiating the appropriate classes directly. Which option is specified determines which other properties need to be set.
`ConnectionFactoryClassName`	A valid class name	None	Class name of `javax.jms.ConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is `javabean`.

Property Name	*Valid Values*	*Default Value*	*Description*
`QueueConnection FactoryClassName`	A valid class name	None	Class name of `javax.jms.QueueConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is `javabean`.
`TopicConnectionFactory ClassName`	A valid class name	None	Class name of `javax.jms.TopicConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is specified as `javabean`.
`XAConnectionFactory ClassName`	A valid class name	None	Class name of `javax.jms.ConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is specified as `javabean`.
`XAQueueConnection FactoryClassName`	A valid class name	None	Class name of `javax.jms. XAQueueConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is specified as `javabean`.
`XATopicConnection FactoryClassName`	A valid class name	None	Class name of `javax.jms. XATopicConnectionFactory` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is `javabean`.
`TopicClassName`	A valid class name	None	Class Name of `javax.jms.Topic` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is `javabean`.

Property Name	*Valid Values*	*Default Value*	*Description*
`QueueClassName`	A valid class name	None	Class Name of `javax.jms.Queue` implementation of the JMS client. This class must be made available on the application server classpath. Used if `ProviderIntegrationMode` is specified as a `javabean`.
`SupportsXA`	`True/false`	`FALSE`	Specifies whether the JMS client supports XA or not.
`ConnectionFactory Properties`	Name value pairs separated by comma	None	Specifies the `javabean` property names and values of the `ConnectionFactory` of the JMS client. Required only if `ProviderIntegrationMode` is `javabean`.
`JndiProperties`	Name value pairs separated by comma	None	Specifies the JNDI provider properties to be used for connecting to the JMS provider's JNDI. Used only if `ProviderIntegrationMode` is `jndi`.
`CommonSetter MethodName`	Method name	None	Specifies the common setter method name that some JMS vendors use to set the properties on their administered objects. Used only if `ProviderIntegrationMode` is `javabean`. In the case of Sun Java System Message Queue, this property is named `setProperty`.
`UserName`	Name of the JMS user	None	User name to connect to the JMS Provider.
`Password`	Password for the JMS user	None	Password to connect to the JMS provider.

Property Name	*Valid Values*	*Default Value*	*Description*
`RMPolicy`	`ProviderManaged` or `OnePerPhysicalConnection`	`Provider Managed`	The `isSameRM` method on an `XAResource` is used by the Transaction Manager to determine if the Resource Manager instance represented by two `XAResources` are the same. When `RMPolicy` is set to `ProviderManaged` (the default value), the JMS provider is responsible for determining the `RMPolicy` and the `XAResource` wrappers in the Generic Resource Adapter merely delegate the `isSameRM` call to the message queue provider's XA resource implementations. This should ideally work for most message queue products. 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 Generic Resource Adapter would check if both the `XAResources` use the same physical connection, before delegating to the wrapped objects.

Connection Factory Properties

`ManagedConnectionFactory` properties are specified when a `connector-connection-pool` is created. All the properties specified while creating the resource adapter can be overridden in a `ManagedConnectionFactory`. Additional properties available only in `ManagedConnectionFactory` are given below.

Property Name	*Valid Value*	*Default Value*	*Description*
`ClientId`	A valid client ID	None	`ClientID` as specified by JMS 1.1 specification.

Property Name	*Valid Value*	*Default Value*	*Description*
`ConnectionFactory JndiName`	JNDI Name	None	JNDI name of the connection factory bound in the JNDI tree of the JMS provider. The administrator should provide all connection factory properties (except `clientID`) in the JMS provider itself. This property name will be used only if `ProviderIntegratinMode` is `jndi`.
`ConnectionValidation Enabled`	true/false	FALSE	If set to true, the resource adapter will use an exception listener to catch any connection exception and will send a `CONNECTION_ERROR_OCCURED` event to application server.

Destination Properties

Properties in this section are specified when a destination (queue or topic) is created. All the resource adapter properties can be overridden in a destination. Additional properties available only in the destination are given below.

