JMS Binding Component User's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 821–0452–10
September 2009

090917@22749

Contents

Using the JMS Binding Component 5
JMS Binding Component Overview 5
JMS Binding Component Features 7
JMS Binding Component Scenarios 7
    Outbound Scenario 7
    Inbound Scenario 8
    XA Scenario 9
    Outbound InOut Exchange Scenario 10
    Inbound InOut Exchange Scenario 11
    On Demand Receive Scenario 12
Creating JMS-Based Concrete WSDL Documents 13
  ▼ To Create a JMS-Based Concrete WSDL Document from the New WSDL Document Wizard 14
  ▼ To Create a JMS-Based Concrete WSDL Document from the New File Wizard 14
    Request Connection Configuration 15
    Request Consumer Configuration 16
    Request Publisher Configuration 17
    Response Destination Configuration 19
    Response Consumer Configuration 20
    Response Publisher Configuration 21
    Advanced Configuration 22
Connecting to the JMS Server 24
    Specifying the Protocol, Server, and Port 24
    Connecting to the JMS Server by Using JNDI 25
Configuring Redelivery Handling 27
    Configuring Redelivery Handling from the Wizard 28
    Configuring Redelivery Handling by Editing the WSDL Document 29
Using the Normalized Message Properties 30
    Inbound Normalized Message Properties 30

- Outbound Normalized Message Properties 33
- General Normalized Message Properties 37

JMS WSDL Extensibility Elements 40

- JMS Connectivity Elements 41
- JMS Binding Elements 42

JMS Binding Component Clustering 49

- JMS High Availability 49
- JMS Load Balancing 50
- JMS Performance 50

Publishing and Subscribing to Multiple WebSphere Queues 52

Using the JMS Binding Component

The topics listed here provide information about how to use the Java Message Service (JMS) Binding Component.

If you have any questions or problems, see the Java CAPS web site at `http://goldstar.stc.com/support`.

What You Need to Know

- "JMS Binding Component Overview" on page 5
- "JMS Binding Component Features" on page 7
- "JMS Binding Component Scenarios" on page 7

What You Need to Do

- "Creating JMS-Based Concrete WSDL Documents" on page 13
- "Connecting to the JMS Server" on page 24
- "Configuring Redelivery Handling" on page 27
- "Using the Normalized Message Properties" on page 30
- "JMS WSDL Extensibility Elements" on page 40
- "JMS Binding Component Clustering" on page 49

JMS Binding Component Overview

The JMS Binding Component includes NetBeans design-time components and a Java Business Integration (JBI) runtime.

The design-time component is a NetBeans module that plugs into the NetBeans project system. The runtime portion provides interfaces for accepting messages from the JBI runtime, or using interfaces provided by JBI for communicating with the JBI runtime.

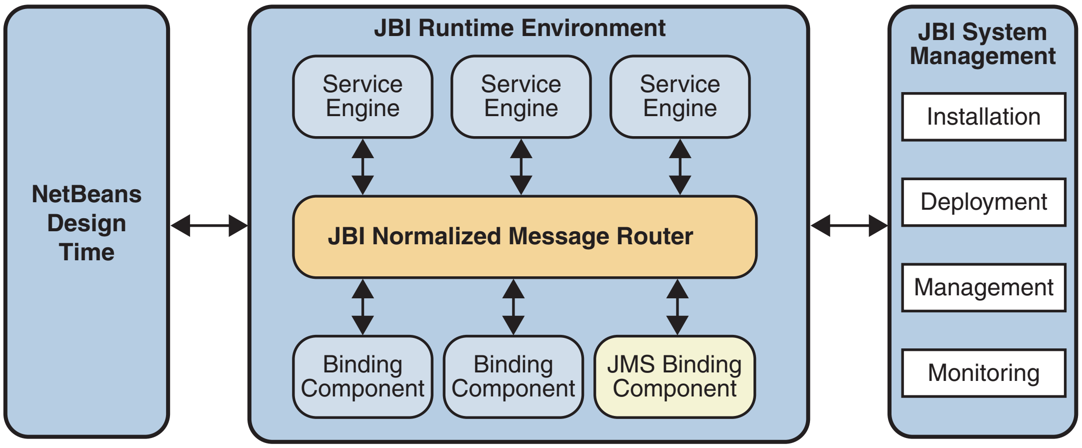
The flexible and extensible architecture of JBI enables components such as the JMS Binding Component to use a messaging model that separates service consumers from service providers. This messaging model is defined using Web Services Description Language (WSDL), which describes the operations exposed by the binding component. WSDL is also used to define transport-level bindings for the abstract service operations. During design time, you configure the JMS Binding Component by using the JMS WSDL extensibility elements.

The JBI specification also includes a model that describes the exchange of messages between message consumers and message providers.

In an outbound message flow, the JMS Binding Component acts as a service provider. The JMS Binding Component receives a normalized message from the Normalized Message Router (NMR), converts that message to a JMS message, and then sends the message to a JMS destination.

In an inbound message flow, the JMS Binding Component acts as a proxy consumer. The JMS Binding Component converts the JMS message that it receives from a JMS service to a normalized message, and then sends the normalized message as part of the message exchange to another component as a service request.

The following diagram illustrates the relationship between the JMS Binding Component in the JBI Runtime Environment, the design-time (NetBeans) and other JBI system management components.



JMS Binding Component Features

The features of the JMS Binding Component include:

- Sending inbound and outbound JMS messages (text, binary, XML)
- Auto acknowledgement of received inbound JMS messages
- Receiving batch inbound JMS messages
- Receiving inbound JMS messages through a selection filter
- Mapping inbound JMS properties to NMR message parts
- Mapping NMR message parts to outbound JMS properties
- Normalizing and denormalizing JMS messages and WSDL message parts
- Invoking request/response operations
- JNDI lookups of connection factories
- sync and cc concurrency modes

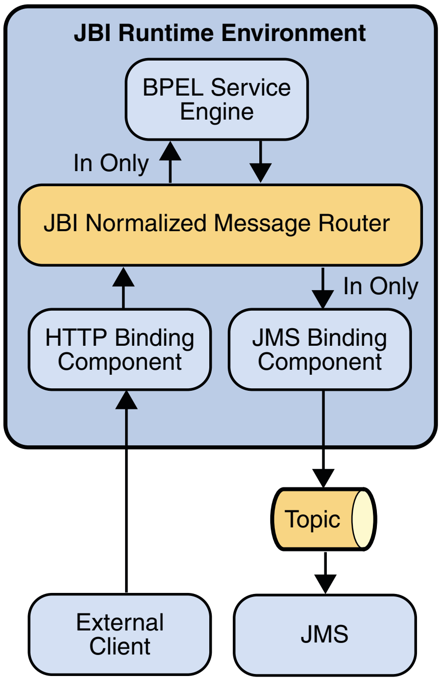
JMS Binding Component Scenarios

The following scenarios show how the JMS Binding Component can consume messages and provide messages.

- "Outbound Scenario" on page 7
- "Inbound Scenario" on page 8
- "XA Scenario" on page 9
- "Outbound InOut Exchange Scenario" on page 10
- "Inbound InOut Exchange Scenario" on page 11
- "On Demand Receive Scenario" on page 12

Outbound Scenario

The following diagram illustrates an outbound scenario.

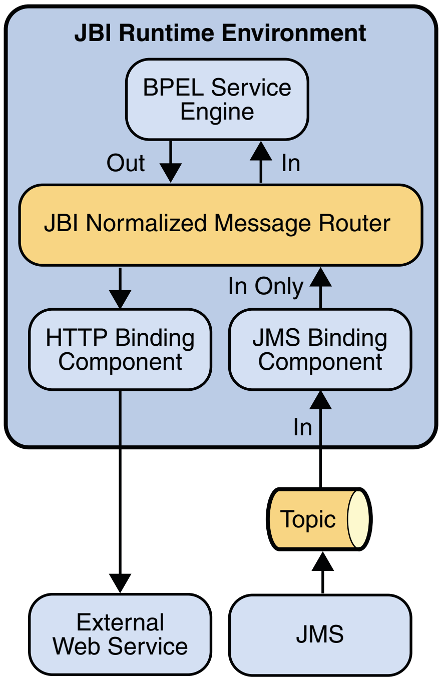


The HTTP Binding Component consumes an external web service client message. The HTTP Binding Component sends an `InOnly` message exchange to the BPEL Service Engine. The BPEL Service Engine performs a simple translation of the input message to an output message. The BPEL Service Engine sends the message exchange to the JMS Binding Component.

The JMS Binding Component converts the normalized message in the `InOnly` message exchange to a JMS text message. The JMS Binding Component sends the text message to a configured JMS topic.

Inbound Scenario

The following diagram illustrates an outbound scenario.

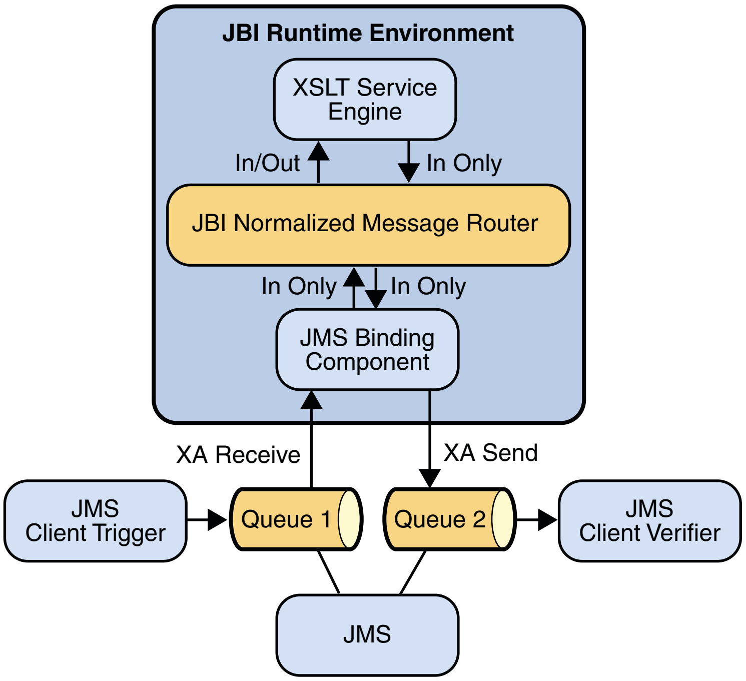


The JMS Binding Component receives a JMS message from a JMS topic. The JMS Binding Component normalizes the JMS message to an `InOnly` message exchange. The JMS Binding Component sends the message exchange to the BPEL Service Engine.

