Accessing Remote Servers

Changing the provider and host to a remote system causes all JMS applications to run on the remote server. To use both the local server and one or more remote servers, create a connection factory resource with the `AddressList` property. This creates connections that access remote servers.

Troubleshooting JMS

When you start GlassFish Server, the JMS service is available but is not loaded until it is needed (for example, when you create a JMS resource). Use the `jms-ping`(1)subcommand to check if the JMS service is running or, if it is not yet running, to start it. If the`jms-ping` subcommand is unable to contact a built-in JMS service, an error message is displayed.

If you encounter problems, consider the following:

- View the GlassFish Server log file, typically located at *domain-dir*`/logs/server.log`.

  If a the log file indicates that a Message Queue broker did not respond to a message, stop the broker and then restart it.
- View the broker log, typically available at *as-install*`/domains/domain1/imq/instances/imqbroker/log/log.txt`.
- For JMS REMOTE mode, be sure to start Message Queue brokers first, then GlassFish Server.
- If all Message Queue brokers are down, it takes 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 JMS Resource Adapter to Integrate External JMS Providers

GlassFish Server supports the integration and use of external JMS Providers through the use of the Generic Resource Adapter for JMS, available at `http://genericjmsra.java.net`. This Java EE connector 1.5 resource adapter, named `genericjmsra`, can wrap the JMS client library of external JMS providers such as Oracle WebLogic JMS, IBM WebSphere MQ, Tibco EMS, and Sonic MQ among others. This allows GlassFish Server to be used with any supported JMS provider. The adapter is a `.rar` archive that can be deployed and configured using GlassFish Server administration tools.

The following topics are addressed here:

- "Configuring the Generic JMS Resource Adapter for External JMS Providers" on page 300
- "Using the Generic JMS Resource Adapter with WebLogic JMS" on page 306
- "Using the Generic JMS Resource Adapter with IBM WebSphere MQ" on page 316

Configuring the Generic JMS Resource Adapter for External JMS Providers

The generic resource adapter 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 the generic resource adapter. 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 the generic resource adapter. 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 Configure the Generic JMS Resource Adapter

Before deploying the generic resource adapter, 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 `genericjmsra.rar` archive from the "Generic Resource Adapter for JMS" project page: `http://genericjmsra.java.net/`**

2. **Deploy the generic resource adapter the same way you would deploy a connector module.**

3. **Create a connector connection pool.**

   See "To Create a Connector Connection Pool" on page 239.

4. **Create a connector resource.**

   See "To Create a Connector Resource" on page 242.

5. **Create an administered object resource.**

   See "To Create an Administered Object" on page 254.

6. **Make the following changes to the security GlassFish Server policy files:**

   - Modify the *domain-dir*`/config/server.policy` file to add the following:

     ```
     java.util.logging.LoggingPermission "control"
     ```

   - Modify the *as-install*`/lib/appclient/client.policy` file to add permission:

```
javax.security.auth.PrivateCredentialPermission
"javax.resource.spi.security.PasswordCredential ^ \"^\"","read":
```

Generic JMS Resource Adapter Properties

The following table presents the properties to be used while creating 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.
`ConnectionFactoryClassName`	Name of the class available in the application server classpath, for example: `com.sun.messaging.ConnectionFactory`	None	Class name of `javax.jms.ConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is `javabean`.
`QueueConnectionFactoryClassName`	Name of the class available in the application server classpath, for example: `com.sun.messaging.QueueConnectionFactory`	None	Class name of `javax.jms.QueueConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is `javabean`.
`TopicConnectionFactoryClassName`	Name of the class available in the application server classpath , for example: `com.sun.messaging.TopicConnectionFactory`	None	Class name of `javax.jms.TopicConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is specified as `javabean`.
`XAConnectionFactoryClassName`	Name of the class available in application server classpath , for example: `com.sun.messaging.XAConnectionFactory`	None	Class name of `javax.jms.ConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is specified as `javabean`.
`XAQueueConnectionFactoryClassName`	Name of the class available in application server classpath , for example: `com.sun.messaging.XAQueueConnectionFactory`	None	Class name of `javax.jms.XAQueueConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is specified as `javabean`.

Property Name	*Valid Values*	*Default Value*	*Description*
`XATopicConnection FactoryClassName`	Name of the class available in application server classpath , for example: `com.sun.messaging. XATopicConnectionFactory`	None	Class name of `javax.jms. XATopicConnectionFactory` implementation of the JMS client. Used if `ProviderIntegrationMode` is `javabean`.
`TopicClassName`	Name of the class available in application server classpath , for example: `com.sun.messaging.Topic`	None	Class Name of `javax.jms.Topic` implementation of the JMS client. Used if `ProviderIntegrationMode` is `javabean`.
`QueueClassName`	Name of the class available in application server classpath , for example: `com.sun.messaging.Queue`	None	Class Name of `javax.jms.Queue` implementation of the JMS client. 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 IBM MQ Series 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.