Property Name	*Valid Value*	*Default Value*	*Description*
`DestinationJndiName`	JNDI Name	None	JNDI name of the destination bound in the JNDI tree of the JMS provider. The Administrator should provide all properties in the JMS provider itself. This property name will be used only if `ProviderIntegrationMode` is `jndi`.
`DestinationProperties`	Name value pairs separated by a comma	None	Specifies the `javabean` property names and values of the destination of the JMS client. Required only if `ProviderIntegrationMode` is `javabean`.

Activation Spec Properties

Properties in this section are specified in the Sun-specific deployment descriptor of MDB as `activation-config-properties`. All the resource adapter properties can be overridden in an Activation Spec. Additional properties available only in ActivationSpec are given below.

Property Name	*Valid Value*	*Default Value*	*Description*
`MaxPoolSize`	An integer	8	Maximum size of server session pool internally created by the resource adapter for achieving concurrent message delivery. This should be equal to the maximum pool size of MDB objects.
`MaxWaitTime`	An integer	3	The resource adapter will wait for the time in seconds specified by this property to obtain a server session from its internal pool. If this limit is exceeded, message delivery will fail.
`Subscription Durability`	Durable or Non-Durable	Non-Durable	`SubscriptionDurability` as specified by JMS 1.1 specification.
`SubscriptionName`		None	`SubscriptionName` as specified by JMS 1.1 specification.
`MessageSelector`	A valid message selector	None	`MessageSelector` as specified by JMS 1.1 specification.
`ClientID`	A valid client ID	None	`ClientID` as specified by JMS 1.1 specification.
`ConnectionFactory JndiName`	A valid JNDI Name	None	JNDI name of connection factory created in JMS provider. This connection factory will be used by resource adapter to create a connection to receive messages. Used only if `ProviderIntegrationMode` is configured as `jndi`.
`DestinationJndiName`	A valid JNDI Name	None	JNDI name of destination created in JMS provider. This destination will be used by resource adapter to create a connection to receive messages from. Used only if `ProviderIntegrationMode` is configured as `jndi`.
`DestinationType`	`javax.jms.Queue` or `javax.jms.Topic`	Null	Type of the destination the MDB will listen to.
`Destination Properties`	Name-value pairs separated by comma	None	Specifies the `javabean` property names and values of the destination of the JMS client. Required only if `ProviderIntegrationMode` is `javabean`.

Property Name	*Valid Value*	*Default Value*	*Description*
`RedeliveryAttempts`	integer		Number of times a message will be delivered if a message causes a runtime exception in the MDB.
`RedeliveryInterval`	time in seconds		Interval between repeated deliveries, if a message causes a runtime exception in the MDB.
`SendBadMessages ToDMD`	true/false	False	Indicates whether the resource adapter should send the messages to a dead message destination, if the number of delivery attempts is exceeded.
`DeadMessage Destination JndiName`	a valid JNDI name.	None	JNDI name of the destination created in the JMS provider. This is the target destination for dead messages. This is used only if `ProviderIntegrationMode` is `jndi`.
`DeadMessage Destination ClassName`	class name of destination object.	None	Used if `ProviderIntegrationMode` is `javabean`.
`DeadMessage Destination Properties`	Name Value Pairs separated by comma	None	Specifies the `javabean` property names and values of the destination of the JMS client. This is required only if `ProviderIntegrationMode` is `javabean`.
`DeadMessageConnectionFactoryJndiName`	a valid JNDI name	None	JNDI name of the connection factory created in the JMS provider. This is the target connection factory for dead messages. This is used only if `ProviderIntegrationMode` is `jndi`.
`DeadMessageDestinationType`	Type queue or topic destination	None	The destination type for dead messages.
`ReconnectAttempts`	integer	0	Number of times a reconnect will be attempted in case exception listener catches an error on connection.
`ReconnectInterval`	time in seconds	0	Interval between reconnects.

Using GenericJMSRA with WebLogic JMS

You can configure GenericJMSRA to enable applications running in GlassFish Server to send messages to, and receive messages from, Oracle WebLogic JMS.