The BPEL Service Engine invokes the HTTP Binding Component to send a message to a web service.

XA Scenario

The following diagram illustrates an XA scenario.



The inbound JMS Binding Component receives a text message from JMS queue 1. The JMS Binding Component starts an XA transaction and enlists its `XAResource`. The JMS Binding Component normalizes the JMS message and sends the message as a transacted `InOnly` message exchange. The message exchange contains the transactional context as a property keyed by `MessageExchange.JTA_TRANSACTION_PROPERTY_NAME`.

The XSLT Service Engine propagates the transaction in the `InOnly` message exchange that it creates and sends it to the outbound JMS Binding Component. The outbound JMS Binding Component denormalizes the message exchange to a JMS message and sends the JMS message to JMS queue 2, as part of the XA transaction created by the inbound JMS Binding Component.

Upon completion of both `InOnly` message exchanges, the inbound JMS Binding Component commits the XA transaction (thereby committing both the JMS receive and the JMS send).

The JMS Client Trigger and JMS Client Verifier are external JMS clients used to initiate the message exchanges and to verify the result. Once the transaction is committed, the JMS Client Verifier receives the expected JMS message sent by the outbound JMS Binding Component.

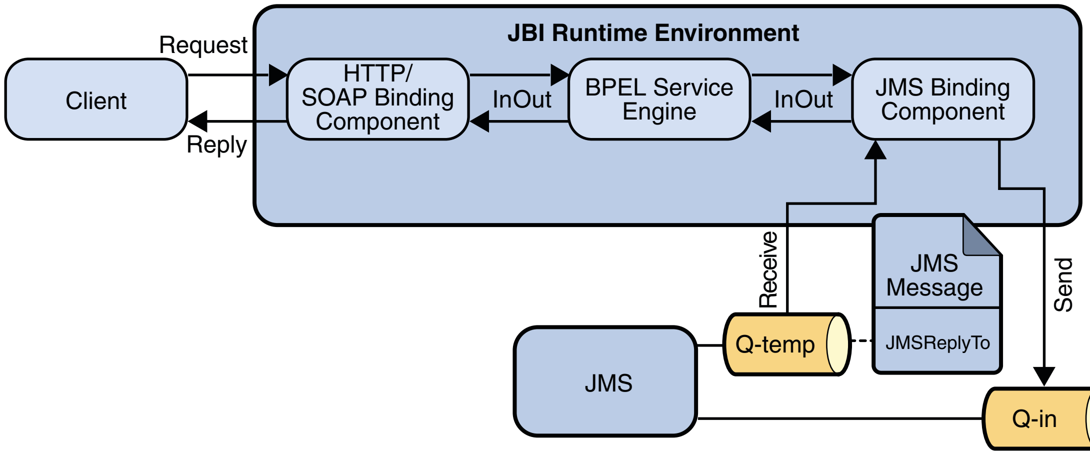
Outbound InOut Exchange Scenario

In this use case, the JMS Binding Component receives an InOut exchange from its consuming partner (for example, the BPEL Service Engine).

It denormalizes the message found in the "In" portion of the InOut message exchange to a JMS message. It then creates a temporary queue or topic, "Q-temp", and sets the JMS message property, called JMSReplyTo, with this temporary queue or topic. It sends the JMS message to the request queue "Q-in". It waits to receive the reply (a JMS message) from the temporary queue or topic.

When it receives the reply message from the temporary queue or topic, it normalizes the JMS message and sets it as the message in the "Out" portion of the InOut message exchange.

It then sends the InOut message exchange to the consuming partner. Finally, it waits for the message exchange status from the consuming partner to complete the InOut message exchange.



Inbound InOut Exchange Scenario

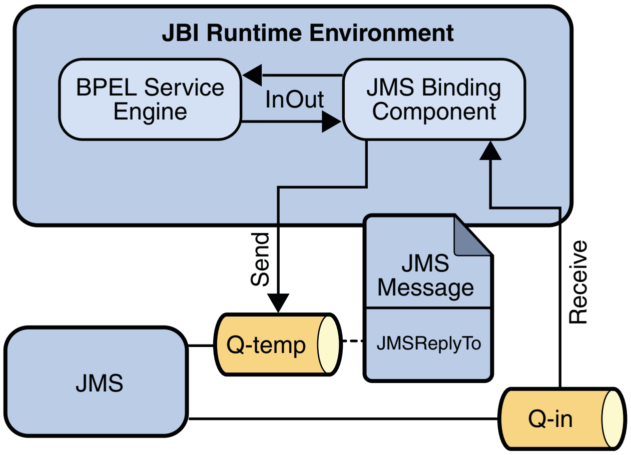
In this use case, the JMS Binding Component is an external proxy consumer for an external service consumer.

It receives a JMS message from input queue "Q-in". It creates an InOut message exchange. It normalizes the JMS message and sets it as the message in the "In" portion of the InOut message exchange. It then sends the InOut exchange to its providing partner (for example, the BPEL Service Engine). It then waits for the reply from its providing partner for the InOut exchange via the NMR.

Upon receiving the reply, it denormalizes the message found in the "Out" portion of the InOut exchange to a JMS message.

It then extracts the topic or queue, "Q-temp", found in the JMS request message's JMSReplyTo property and sends the JMS reply message to this reply queue or topic.

Finally, upon successfully sending the JMS reply message to the JMSReplyTo queue or topic, it sends back a DONE response on the InOut message exchange to the providing partner.



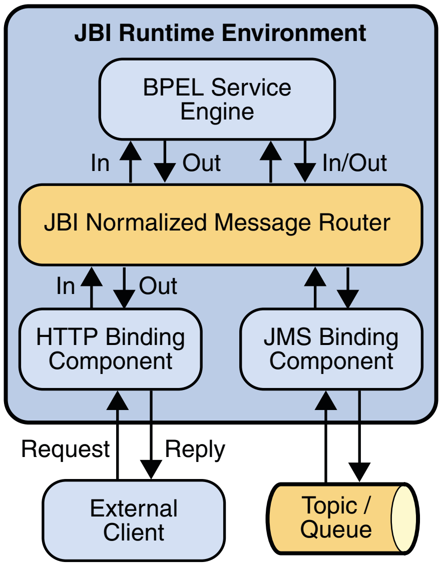
On Demand Receive Scenario

The HTTP Binding Component consumes a request message from an external web service client. The HTTP Binding Component sends an `InOnly` message exchange to the BPEL Service Engine. The BPEL Service Engine performs a simple translation of the input message to an output message. The BPEL Service Engine sends the message exchange to the JMS Binding Component.

The JMS Binding Component converts the normalized message in the `InOnly` message exchange to a JMS text message. The JMS Binding Component sends the text message to a configured JMS topic or queue.

The JMS Binding Component receives a JMS reply message from the JMS topic or queue. The JMS Binding Component normalizes the JMS message to an `InOnly` message exchange. The JMS Binding Component sends the message exchange to the BPEL Service Engine.

The BPEL Service Engine invokes the HTTP Binding Component to send a message to the external web service client.



Creating JMS-Based Concrete WSDL Documents

You can use a wizard to create a concrete WSDL document that contains JMS binding and service information.

In the initial part of the wizard, you must select one of the following types:

- **Receive** – Choose this type for scenarios in which the JMS Binding Component receives a message from a JMS destination and invokes a JBI service with the message.
- **Receive with Reply** – Choose this type for scenarios in which the JMS Binding Component receives a message from a JMS destination, invokes a JBI service, and sends the response from the service back over JMS.
- **Send** – Choose this type for scenarios in which a JBI service invokes the JMS Binding Component to send a message to a JMS destination.
- **Send and Wait for Reply** – Choose this type for scenarios in which a JBI service invokes the JMS Binding Component to send a message to a JMS destination and receive the response over JMS.
- **On Demand Receive** – Choose this type for scenarios in which a JBI service invokes the JMS Binding Component to read a message from a JMS destination.

The configuration steps that appear next depend on the type that you selected.