`ManagedConnectionFactory` 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.

Administered Object Resource Properties

Properties in this section are specified when an administered object resource is created. All the resource adapter properties can be overridden in an administered resource object. Additional properties available only in the administered object resource 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 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`.
`ReconnectAttempts`	integer		Number of times a reconnect will be attempted in case exception listener catches an error on connection.
`ReconnectInterval`	time in seconds		Interval between reconnects.

Using the Generic JMS Resource Adapter with WebLogic JMS

You can configure the Generic JMS Resource Adapter 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 the Generic JMS Resource Adapter with WebLogic JMS" on page 313.

The following topics are addressed here:

- "Deploy the WebLogic Thin T3 Client JAR in GlassFish Server" on page 307
- "Configure WebLogic JMS Resources" on page 307
- "Create the Generic JMS Resource Adapter Configuration" on page 308
- "Deploy the Generic Resource Adapter " on page 309
- "Configuring an MDB to Receive Messages" on page 310
- "Accessing Connections and Destinations Directly" on page 311
- "Limitations When Using the Generic JMS Resource Adapter with WebLogic JMS" on page 313

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*`/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

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 the Generic JMS Resource Adapter Configuration

Before deploying the generic JMS resource adapter, 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
--user <adminname> --password <admin password-####>
--property SupportsXA=false:DeliveryType=Synchronous:ProviderIntegrationMode=jndi:
  UserName=weblogic:Password=###: JndiProperties=java.naming.factory.url.pkgs
  =weblogic.corba.client.naming,java.naming.factory.initial
  =weblogic.jndi.WLInitialContextFactory, java.naming.provider.url
  =t3://localhost:7001:LogLevel=finest 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:

Note – The tables in this section describe the GenericJMSRA properties that are relevant only for WebLogic JMS. For a complete list of properties, see the comprehensive table in "Generic JMS Resource Adapter Properties" on page 301.

Property Name	*Required Value*	**Default Value**	*Description*
`SupportsXA`	`false`	false	Specifies whether the JMS client supports XA transactions. Set to `false` for WebLogic JMS.
`DeliveryType`	`Synchronous`	Asynchronous	Specifies whether an MDB should use a `ConnectionConsumer` (Asynchronous) or `consumer.receive()` (Synchronous) when consuming messages. Set to `Synchronous` for WebLogic JMS.
`ProviderIntegrationMode`	`jndi`	javabean	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.

Property Name	*Required Value*	**Default Value**	*Description*
`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 host:port of WebLogic)	None	JNDI properties for connect to WebLogic JNDI, specified as a comma-separated list of name=value pairs.
`UserName`	`Name of the WebLogic JMS user`	None	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`	None	Password to connect to WebLogic JMS. The password can be overridden in `ActivationSpec` and `ManagedConnection`.
`LogLevel`	`Desired log level of JDK logger`	None	Used to specify the level of logging.

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

Tip – When using `asadmin` you need to escape each = and any : characters by prepending a backward slash \.

▼ Deploy the Generic Resource Adapter

The GenericJMSRA archive is available on the GlassFish Server Update Center. It is also available out-of-the-box with the Sun Java System Application Server 9.1 Platform Edition, application server. Therefore, you can also use the bundled resource adapter as well in the step below.

1 **Download the GenericJMSRA archive from the "Generic Resource Adapter for JMS" java.net project page: `http://genericjmsra.java.net/`**

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

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 need to 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 the Generic JMS Resource Adapter with WebLogic JMS" on page 313.

---

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 the Generic JMS Resource Adapter Configuration" on page 308 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.

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" on page 310. 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 will need to obtain these objects using a JNDI lookup.

1 Looking up the connection factory and destination

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

```
Context initialContect  = new InitialContext();
QueueConnectionFactoryqueueConnectionFactory = (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 `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 the 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 that has the JNDI name `jms/WLoutboundQueueFactory` and obtains connections from a connection pool `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 `NolTransaction`. However, the default setting of `XATransaction` cannot be used. For more information, see "Limitations When Using the Generic JMS Resource Adapter with WebLogic JMS" on page 313.
- 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.

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 command creates a destination that has the JNDI name `jms/outboundQueue`, which uses the resource adapter `genericra`, and is of type `javax.jms.Queue`, which is configured with the properties specified using the `--properties` argument. The `DestinationJndiName` property *must* be set to the JNDI name of the corresponding destination in the WebLogic JMS JNDI store Therefore, replace `jms/outboundQueue` in the example above with the JNDI name used in your environment.

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 239 and "To Create a Connector Resource" on page 242.

Limitations When Using the Generic JMS Resource Adapter 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 the 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 generic resource adapter 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 the Generic JMS Resource Adapter'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.