Due to the nature of the WebLogic Server Thin T3 Client that is supported for this purpose, messages exchanged between GlassFish Server and WebLogic Server cannot contain XA transactions, nor can they be asynchronous, as described in detail in "Limitations When Using GenericJMSRA with WebLogic JMS" on page 49.

The following topics are addressed here:

Deploy the WebLogic Thin T3 Client JAR in GlassFish Server

WebLogic Server provides several different clients for use by stand-alone applications that run outside of WebLogic Server. These client are summarized in Overview of Stand-alone Clients in *Programming Stand-alone Clients for Oracle WebLogic Server*. When connecting from GlassFish Server to WebLogic JMS resources you must use the WebLogic Thin T3 client, `wlthint3client.jar`. For Glassfish 3.1 or later, simply add the Thin T3 client JAR to the classpath of your running applications.

There are a couple of methods to deploy the WebLogic Thin T3 client in GlassFish Server:

- To make the Thin T3 client available to all applications, copy the `wlthint3client.jar` to the *as-install*`/lib` directory under your GlassFish Server installation. The Thin T3 client can be found in a WebLogic Server installation in a directory similar to *MW_HOME*`/server/lib`.
- It is also possible to deploy the Thin T3 client in a less global manner, so that it is specific to an individual application. For information on how to do this, see "Application-Specific Class Loading" in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

Configure WebLogic JMS Resources for Integration

If you need to configure the necessary WebLogic JMS resources on the WebLogic Server from which you want to access messages using GlassFish Server, then follow the instructions in the WebLogic Server documentation for configuring the necessary resources, such as destinations, and connection factories.

- JMS System Module Configuration
- Queue and Topic Destination Configuration
- Connection Factory Configuration

The example code snippets in this section refer to a WebLogic JMS connection factory named `WLoutboundQueueFactory` and queue destination named `WLoutboundQueue`. For conceptual overviews on configuring WebLogic JMS resources, refer to Understanding JMS Resource Configuration in *Configuring and Managing JMS for Oracle WebLogic Server*. For detailed instructions on configuring WebLogic JMS resources, refer to Configure JMS system modules and add JMS resources in the WebLogic Administration Console Online Help.

Create a Resource Adapter Configuration for GenericJMSRA to Work With WebLogic JMS

Before deploying GenericJMSRA, you need to create a resource adapter configuration in GlassFish Server. You can do this using either the GlassFish Server Administration console or the `asadmin` command. Here's an example using `asadmin`:

```
asadmin create-resource-adapter-config --host localhost --port 4848
        --property SupportsXA=false:DeliveryType=Synchronous:ProviderIntegrationMode
        =jndi:JndiProperties=java.naming.factory.initial\
        =weblogic.jndi.WLInitialContextFactory,java.naming.provider.url\
        =t3\://localhost\:7001,java.naming.factory.url.pkgs\
        =weblogic.corba.client.naming genericra
```

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. You will also need to change the host and port that WebLogic Server is running on to suit your installation.

In this example, the following properties are configured:

Property	*Value*
`SupportsXA`	`false`
`DeliveryType`	`Synchronous`
`ProviderIntegration Mode`	`jndi`

`JndiProperties`	`java.naming.factory.initial =weblogic.jndi.WLInitialContextFactory,java.naming.provider.url =t3://localhost:7001,java.naming.factory.url.pkgs =weblogic.corba.client.naming`
	
	`(replace "localhost:7001" with the host:port of WebLogic Server)`

You must use the same values for `SupportsXA`, `DeliveryType` and `ProviderIntegrationMode` as the required values that are used in this table. The `JndiProperties` value must be set to a list of JNDI properties needed for connecting to WebLogic JNDI.

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 GUI.

For a description of all the resource adapter properties that are relevant for WebLogic JMS, see the "Configuration Reference of GenericJMSRA Properties for WebLogic JMS" on page 51.