- "Request Connection Configuration" on page 15
- "Request Consumer Configuration" on page 16
- "Request Publisher Configuration" on page 17
- "Response Destination Configuration" on page 19

- "Response Consumer Configuration" on page 20
- "Response Publisher Configuration" on page 21
- "Advanced Configuration" on page 22

▼ To Create a JMS-Based Concrete WSDL Document from the New WSDL Document Wizard

1. **In the Projects window of the NetBeans IDE, right-click the project node and choose New ⇒ WSDL Document.**
2. **Set the WSDL Type option to Concrete WSDL Document.**
3. **Set the Binding drop-down list to JMS.**
4. **Set the Type drop-down list to the appropriate type of JMS-based WSDL document.**
5. **Click Next.**
6. **Complete the configuration steps that appear next. For a description of each step, see the following topics.**

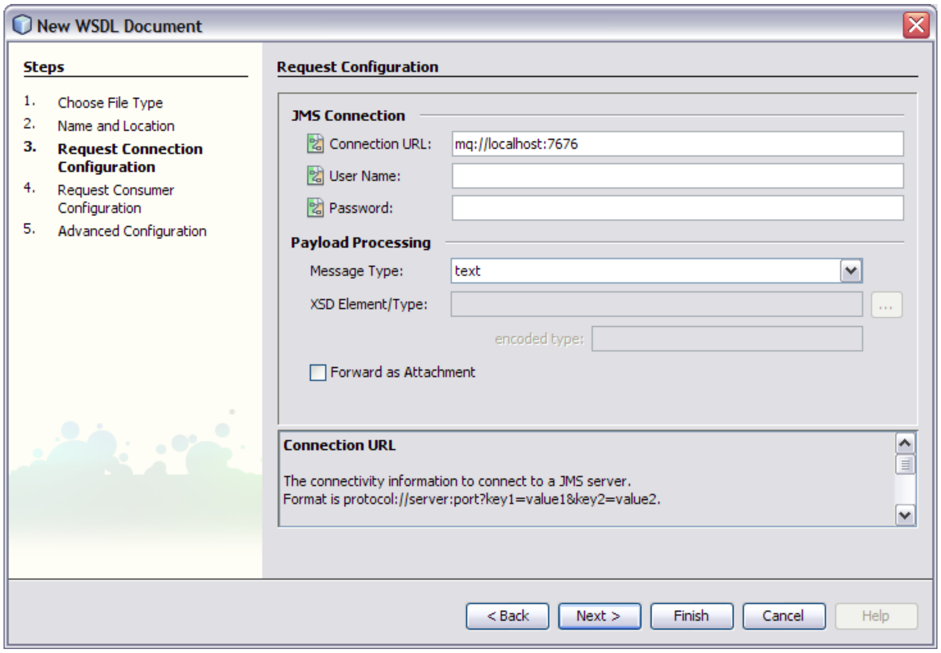
▼ To Create a JMS-Based Concrete WSDL Document from the New File Wizard

1. **In the Projects window of the NetBeans IDE, right-click the project node and choose New ⇒ Other.**
2. **In the Categories list, select ESB.**
3. **In the File Types list, select Binding.**
4. **Click Next.**
5. **Set the Binding drop-down list to JMS.**
6. **Set the Type drop-down list to the appropriate type of JMS-based WSDL document.**
7. **Click Next.**

8 Complete the configuration steps that appear next. For a description of each step, see the following topics.

Request Connection Configuration

When you are creating a JMS-based concrete WSDL document, the Request Connection Configuration step appears for all of the possible types.



Connection URL: mq://localhost:7676
User Name:
Password:
Message Type: text
XSD Element/Type:
encoded type:

Connection URL

The connectivity information to connect to a JMS server.
Format is protocol://server:port?key1=value1&key2=value2.

The JMS Connection section enables you to specify information for connecting to the JMS server.

- **Connection URL** – The URL for connecting to the JMS server.

  For more information, see "Connecting to the JMS Server" on page 24.
- **User Name** – The user name for connecting to the JMS server.
- **Password** – The password for connecting to the JMS server.

The Payload Processing section enables you to define the type of JMS messages being created and sent to the JMS destination.

- **Message Type** – Specifies whether the message payload is treated as text data, binary data, XML data, or encoded data.

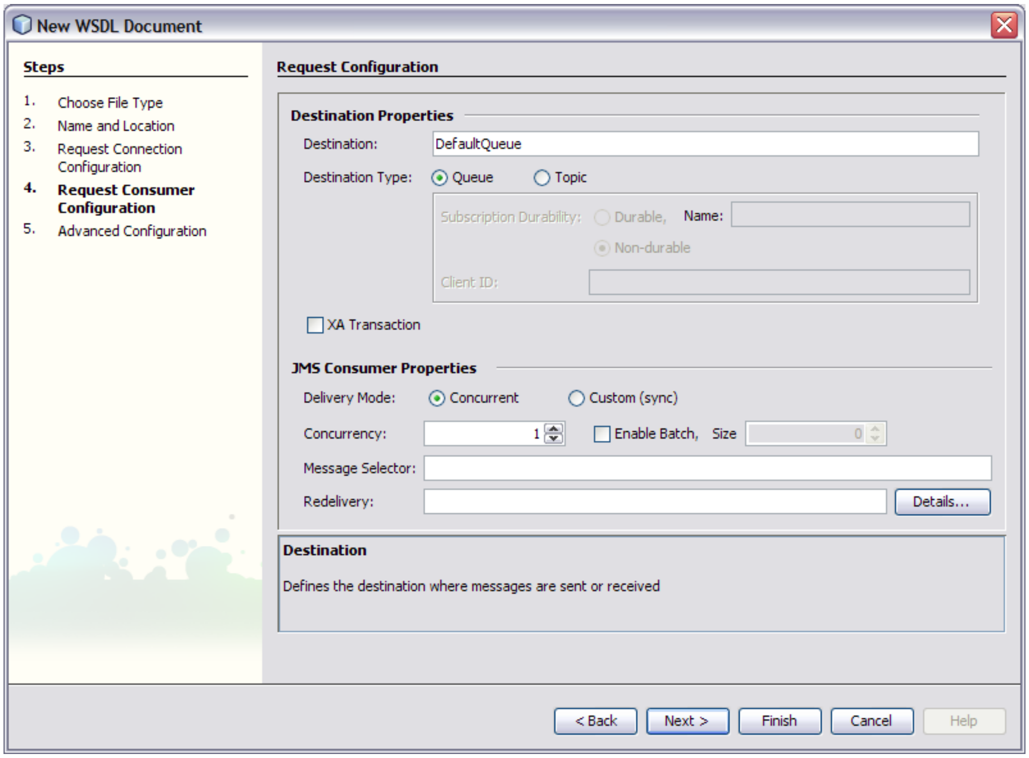
- **XSD Element/Type** – This field is enabled when the message type is XML data or encoded data. Click the ellipsis points (...) and select a user-defined XSD element.
- **Encoded Type** – This field is enabled when the message type is encoded data.
- **Forward as Attachment**– If you want to send the message data as an attachment, then select this check box.

  For binary data, the data is sent as an attachment by default.

  For XML data, sending the data as an attachment prevents the JMS Binding Component from parsing the XML. If the XML is large, then this approach can improve performance.

Request Consumer Configuration

When you are creating a JMS-based concrete WSDL document, the Request Consumer Configuration step appears when you select the Receive type.



Destination: DefaultQueue

Destination Type: [✓ Queue] Topic

Subscription Durability: Durable, [✓ Non-durable] Name:

Client ID:

XA Transaction

Delivery Mode: [✓ Concurrent] Custom (sync)

Concurrency: 1 Enable Batch, Size 0

Message Selector:

Redelivery:

Destination

Defines the destination where messages are sent or received

The Destination Properties section enables you to specify the JMS destination from which messages are received.

- **Destination** – The name of the destination.
- **Destination Type** – Indicates whether the destination is a queue or a topic.
- **Subscription Durability**. This field is enabled when the destination is a topic. Indicates whether the subscriber is durable or nondurable. Durable subscribers can survive any disconnection from the JMS server.
- **Subscription Name** – For a durable subscriber, indicates the name that is used to denote the durable subscription.
- **Client ID** – This field is enabled when the destination is a topic. Defines a unique client ID.
- **XA Transaction** – If you want to configure a two-phase commit scenario, then select this check box.

  For more information, see "XA Scenario" on page 9.

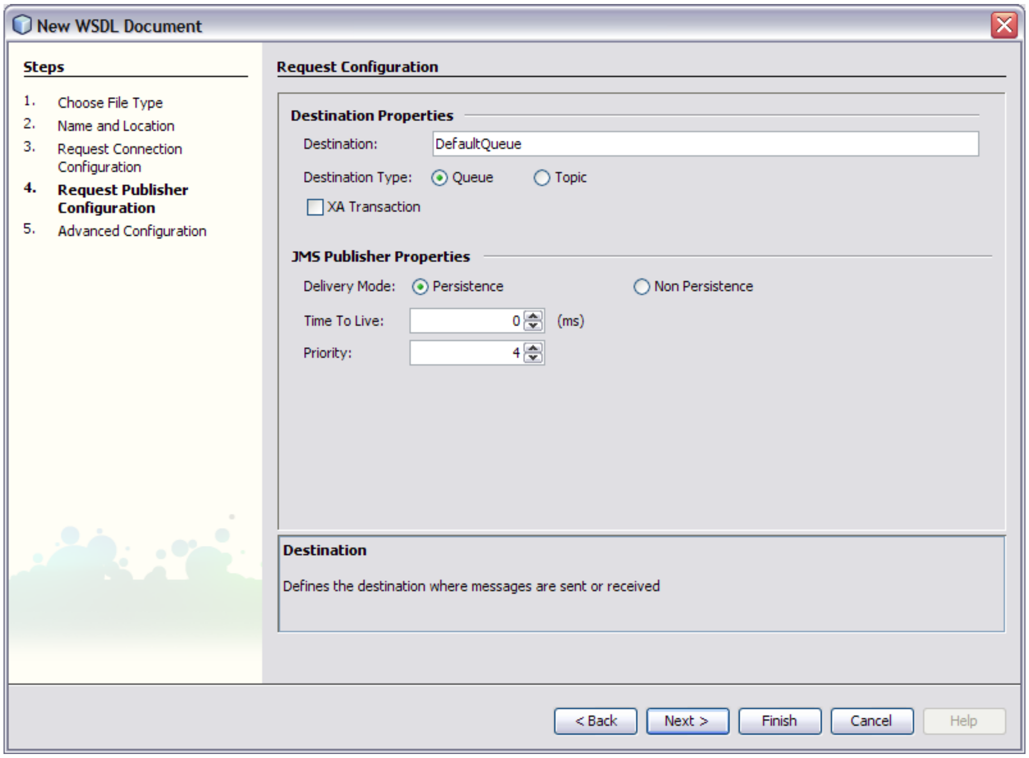
The JMS Consumer Properties section enables you to configure properties that apply only to consumer message flows.

- **Delivery Mode** – Specifies the concurrency mode.
- **Concurrency** – Indicates the maximum number of concurrent receivers.
- **Enable Batch** – If you want to receive messages in a batch, then select the Enable Batch check box and specify the size.
- **Message Selector** – Enables you to filter messages. A message selector consists of a boolean expression, such as `Age > 30`.
- **Redelivery** – Enables you to specify what actions to take when a message is repeatedly redelivered. For example, you could place the message in a dead letter queue.

  For more information, see "Configuring Redelivery Handling" on page 27.

Request Publisher Configuration

When you are creating a JMS-based concrete WSDL document, the Request Publisher Configuration step appears when you select the Send type.


New WSDL Document
Steps
1. Choose File Type
2. Name and Location
3. Request Connection Configuration
4. Request Publisher Configuration
5. Advanced Configuration
< Back
Next >
Finish
Cancel
Help

Request Configuration

Destination Properties

Destination: DefaultQueue

Destination Type: [✓ Queue] Topic

☐ XA Transaction

JMS Publisher Properties

Delivery Mode: [✓ Persistence] Non Persistence

Time To Live: 0 (ms)

Priority: 4

Destination

Defines the destination where messages are sent or received

The Destination Properties section enables you to specify the JMS destination to which messages are sent.

- **Destination** – The name of the destination.
- **Destination Type** – Indicates whether the destination is a queue or a topic.
- **XA Transaction** – If you want to configure a two-phase commit scenario, then select this check box.

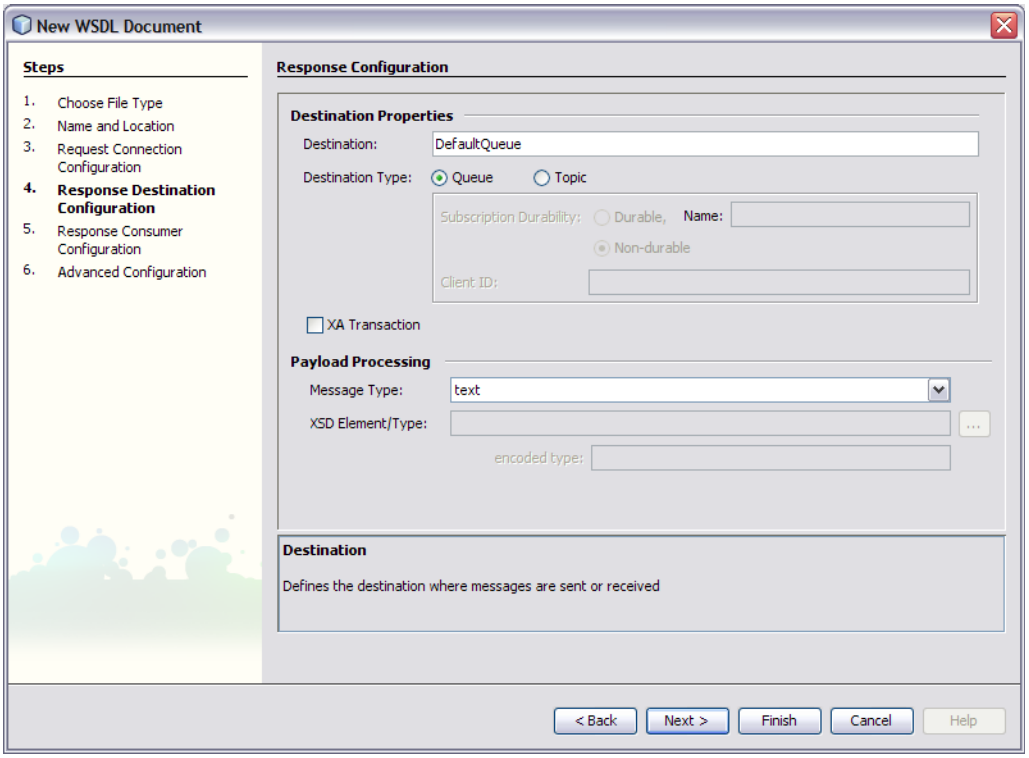
  For more information, see "XA Scenario" on page 9.

The JMS Publisher Properties section enables you to configure properties that apply only to provider message flows.

- **Delivery Mode** – Indicates whether the message is persistent or nonpersistent. Persistent messages can survive the failure of the JMS server.
- **Time To Live** – Indicates how long the message is retained (in milliseconds).
- **Priority** – Defines the message priority. The valid values are 0 through 9, where 0 is the lowest priority and 9 is the highest priority. The default value is 4.

Response Destination Configuration

When you are creating a JMS-based concrete WSDL document, the Response Destination Configuration step appears when you select the Receive with Reply type or the Send and Wait for Reply type.


New WSDL Document

Steps
1. Choose File Type
2. Name and Location
3. Request Connection Configuration
4. **Response Destination Configuration**
5. Response Consumer Configuration
6. Advanced Configuration

Response Configuration

Destination Properties

Destination: DefaultQueue

Destination Type: [✓ Queue] Topic

Subscription Durability: Durable, [✓ Non-durable] Name:

Client ID:

XA Transaction

Payload Processing

Message Type: text

XSD Element/Type:

encoded type:

Destination

Defines the destination where messages are sent or received

The Destination Properties section enables you to specify the JMS destination to which messages are sent.

- **Destination** – The name of the destination.
- **Destination Type** – Indicates whether the destination is a queue or a topic.
- **Subscription Durability** – This field is enabled when the destination is a topic. Indicates whether the subscriber is durable or nondurable. Durable subscribers can survive any disconnection from the JMS server.
- **Subscription Name** – For a durable subscriber, indicates the name that is used to denote the durable subscription.
- **Client ID** – This field is enabled when the destination is a topic. Defines a unique client ID.

- **XA Transaction** – If you want to configure a two-phase commit scenario, then select this check box.

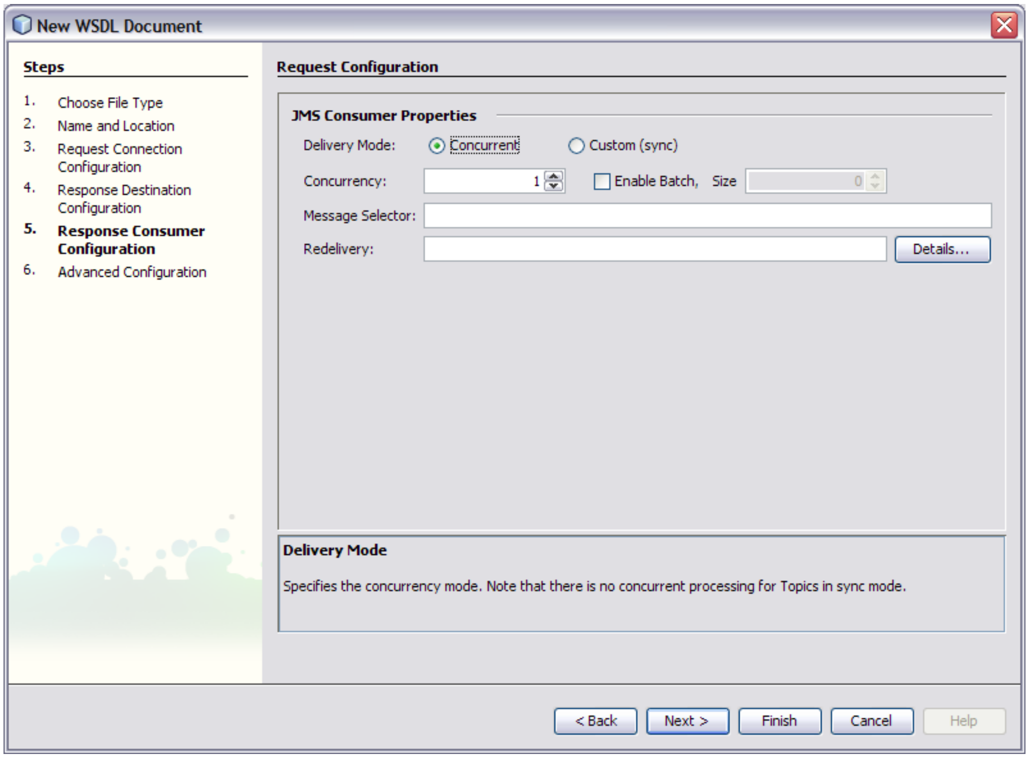
  For more information, see "XA Scenario" on page 9.

The Payload Processing section enables you to define the type of JMS messages being created and sent to the JMS destination.

- **Message Type** – Specifies whether the message payload is treated as text data, binary data, XML data, or encoded data.
- **XSD Element/Type** – This field is enabled when the message type is XML data or encoded data. Click the ellipsis points (...) and select a user-defined XSD element.
- **Encoded Type** – This field is enabled when the message type is encoded data.

Response Consumer Configuration

When you are creating a JMS-based concrete WSDL document, the Response Consumer Configuration step appears when you select the Receive with Reply type.


New WSDL Document

Steps

1. Choose File Type
2. Name and Location
3. Request Connection Configuration
4. Response Destination Configuration
5. **Response Consumer Configuration**
6. Advanced Configuration

Request Configuration

JMS Consumer Properties

Delivery Mode: [✓ Concurrent] Custom (sync)

Concurrency: 1 Enable Batch, Size 0

Message Selector:

Redelivery:

Delivery Mode

Specifies the concurrency mode. Note that there is no concurrent processing for Topics in sync mode.