▼ Deploy the GenericJMSRA Archive

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

1 **Download the GenericJMSRA archive (genericra.rar) from the GlassFish Server Update Center.**

2 **Deploy the resource adapter using the `asadmin` deploy command:**

```
$ asadmin deploy --user admin --password adminadmin
  <location of the generic resource adapter rar file>
```

▼ Configuring an MDB to Receive Messages from WebLogic JMS

In this example, all configuration information is defined in two deployment descriptor files: `ejb-jar.xml` and the GlassFish Server `glassfish-ejb-jar.xml` file. To configure a MDB to receive messages from WebLogic JMS, you would configure these deployment descriptor files as follows:

1 **Configure the ejb-jar.xml deployment descriptor:**

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>SimpleMessageEJB</ejb-name>
      <ejb-class>test.simple.queue.ejb.SimpleMessageBean</ejb-class>
      <transaction-type>Container</transaction-type>
    </message-driven>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>SimpleMessageEJB</ejb-name>
        <method-name>onMessage</method-name>
        <method-params>
          <method-param>javax.jms.Message</method-param>
        </method-params>
      </method>
```

```
      <trans-attribute>NotSupported</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

Note – If container-managed transactions are configured, then the transactional attribute must be set to `NotSupported`. For more information, see "Limitations When Using GenericJMSRA with WebLogic JMS" on page 49.

2 Configure the glassfish-ejb-jar.xml deployment descriptor:

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>SimpleMessageEJB</ejb-name>
      <mdb-resource-adapter>
        <resource-adapter-mid>genericra</resource-adapter-mid>
        <activation-config>
          <activation-config-property>
            <activation-config-property-name>
              ConnectionFactoryJndiName
            </activation-config-property-name>
            <activation-config-property-value>
              jms/WLInboundQueueFactory
            </activation-config-property-value>
          </activation-config-property>
          <activation-config-property>
            <activation-config-property-name>
              DestinationJndiName
            </activation-config-property-name>
            <activation-config-property-value>
              jms/WLInboundQueue
            </activation-config-property-value>
          </activation-config-property>
        </activation-config>
      </mdb-resource-adapter>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

where:

The `<resource-adapter-mid>genericra</resource-adapter-mid>` element is used to specify the resource adapter and resource adapter configurations that was deployed in the "Create a Resource Adapter Configuration for GenericJMSRA to Work With WebLogic JMS" on page 44 instructions. It is recommended you stick to `genericra` as is used here.

The `activation-config` element in `glassfish-ejb-jar.xml` is the one which defines how and where the MDB receives messages, as follows:

- The `ConnectionFactoryJndiName` property must be set to the JNDI name of the connection factory in the WebLogic JNDI store that will be used to receive messages. Therefore, replace `jms/WLInboundQueueFactory` in the example above with the JNDI name used in your environment.

- The `DestinationJndiName` property must be set to the JNDI name of the destination (the queue or topic from which messages will be consumed) in the WebLogic JNDI store. Therefore, replace `jms/WLInboundQueue` in the example above with the JNDI name used in your environment.

For a description of all the ActivationSpec properties that are relevant for WebLogic JMS, see the "Configuration Reference of GenericJMSRA Properties for WebLogic JMS" on page 51.

Make sure to use the appropriate WebLogic administration tools, such as the WebLogic Administration Console or the WebLogic Scripting Tool (WLST). For more information, see Configure Messaging in the *WebLogic Server Administration Console Online Help* and the WebLogic Server WLST Online and Offline Command Reference.

Accessing Connections and Destinations Directly

When configuring a MDB to consume messages from WebLogic JMS your code does not need to access the WebLogic JMS connection factory and destination directly. You simply define them in the activation configuration, as shown in "Configuring an MDB to Receive Messages from WebLogic JMS" on page 45. However when configuring an MDB to send messages, or when configuring a EJB, Servlet, or application client to either send or receive messages, your code needs to obtain these objects using a JNDI lookup.

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 instruction 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 .

---

1 Looking up the connection factory and destination

The following code looks up a connection factory with the JNDI name `jms/QCFactory` and a queue with the name`jms/outboundQueue` from the GlassFish Server JNDI store:

```
Context initialContect  = new InitialContext();
QueueConnectionFactory queueConnectionFactory = (QueueConnectionFactory)
     jndiContext.lookup("java:comp/env/jms/MyQCFactory");
Queue queue = (Queue) jndiContext.lookup("java:comp/env/jms/outboundQueue");
```

Note that the resources used are GlassFish Server resources, not WebLogic JMS resources. For every connection factory or destination that you want to use in the WebLogic JMS JNDI store, you need to create a corresponding connection factory or destination in the GlassFish Server JNDI store and configure the GlassFish Server object to point to the corresponding WebLogic JMS object.