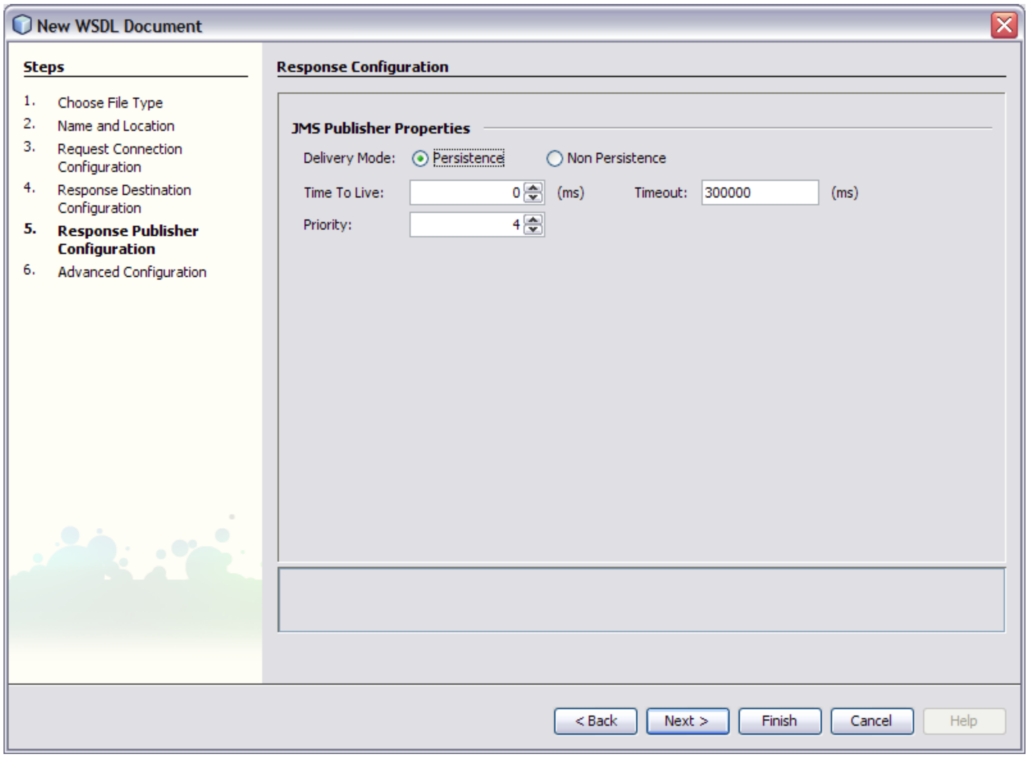
The JMS Consumer Properties section enables you to configure properties that apply only to consumer message flows.

- **Delivery Mode** – Specifies the concurrency mode.
- **Concurrency** – Indicates the maximum number of concurrent receivers.
- **Enable Batch** – If you want to receive messages in a batch, then select the Enable Batch check box and specify the size.
- **Message Selector** – Enables you to filter messages. A message selector consists of a boolean expression, such as `Age > 30`.
- **Redelivery** – Enables you to specify what actions to take when a message is repeatedly redelivered. For example, you could place the message in a dead letter queue.

  For more information, see "Configuring Redelivery Handling" on page 27.

Response Publisher Configuration

When you are creating a JMS-based concrete WSDL document, the Response Publisher Configuration step appears when you select the Send and Wait for Reply type.


New WSDL Document

Steps
1. Choose File Type
2. Name and Location
3. Request Connection Configuration
4. Response Destination Configuration
5. **Response Publisher Configuration**
6. Advanced Configuration

Response Configuration

JMS Publisher Properties

Delivery Mode: [✓ Persistence] Non Persistence

Time To Live: 0 (ms) Timeout: 300000 (ms)

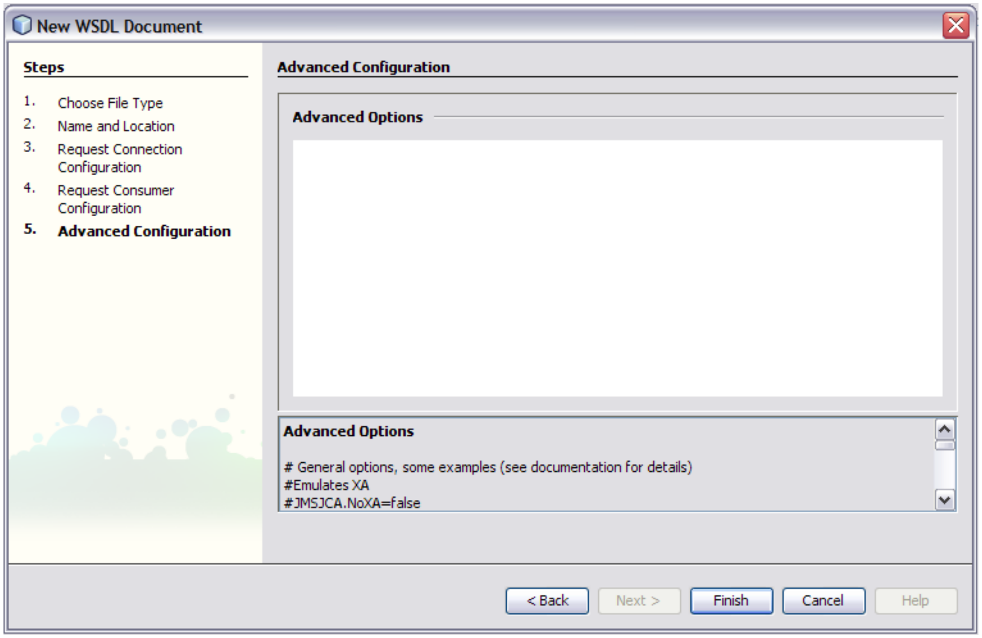
Priority: 4

The JMS Publisher Properties section enables you to configure properties that apply only to provider message flows.

- **Delivery Mode** – Indicates whether the message is persistent or nonpersistent. Persistent messages can survive the failure of the JMS server.
- **Time To Live** – Indicates how long the message is retained (in milliseconds).
- **Timeout** – The timeout in milliseconds on a message consumer receive for a reply message.
- **Priority** – Defines the message priority. The valid values are 0 through 9, where 0 is the lowest priority and 9 is the highest priority. The default value is 4.

Advanced Configuration

When you are creating a JMS-based concrete WSDL document, the Advanced Configuration step appears for all of the possible types.



```
# General options, some examples (see documentation for details)
#Emulates XA
#JMSJCA.NoXA=false
```

You can specify any option supported by JMSJCA in the Advanced Configuration step. For example:

```
JMSJCA.NoXA=false
```

If you set the connection URL to `jndi://` in the Request Connection Configuration step, then you specify the JNDI options in the Advanced Configuration step. For example:

```
JMSJCA.UnifiedCF=JNDI-name
JMSJCA.TopicCF=JNDI-name
JMSJCA.QueueCF=JNDI-name
java.naming.factory.initial=com.sonicsw.jndi.mfcontext.MFContextFactory
java.naming.provider.url=tcp://localhost:2506
java.naming.security.principal=Administrator
java.naming.security.credentials=Administrator
com.sonicsw.jndi.mfcontext.domain=Domain1
com.sonicsw.jndi.mfcontext.idleTimeout=60000
```

For more information about using the `jndi` approach, see "Connecting to the JMS Server" on page 24.

Connecting to the JMS Server

You use a *connection URL* to specify the information for connecting to the JMS server.

For any JMS server that is supported by JMSJCA, you can specify the protocol, server, and port in the connection URL. The JMS Binding Component uses the information in the connection URL to create a new JMS connection factory.

Note – JMSJCA is a library that abstracts the differences between JMS servers and provides a single interface to the JMS servers. JMSJCA is shipped as part of the JMS Binding Component. For a list of supported JMS servers, go to `https://jmsjca.dev.java.net`.

You can connect to any JMS server by using the Java Naming and Directory Interface™ (JNDI) to locate an existing JMS connection factory.

A *connection factory* is a Java class supplied by the JMS provider. For example:

```
com.stc.jmsjca.core.JConnectionFactoryXA
```

- "Specifying the Protocol, Server, and Port" on page 24
- "Connecting to the JMS Server by Using JNDI" on page 25

Specifying the Protocol, Server, and Port

For any JMS server that is supported by JMSJCA, you can specify the protocol, server, and port in the connection URL. The JMS Binding Component uses the information in the connection URL to create a new JMS connection factory.

The following connection URL includes a protocol, server, and port for Sun Java System Message Queue:

```
mq://localhost:7676
```

The following connection URL includes a protocol, server, and port for Sun JMS IQ Manager:

```
stcms://localhost:18007
```

When you use a wizard to create a JMS-based concrete WSDL document, you set the connection URL in the Request Connection Configuration step of the wizard. For more information, see "Creating JMS-Based Concrete WSDL Documents" on page 13.

When you finish the wizard, the appropriate WSDL code is generated. The connection URL appears in the `address` element. For example:

```
<port name="newWSDL_InPort" binding="tns:JMSInBinding">
  <jms:address connectionURL="mq://localhost:7676" username="admin" password="admin">
    <jms:jmsjcaOptions><![CDATA[]]></jms:jmsjcaOptions>
  </jms:address>
</port>
```

Connecting to the JMS Server by Using JNDI

You can connect to any JMS server by using the Java Naming and Directory Interface (JNDI) to locate an existing JMS connection factory.

Depending on where the connection factory is bound, the connection URL can begin with the string `lookup` or the string `jndi`.

The appropriate JNDI provider jar files must be in the runtime classpath. With the GlassFish application server, you copy these jar files to the `lib` directory. In addition, JMS provider client jar files are needed in the runtime classpath.