2 Declaring the connection factory and destination

In accordance with standard Java EE requirements, these resources need to be declared in the deployment descriptor for the MDB, EJB or other component. For example, for a session bean, configure the `ejb-jar.xml` with `<resource-env-ref>` elements, as follows:

```
<ejb-jar>
  <enterprise-beans>
    <session>
      . . .
      <resource-env-ref>
        <resource-env-ref-name>jms/QCFactory</resource-env-ref-name>
        <resource-env-ref-type>javax.jms.QueueConnectionFactory</resource-env-ref-type>
      </resource-env-ref>
      <resource-env-ref>
        <resource-env-ref-name>jms/outboundQueue</resource-env-ref-name>
        <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
      </resource-env-ref>
```

3 Create a Connector Connection Pool and Connector Resource by entering the following `asadmin` commands, both all in one line:

In order to configure a JMS Connection Factory using GenericJMSRA, a Connector connection pool and resource need to be created in GlassFish Server using names that map to the corresponding connection factory in the WebLogic JNDI store.

```
asadmin create-connector-connection-pool --host localhost --port 4848
   --raname genericra --connectiondefinition javax.jms.QueueConnectionFactory
   --target server --transactionsupport LocalTransaction
   --property ConnectionFactoryJndiName=jms/WLOutboundQueueFactory
     qcpool

asadmin create-connector-resource --host localhost --port 4848
    --poolname qcpool --target server jms/QCFactory
```

These `asadmin` commands together creates a connection factory in GlassFish Server and its corresponding connection pool.

- The connection pool has the JNDI name `jms/WLoutboundQueueFactory` and obtains connections from a connection pool named `qcpool`.
- The connection pool `qcpool` uses the resource adapter `genericra` and contains objects of type `javax.jms.QueueConnectionFactory`.
- The `transactionsupport` argument is set to `LocalTransaction`, which specifies that the connection will be used in local transactions only. You can also specify `NoTransaction`. However, the default setting of `XATransaction` cannot be used. For more information, see "Limitations When Using GenericJMSRA with WebLogic JMS" on page 49.
- The connection pool is configured with the properties specified using the `properties` argument; multiple properties are configured as a colon-separated list of name-value pairs. Only one property is configured in this example, as follows:

```
ConnectionFactoryJndiName=jms/WLOutboundQueueFactory
```

The `ConnectionFactoryJndiName` property *must* be set to the JNDI name of the corresponding connection factory in the WebLogic JMS JNDI store. Therefore, replace `jms/WLOutboundQueueFactory` in the example above with the JNDI name used in your environment.

- For a description of the `ManagedConnectionFactory` properties that are relevant for WebLogic JMS, see the "Configuration Reference of GenericJMSRA Properties for WebLogic JMS" on page 51.

4 Create a destination object that refers to a corresponding WebLogic JMS destination by entering the following `asadmin` command, all in one line:

```
asadmin create-admin-object --host localhost --port 4848 --target server
    --restype javax.jms.Queue --property DestinationJndiName=jms/WLOutboundQueue
    --raname genericra jms/outboundQueue
```

This `asadmin` command creates a destination in GlassFish Server.

- The destination has the JNDI name `jms/outboundQueue`, uses the resource adapter `genericra`, and is of type `javax.jms.Queue`.
- The destination is configured with the properties specified using the `properties` argument; multiple properties are configured as a colon-separated list of name-value pairs. Only one property is configured in this example, as follows:

  ```
  DestinationJndiName=jms/WLOutboundQueue
  ```

  The `DestinationJndiName` property *must* be set to the JNDI name of the corresponding destination in the WebLogic JMS JNDI store. Therefore, replace `jms/WLOutboundQueue` in the example above with the JNDI name used in your environment.
- For a description of the destination properties that are relevant for WebLogic JMS, see the "Configuration Reference of GenericJMSRA Properties for WebLogic JMS" on page 51.

Limitations When Using GenericJMSRA with WebLogic JMS

Due to the nature of the WebLogic T3 Thin Client there are a number of limitations in the way in which it can be used with GenericJMSRA.

No Support for XA Transactions

WebLogic JMS does not support the optional JMS "Chapter 8" interfaces for XA transactions in a form suitable for use outside of WebLogic Server. Therefore, the GenericJMSRA configuration must have the `SupportsXA` property set to `-false`. This has a number of implications for the way in which applications may be used, as described in this section.

Using a MDB to Receive Messages: Container-managed Transactions (CMT)

- If container-managed transactions are used, the transactional attribute of a MDB should be set to `NotSupported`. No transaction will be started. Messages will be received in a non-transacted session with an *acknowledgeMode* of `AUTO_ACKNOWLEDGE`.

- A transactional `Required` attribute should not be used; otherwise, MDB activation will fail with an exception: `javax.resource.ResourceException: MDB is configured to use container managed transaction. But SupportsXA is configured to false in the resource adapter.`

The remaining transactional attributes are normally considered inappropriate for use with a MDB. If used, the following behavior will occur:

- If the transactional attribute is `RequiresNew`, then MDB activation will fail with an exception: `javax.resource.ResourceException: MDB is configured to use container managed transaction But SupportsXA is configured to false in the resource adapter.`
- If the transactional attribute is `Mandatory`, the MDB can be activated but a `TransactionRequiredException` (or similar) will always be thrown by the server.
- If the transactional attribute is `Supports`, then no transaction will be started and the MDB will work as if `NotSupported` had been used.
- If the transactional attribute is `Never`, then no transaction will be started and the MDB will work as if `NotSupported` had been used.

Using a MDB to Receive Messages: Bean-managed Transactions (BMT)

- If bean-managed transactions are configured in accordance with the EJB specification any `UserTransaction` started by the bean will have no effect on the consumption of messages.
- Messages will be received in a non-transacted session with an *acknowledgeMode* of `AUTO_ACKNOWLEDGE`.

Accessing Connections and Destinations Directly - Container-managed Transactions (CMT)

When accessing connections directly (such as when sending messages from a MDB or an EJB) and container-managed transactions are being used, the connection pool's `transaction-support` property should be set to either `LocalTransaction` or `NoTransaction`. If the default value of `XATransaction` is used, an exception will be thrown at runtime when `createConnection()` is called. This is the case irrespective of the transactional attribute of the MDB or EJB. Note that MDBs must have their transactional attribute set to `NotSupported` as specified above; whereas, an EJB can use any transactional attribute.

If there is no transaction in progress within the bean method (for example, `notSupported` is being used) then it does not make any difference whether the connection pool's `transaction-support` property is set to `LocalTransaction` or `NoTransaction`; the transactional behavior will be determined by the arguments to `createSession()`. If you want the outbound message to be sent without a transaction, call `createSession(false, ...)`. If you want the outbound message to be sent in a local transaction call `createSession(true, Session.SESSION_TRANSACTED)`, remembering to call `session.commit()` or `session.rollback()` after the message is sent.

If there is a transaction in progress within the bean method (which will only be possible for EJBs), then setting the connection pool's `transaction-support` property to `LocalTransaction` or `NoTransaction` gives different results:

- If it is set to `NoTransaction` then a non-transacted session will be used.
- If it is set to `LocalTransaction` then a (local, non-XA) transacted session will be used, which will be committed or rolled back when the `UserTransaction` is committed or rolled back. In this case, calling `session.commit()` or `session.rollback()` will cause an exception.

No Support for Redelivery Limits and Dead Message Queue

Due to the lack of XA support when using WebLogic JMS, there is no support for GenericJMSRA's dead message queue feature, in which a message that has been redelivered to a MDB a defined number of times is sent to a dead message queue.

Limited Support for Asynchronous Receipt of Messages In a MDB

WebLogic JMS does not support the optional JMS "Chapter 8" interfaces for "Concurrent Processing of a Subscription's Messages" (that is, `ServerSession`, `ServerSessionPool` and `ConnectionConsumer`) in a form suitable for use outside of WebLogic Server. Therefore, the generic JMSRA configuration must set the property `DeliveryType` to `Synchronous`.