Using the `lookup` Approach

To access a connection factory that is bound in the JNDI space of the GlassFish application server itself, use the `lookup` approach.

This scenario can occur when a *managed connection factory* has been created. Managed connection factories provide additional services on top of a connection factory. The additional services include connection pooling. The managed connection factory creates a connection factory wrapper.

Set the connection URL to `lookup://JNDI-name`, where `JNDI-name` is the JNDI name to which the connection factory is bound. For example:

```
lookup://jms/tx/default
```

When you use a wizard to create a JMS-based concrete WSDL document, you set the connection URL in the Request Connection Configuration step of the wizard. For more information, see "Creating JMS-Based Concrete WSDL Documents" on page 13.

When you finish the wizard, the appropriate WSDL code is generated. The connection URL appears in the `address` element. For example:

```
<port name="newWSDL_InPort" binding="tns:JMSInBinding">
  <jms:address connectionURL="lookup://jms/tx/default">
    <jms:jmsjcaOptions><![CDATA[]]></jms:jmsjcaOptions>
  </jms:address>
</port>
```

At runtime, the JMS Binding Component uses the JNDI name to obtain the connection factory.

Using the `jndi` Approach

Use the `jndi` approach for either of the following scenarios:

- To look up a connection factory in an external JNDI provider (for example, file or LDAP).
- To look up a connection factory in a JNDI provider in the JMS server itself.

Set the connection URL to `jndi://` and specify the JNDI options.

When you use a wizard to create a JMS-based concrete WSDL document, you set the connection URL in the Request Connection Configuration step of the wizard. You then specify the JNDI options in the Advanced Configuration step of the wizard. For more information, see "Creating JMS-Based Concrete WSDL Documents" on page 13.

The following table describes the available JNDI options.

JNDI Option	Description
`JMSJCA.UnifiedCF`	The JNDI name of the connection factory. Use this option only in outbound scenarios.
`JMSJCA.TopicCF`	The JNDI name of topic connection factory. You can use this option in inbound or outbound scenarios.
`JMSJCA.QueueCF`	The JNDI name of queue connection factory. You can use this option in inbound or outbound scenarios.
`java.naming.factory.initial`	The fully qualified class name of the factory class that will create the initial context. An initial context is the starting point for naming operations. For more information, see the Java API documentation for `javax.naming.Context.INITIAL_CONTEXT_FACTORY`.
`java.naming.provider.url`	The configuration information for the service provider to use. The value should contain a URL string. For more information, see the Java API documentation for `javax.naming.Context.PROVIDER_URL`.
`java.naming.security.principal`	The identity of the principal for authenticating the caller to the service. For more information, see the Java API documentation for `javax.naming.Context.SECURITY_PRINCIPAL`.
`java.naming.security.credentials`	The credentials of the principal for authenticating the caller to the service. For more information, see the Java API documentation for `javax.naming.Context.SECURITY_CREDENTIALS`.

The `destinationType` and `transaction` attributes in the WSDL document determine what type of connection factory is being looked up. For example, if the `destinationType` attribute is `Queue` and the `transaction` attribute is `XATransaction`, then the connection factory must be of type `javax.jms.XAQueueConnectionFactory`.

The following code shows an example of specifying the JNDI options. In this scenario, the connection factory is bound in the JMS server itself.

```
JMSJCA.UnifiedCF=connectionfactories/xaconnectionfactory
JMSJCA.TopicCF=connectionfactories/xatopicconnectionfactory
JMSJCA.QueueCF=connectionfactories/xaqueueconnectionfactory
java.naming.factory.initial=com.stc.jms.jndispi.InitialContextFactory
java.naming.provider.url=stcms://localhost:18007
java.naming.security.principal=Administrator
java.naming.security.credentials=STC
```

When you finish the wizard, the appropriate WSDL code is generated. The connection URL appears in the `address` element. The JNDI options appear in the `jmsjcaOptions` element. For example:

```
<port name="newWSDL_InPort" binding="tns:JMSInBinding">
  <jms:address connectionURL="jndi://">
    <jms:jmsjcaOptions>
      <![CDATA[JMSJCA.UnifiedCF=connectionfactories/xaconnectionfactory
      JMSJCA.TopicCF=connectionfactories/xatopicconnectionfactory
      JMSJCA.QueueCF=connectionfactories/xaqueueconnectionfactory
      java.naming.factory.initial=com.stc.jms.jndispi.InitialContextFactory
      java.naming.provider.url=stcms://localhost:18007
      java.naming.security.principal=Administrator
      java.naming.security.credentials=STC]]>
    </jms:jmsjcaOptions>
  </jms:address>
</port>
```

At runtime, the JMS Binding Component uses the JNDI options to obtain the connection factory.

Configuring Redelivery Handling

A JMS message is typically redelivered because of an error in the processing of the message. The error may be transient or permanent.

If the error is transient, the message will eventually go through. If the error is permanent, moving messages to a different destination may be a better approach. If the message is not valuable, deleting the message is another option.

Delaying delivery of a redelivered message is useful to save CPU cycles instead of letting the message "spin" rapidly.

You can configure redelivery handling by using either of the following approaches:

- "Configuring Redelivery Handling from the Wizard" on page 28

- "Configuring Redelivery Handling by Editing the WSDL Document" on page 29

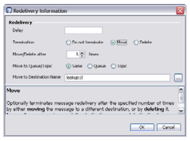
Configuring Redelivery Handling from the Wizard

You can use a wizard to create a concrete WSDL document that contains JMS binding and service information. The wizard is described in "Creating JMS-Based Concrete WSDL Documents" on page 13.

You can configure redelivery handling from the following wizard steps:

- Request Consumer Configuration
- Response Consumer Configuration

Click the Details button, which appears to the right of the Redelivery field. The Redelivery Information dialog box appears.



Delay

Termination ( ) Do not terminate [✓ Move] ( ) Delete

Move/Delete after 1 times

Move to Queue/Topic [✓ Same] ( ) Queue ( ) Topic

Move to Destination Name lookup://

Move

Optionally terminates message redelivery after the specified number of times by either **moving** the message to a different destination, or by **deleting** it.

The Redelivery Information dialog box contains the following fields:

- **Delay** – An entry consists of two numbers, separated by a colon (:). The first number is the number of times that the message has been delivered. The second number is how many milliseconds to wait before resending the message.

  Assume that the entry is `5:1000`. When a message is received for the fifth time, the message is delayed for one second. If you specify more than one entry, then you must separate the entries with a semicolon (;). For example:

  ```
  5:1000; 10:5000
  ```

- **Termination** – If you want to move the message to another destination, then select the Move option and set the remaining fields in the dialog box.

  If you want to delete the message, then select the Delete option and set the Move/Delete After field.

- **Move/Delete After** – The number of times after which the message is moved or deleted.
- **Move to Queue/Topic** – Indicates whether the message should be moved to a queue or topic.
- **Move to Destination Name** – Indicates the name of the destination where the message should be moved. The string can include the dollar sign ($) character, which is replaced with the original destination name.

Configuring Redelivery Handling by Editing the WSDL Document

You can configure redelivery handling by entering a specially formatted string in a WSDL document.

The string specifies what actions to take when the message is repeatedly redelivered. The string has the following format:

```
format := entry[; entry]*
entry := index ":" action
index := number (denotes the n-th time a message is seen)
action := number (denotes delay in milliseconds) | "delete" | "move"(args)
move := "queue"|"topic" | "same" ":" destname
destname := any string, may include "$", which is replaced with the original
destination name
```

Redelivery Handling Example 1

Assume that the string is set as follows:

```
redeliveryHandling="5:1000; 10:5000"
```

This example causes no delay up to the 5th delivery. When the message is seen for the 5th, 6th, 7th, 8th, and 9th time, a 1000–millisecond delay is invoked. For each time the message is seen thereafter, a 5000–millisecond delay is invoked.

Redelivery Handling Example 2

Assume that the string is set as follows:

```
redeliveryHandling="5:1000; 10:5000; 50:move(queue:mydlq)"
```

This example causes no delay up to the 5th delivery. When the message is seen for the 5th, 6th, 7th, 8th, and 9th time, a 1000–millisecond delay is invoked. When the message is invoked for the 10th, 11th, ..., and 49th time, a 5000–millisecond delay is invoked. When the message is seen for the 50th time, the message is moved to a queue called `mydlq`.

Assume that the messages are received from `Queue1` and the string is set as follows:

```
redeliveryHandling="5:1000; 10:5000; 50:move(queue:dlq$oops)"
```

In this case, the messages are moved to the destination `dlqQueue1oops`. The dollar sign ($) character denotes the original destination name. Instead of `queue`, you can specify `topic` or `same`. The value `same` denotes a queue if the message was received from a queue, or can denote a topic if the message was received from a topic.

Using the Normalized Message Properties

This topic describes the normalized message properties for the JMS Binding Component.