This affects the way in which MDBs consume messages from a queue or topic as follows:

- When messages are being received from a queue, each MDB instance will have its own session and consumer, and it will consume messages by repeatedly calling `receive(timeout)`. This allows the use of a pool of MDBs to process messages from the queue.
- When messages are being received from a topic, only one MDB instance will be used irrespective of the configured pool size. This means that a pool of multiple MDBs cannot be used to share the load of processing messages, which may reduce the rate at which messages can be received and processed.

  This restriction is a consequence of the semantics of synchronously consuming messages from topics in JMS: In the case of non-durable topic subscriptions, each consumer receives a copy of all the messages on the topic, so using multiple consumers would result in multiple copies of each message being received rather than allowing the load to be shared among the multiple MDBs. In the case of durable topic subscriptions, only one active consumer is allowed to exist at a time.

Configuration Reference of GenericJMSRA Properties for WebLogic JMS

The tables in this section list the properties that need to be set to configure the resource adapter and any activation specs, managed connections, and other administered objects that are

relevant only when using GenericJMSRA to communicate with WebLogic JMS. For a complete list of properties, see the comprehensive table in "GenericJMSRA Configuration Properties" on page 36

Resource Adapter Properties

These properties are used to configure the resource adapter itself when it is deployed, and can be specified using the `create-resource-adapter-config` command.

Property Name	*Required Value*	*Description*
`SupportsXA`	`false`	Specifies whether the JMS client supports XA transactions. Set to `false` for WebLogic JMS.
`DeliveryType`	`Synchronous`	Specifies whether an MDB should use a `ConnectionConsumer` (Asynchronous) or `consumer.receive()` (Synchronous) when consuming messages. Set to `Synchronous` for WebLogic JMS.
`ProviderIntegrationMode`	`jndi`	Specifies that connection factories and destinations in GlassFish's JNDI store are configured to refer to connection factories and destinations in WebLogic's JNDI store. Set to `jndi` for WebLogic JMS.
`JndiProperties`	`java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory, java.naming.provider.url=t3://localhost:7001,java.naming.factory.url.pkgs=weblogic.corba.client.naming (replace **localhost:7001** with the host:port of WebLogic Server)`	JNDI properties for connect to WebLogic JNDI, specified as comma-separated list of name-value pairs without spaces.

Property Name	*Required Value*	*Description*
`UserName`	`Name of the WebLogic JMS user`	User name to connect to WebLogic JMS. 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 WebLogic JMS user`	Password to connect to WebLogic JMS. The password can be overridden in `ActivationSpec` and `ManagedConnection`.
`LogLevel`	`Desired log level of JDK logger`	Used to specify the level of logging.

Connection Factory Properties

`ManagedConnectionFactory` objects are created in the GlassFish Server JNDI store using the Administration Console or the `asadmin connector-connection-pool` command. All the properties that can be set on a resource adapter configuration can be overridden by setting them on a destination object. The properties specific to `ManagedConnectionFactory` objects are listed in the following table.

Property Name	*Valid Value*	*Default Value*	*Description*
`ClientId`	A valid client ID	None	`ClientID` as specified by JMS 1.1 specification.
`ConnectionFactory JndiName`	A valid JNDI Name	None	JNDI name of connection factory in the GlassFish Server JNDI store. This connection factory should be configured to refer to the physical connection factory in the WebLogic JNDI store.
`ConnectionValidation Enabled`	`true` or `false`	FALSE	If set to `true`, the resource adapter will use an exception listener to catch any connection exception and will send a `CONNECTION_ERROR_OCCURED` event to GlassFish Server.

Destination Properties

Destination (queue or topic) objects are created in the GlassFish Server JNDI store using the Administration Console or the `asadmin connector-admin-object` command. All the properties that can be set on a resource adapter configuration can be overridden by setting them on a destination object. The properties specific to destination objects are listed in the following table.

Property Name	*Valid Value*	*Default Value*	*Description*
`DestinationJndiName`	A valid JNDI name	None	JNDI name of the destination object in the GlassFish Server JNDI store. This destination object should be configured to refer to the corresponding physical destination in the WebLogic JNDI store.