The normalized message properties enable you to do the following:

- Override the JMS settings in the WSDL document
- Specify JMS user properties

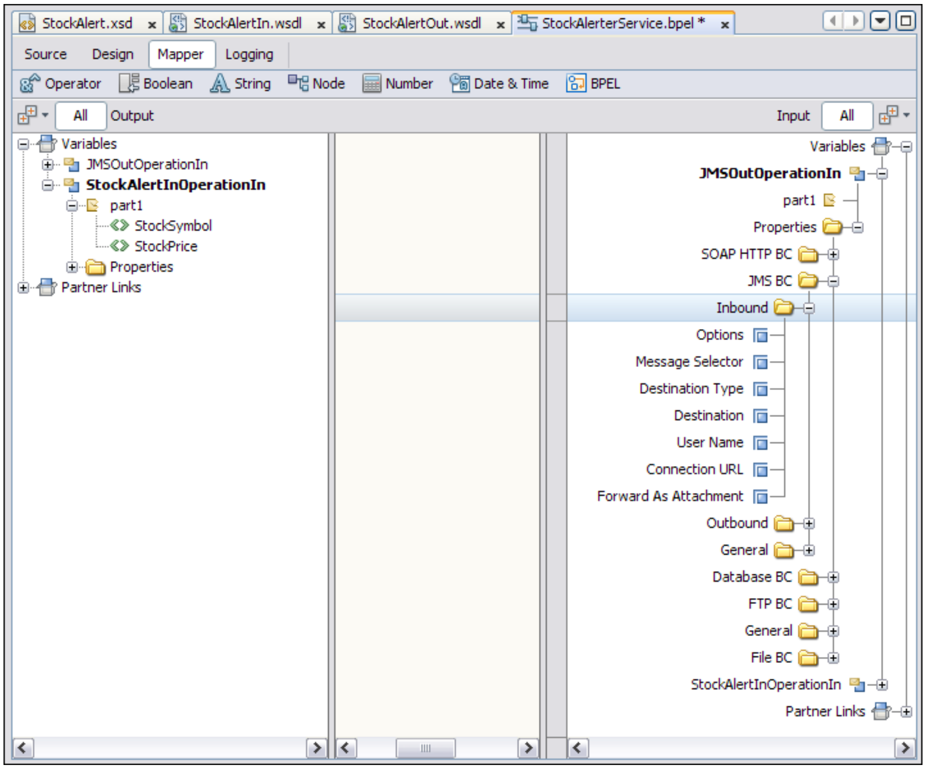
The normalized message properties are divided into the following categories:

- "Inbound Normalized Message Properties" on page 30
- "Outbound Normalized Message Properties" on page 33
- "General Normalized Message Properties" on page 37

Inbound Normalized Message Properties

This category of normalized message properties applies to inbound-only scenarios.

You can access these properties from the left and right panes of the BPEL Mapper. In the following screen capture, the nodes in the right pane are expanded to show the properties.



These properties are read only. You cannot modify the values.

Connection URL Property (org.glassfish.openesb.jms.inbound.connectionurl)

This property enables you to specify the URL for connecting to the JMS server.

This property is applicable to the following binding types:

- Receive
- Receive with Reply
- Send and Wait for Reply
- On Demand Receive

For more information, see “Connecting to the JMS Server” on page 24.

User Name Property (org.glassfish.openesb.jms.inbound.username)

This property enables you to specify the user name for connecting to the JMS server.

This property is applicable to the following binding types:

- Receive
- Receive with Reply
- Send and Wait for Reply
- On Demand Receive

Message Selector Property (org.glassfish.openesb.jms.inbound.messageselector)

This property enables you to filter messages. A message selector consists of a boolean expression, such as `Age > 30`.

This property is applicable to the following binding types:

- Receive
- Receive with Reply
- On Demand Receive

Destination Property (org.glassfish.openesb.jms.inbound.destination)

This property enables you to specify the name of the JMS destination from which messages are received.

This property is applicable to the following binding types:

- Receive
- Receive with Reply
- Send and Wait for Reply
- On Demand Receive

Destination Type Property (org.glassfish.openesb.jms.inbound.destinationtype)

This property enables you to indicate whether the JMS destination is a queue or a topic. The valid values are `Queue` and `Topic`.

This property is applicable to the following binding types:

- Receive
- Receive with Reply
- Send and Wait for Reply
- On Demand Receive

Forward As Attachment Property (org.glassfish.openesb.jms.inbound.forwardasattachment)

This property enables you to send the message data as an attachment. The valid values are `true` and `false`.

For binary data, the data is sent as an attachment by default.

For XML data, sending the data as an attachment prevents the JMS Binding Component from parsing the XML. If the XML is large, then this approach can improve performance.

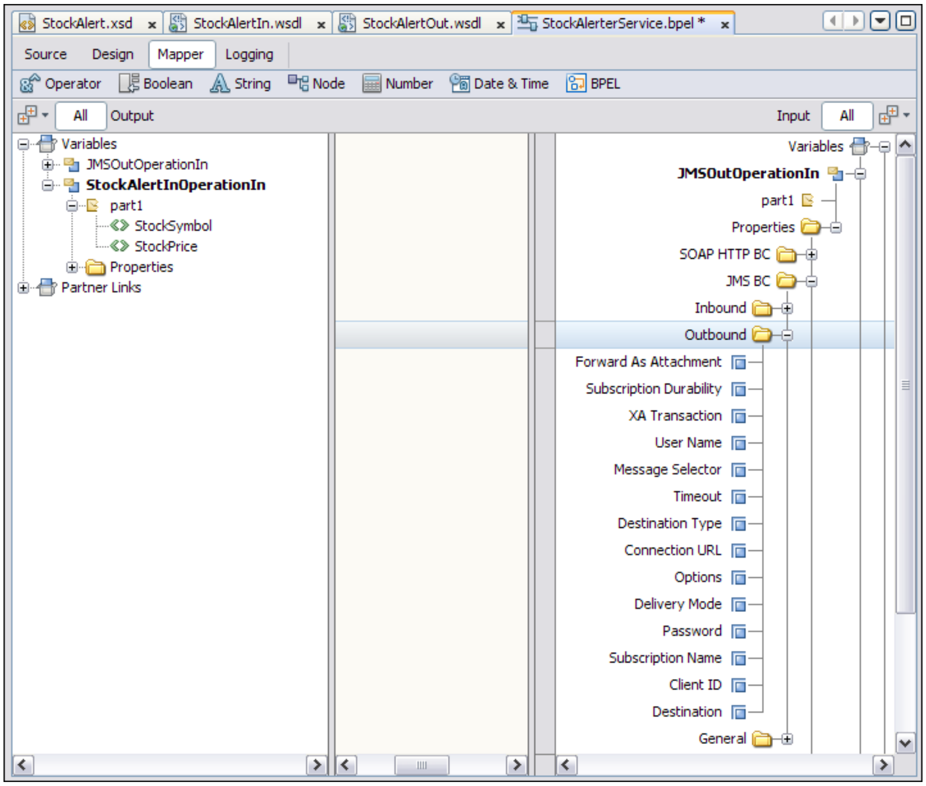
This property is applicable to the following binding types:

- Receive
- Receive with Reply
- Send and Wait for Reply
- On Demand Receive

Outbound Normalized Message Properties

This category of normalized message properties applies to outbound-only scenarios.

You can access these properties from the left and right panes of the BPEL Mapper. In the following screen capture, the nodes in the right pane are expanded to show the properties.



Connection URL Property (org.glassfish.openesb.jms.outbound.connectionurl)

This property enables you to specify the URL for connecting to the JMS server.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply
- On Demand Receive

For more information, see "Connecting to the JMS Server" on page 24.

User Name Property (org.glassfish.openesb.jms.outbound.username)

This property enables you to specify the user name for connecting to the JMS server.

This property is applicable to the following binding types:

- Receive with Reply

- Send
- Send and Wait for Reply
- On Demand Receive

Password Property (org.glassfish.openesb.jms.outbound.password)

This property enables you to specify the password for connecting to the JMS server.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply
- On Demand Receive

Destination Property (org.glassfish.openesb.jms.outbound.destination)

This property enables you to specify the name of the JMS destination to which messages are sent.

This property is applicable to the following binding types:

- Send
- Send and Wait for Reply
- On Demand Receive

Destination Type Property (org.glassfish.openesb.jms.outbound.destinationtype)

This property enables you to indicate whether the JMS destination is a queue or a topic. The valid values are `Queue` and `Topic`.

This property is applicable to the following binding types:

- Send
- Send and Wait for Reply
- On Demand Receive

XA Transaction Property (org.glassfish.openesb.jms.outbound.xatransaction)

This property enables you to define the transaction type for the JMS protocol based operation. The valid values are `NoTransaction` and `XATransaction`.

This property is applicable to the following binding types:

- Receive with Reply

- Send
- On Demand Receive

Delivery Mode Property (org.glassfish.openesb.jms.outbound.deliverymode)

This property enables you to specify the message delivery mode to use when sending a message. The valid values are `PERSISTENT` and `NON_PERSISTENT`.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Timeout Property (org.glassfish.openesb.jms.outbound.timeOut)

This property enables you to specify the timeout in milliseconds on a message consumer receive for a reply message.

This property is applicable to the following binding types:

- Send and Wait for Reply
- On Demand Receive

Client ID (org.glassfish.openesb.jms.outbound.clientid)

This property defines a unique client ID.

This property is applicable to the On Demand Receive binding type.

Message Selector Property (org.glassfish.openesb.jms.outbound.messageselector)

This property enables you to filter messages. A message selector consists of a boolean expression, such as `Age > 30`.

This property is applicable to the On Demand Receive binding type.

Subscription Durability Property (org.glassfish.openesb.jms.outbound.subscriptiondurability)

This property enables you to configure the durability of the topic subscriber. Durable subscribers can survive any disconnection from the JMS server.

The valid values are `Durable` and `NonDurable`.

This property is applicable to the On Demand Receive binding type.

Subscription Name Property (org.glassfish.openesb.jms.outbound.subscriptionname)

For durable subscriptions, this property enables you to specify the name of the durable subscription.

This property is applicable to the On Demand Receive binding type.

Forward As Attachment Property (org.glassfish.openesb.jms.outbound.forwardasattachment)

This property enables you to send the message data as an attachment. The valid values are `true` and `false`.

For binary data, the data is sent as an attachment by default.

For XML data, sending the data as an attachment prevents the JMS Binding Component from parsing the XML. If the XML is large, then this approach can improve performance.