ActivationSpec Properties

An ActivationSpec is a set of properties that configures a MDB. It is defined either in the MDB's Sun-specific deployment descriptor `sun-ejb-jar.xml` using `activation-config-property` elements or in the MDB itself using annotation. All the resource adapter properties listed in the table above can be overridden in a ActivationSpec. Additional properties available only to a ActivationSpec are given below.

Property Name	*Valid Value*	*Default Value*	*Description*
`MaxPoolSize`	An integer	8	Maximum size of server session pool internally created by the resource adapter for achieving concurrent message delivery. This should be equal to the maximum pool size of MDB objects. Only used for queues; ignored for topics, when a value of 1 is always used.
`Subscription Durability`	`Durable` or `Non-Durable`	`Non-Durable`	Only used for topics. Specifies whether the subscription is durable or non-durable.
`SubscriptionName`		None	Only used for topics when `SubscriptionDurability` is `Durable`. Specifies the name of the durable subscription.
`MessageSelector`	A valid message selector	None	JMS message selector.
`ClientID`	A valid client ID	None	JMS ClientID.

Property Name	*Valid Value*	*Default Value*	*Description*
`ConnectionFactory JndiName`	A valid JNDI Name	None	JNDI name of connection factory in the GlassFish Server JNDI store. This connection factory should be configured to refer to the physical connection factory in the WebLogic JNDI store.
`DestinationJndiName`	A valid JNDI Name	None	JNDI name of destination in the GlassFish Server JNDI store. This destination should be configured to refer to the physical destination in the WebLogic JNDI store.
`DestinationType`	`javax.jms.Queue` or `javax.jms.Topic`	Null	Specifies whether the configured `DestinationJndiName` refers to a queue or topic.
`ReconnectAttempts`	integer	0	Number of times a reconnect will be attempted in case exception listener catches an error on connection.
`ReconnectInterval`	time in seconds	0	Interval between reconnection attempts.

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.

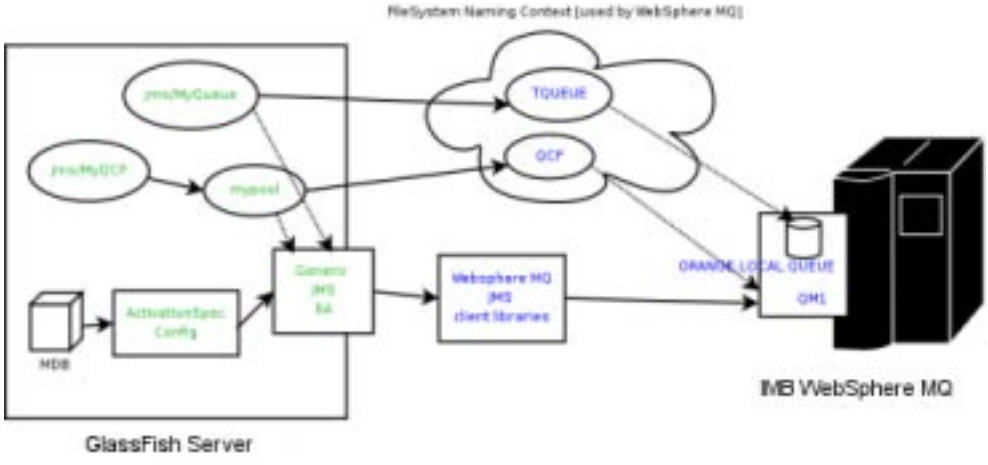
- Set the `LD_LIBRARY_PATH` environment variable to the `java/lib` directory, and then restart GlassFish Server. For example, in a Unix—based system, with WebSphere MQ installed under `/opt/mqm`, you would enter:

```
$ 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 36.

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.

Property Name	*Required Value*	*Description*
`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 Archive" on page 45.

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 57, 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 57.

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 57. 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/MyQueueConnectionFacto
    connection = connectionFactory>createConnection();
} catch (Throwable t) {
    logger>severe("PublisherBean>ejbCreate:" + "Exception: " +
    t>toString());
}
```

PART III

Appendixes