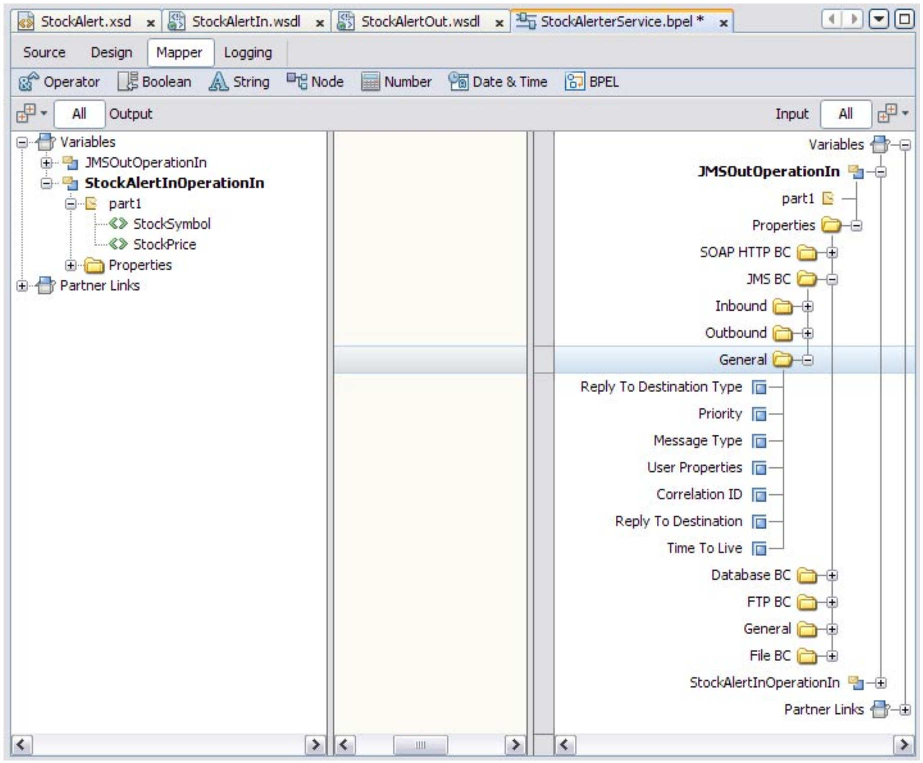
This property is applicable to the following binding types:

- Send and Wait for Reply
- On Demand Receive

General Normalized Message Properties

This category of normalized message properties applies to both inbound and outbound scenarios.

You can access these properties from the left and right panes of the BPEL Mapper. In the following screen capture, the nodes in the right pane are expanded to show the properties.



Time To Live Property (org.glassfish.openesb.jms.timetolive)

This property indicates how long the message is retained (in milliseconds).

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Priority Property (org.glassfish.openesb.jms.priority)

This property enables you to specify the message priority for a message producer. The valid values are 0 through 9, where 0 is the lowest priority and 9 is the highest priority. The default value is 4.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Message Type Property (org.glassfish.openesb.jms.messagetype)

This property enables you to specify whether the messages are text, bytes, XML, or encoded data.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Correlation ID Property (org.glassfish.openesb.jms.correlationid)

This property enables you to reference the message part that contains the value for the `JMSCorrelationID` header.

You can use a JMS correlation ID to associate a reply message with the corresponding request message.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Reply To Destination Property (org.glassfish.openesb.jms.replytodestination)

This property enables you to specify the name of the JMS destination to which messages should be replied.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

Reply To Destination Type Property (org.glassfish.openesb.jms.replytodestinationtype)

This property enables you to indicate whether the JMS destination is a queue or a topic. The valid values are `Queue` and `Topic`.

This property is applicable to the following binding types:

- Receive with Reply
- Send
- Send and Wait for Reply

User Properties Property (org.glassfish.openesb.jms.userproperties)

This property enables you to configure JMS properties, which are custom headers.

This property is the only normalized message property that you cannot configure in the BPEL Mapper. Instead, you must go to the Source view and manually enter a copy statement. Add a period and the property name to `org.glassfish.openesb.jms.userproperties`. For example:

```
<copy>
  <from>$SoapInboundOperationIn.part1/ns0:StockSymbol</from>
  <to variable="JMSOutOperationIn"
    sxnmp:nmProperty="org.glassfish.openesb.jms.userproperties.StockSymbol"/>
</copy>
```

The default property type is `string`. If you want to specify a type other than `string`, then you must enter an additional copy statement for the type. For example:

```
<copy>
  <from>$SoapInboundOperationIn.part1/ns0:StockPrice</from>
  <to variable="JMSOutOperationIn"
    sxnmp:nmProperty="org.glassfish.openesb.jms.userproperties.StockPrice"/>
</copy>
<copy>
  <from>'float'</from>
  <to variable="JMSOutOperationIn"
    sxnmp:nmProperty="org.glassfish.openesb.jms.userproperties.StockPrice.type"/>
</copy>
```

This property is applicable to all of the binding types.

JMS WSDL Extensibility Elements

The JMS WSDL extensibility elements enable you to configure JMS connectivity and binding information for the JMS Binding Component.

- “JMS Connectivity Elements” on page 41
  - “JMS `address` Element” on page 41
  - “JMS `jmsjcaOptions` Element” on page 41
- “JMS Binding Elements” on page 42
  - “JMS `binding` Element” on page 42
  - “JMS `operation` Element” on page 42
  - “JMS `message` Element” on page 45
  - “JMS `properties` and `property` Elements” on page 47
  - “JMS `mapmessage` and `mappart` Elements” on page 48

JMS Connectivity Elements

JMS connectivity elements consist of the `address` element and the `jmsjcaOptions` element.

JMS `address` Element

The JMS `address` extensibility element specifies the JMS server connectivity information.

The `connectionURL` attribute is required. The other attributes are optional.

All of the attributes apply to both providers and consumers.

Attribute	Description	Example
connectionURL	A URL that specifies the connectivity information to connect to the JMS server.	mq://localhost:7676
username	Specifies the username used for connecting to the JMS server.	
password	Specifies the password used for connecting to the JMS server.	

The following example illustrates the JMS `address` extensibility element.

```
<port binding="y:binding" name="jmsOutOnlyTestEndpoint">
  <jms:address connectionURL="mq://localhost:7676"
               username="admin" password="admin"/>
</port>
```

JMS `jmsjcaOptions` Element

The JMS `jmsjcaOptions` extensibility element can specify any option supported by JMSJCA.

Note – JMSJCA is a library that abstracts the differences between JMS servers and provides a single interface to the JMS servers. JMSJCA is shipped as part of the JMS Binding Component. For a list of supported JMS servers, go to `https://jmsjca.dev.java.net`.

The following example illustrates the JMS `jmsjcaOptions` extensibility element.

```
<port name="newWSDL_InPort" binding="tns:JMSInBinding">
  <jms:address connectionURL="jndi://">
    <jms:jmsjcaOptions>
      <![CDATA[JMSJCA.UnifiedCF=connectionfactories/xaconnectionfactory
      JMSJCA.TopicCF=connectionfactories/xatopicconnectionfactory
      JMSJCA.QueueCF=connectionfactories/xaqueueconnectionfactory
      java.naming.factory.initial=com.stc.jms.jndispi.InitialContextFactory
      java.naming.provider.url=stcms://localhost:18007
      java.naming.security.principal=Administrator
      java.naming.security.credentials=STC]]>
    </jms:jmsjcaOptions>
  </jms:address>
</port>
```

JMS Binding Elements

The JMS extensibility elements for binding abstract WSDL messages to JMS messages fall into several sections.

JMS `binding` Element

The JMS `binding` extensibility element indicates a binding that is of interest to the JMS Binding Component. This element is an empty element that serves as a marker enabling the JMS Binding Component to gather JMS "binding" information described by the other JMS extensibility elements.

The JMS `binding` extensibility element must be specified in the WSDL to define a JMS protocol-based binding.

The following example illustrates the JMS `binding` extensibility element.

```
<binding name="bindingJMSOneWayOut"
         type="tns:portTypeOneWayOut">
  <jms:binding></jms:binding>
  ...
</binding>
```

JMS `operation` Element

The JMS `operation` extensibility element indicates an operation binding that is of interest to the JMS Binding Component. This element has attributes and child elements that are used to define JMS message delivery options for the JMS Binding Component.

The `destination` and `destinationType` attributes are required. The other attributes are optional.

The following attributes apply to both providers and consumers: `destination`, `destinationType`, and `transaction`.

The following attributes apply to providers only: `deliveryMode`, `timeToLive`, `priority`, `disableMessageID`, `disableMessageTimeStamp`, `timeout`, and `clientID`.

The following attributes apply to consumers only: `messageSelector`, `subscriptionDurability`, `subscriptionName`, `batchSize`, `maxConcurrentConsumers`, and `redeliveryHandling`.

Attribute	Description	Example
destination	Defines the destination where messages are sent or received.	InvoiceTopic
destinationType	Specifies the destination type. The valid values are `Queue` and `Topic`.	Topic
transaction	Defines the transaction type for the JMS protocol based operation. The valid values are `NoTransaction` and `XATransaction`. XA in-out operations are supported with inbound message flows (when the JMS Binding Component acts as a consumer of messages). They are not supported with outbound message flows (when the JMS Binding Component acts as a provider of messages). The JMS Binding Component generates a deployment error, indicating that this type of operation is not supported, when the transaction attribute is set to `XATransaction` for in-out operations with a provisioning endpoint.	
deliveryMode	The message delivery mode to use when sending a message. The valid values are `PERSISTENT` and `NON_PERSISTENT`. The default value is `NON_PERSISTENT`.	
timeToLive	The time in milliseconds (from the dispatched time) that a produced message should be retained by the message system.	120000
priority	Defines the message priority for a message producer. The valid values are 0 through 9, where 0 is lowest priority and 9 is highest priority. The default value is 4.	4

Attribute	Description	Example
disableMessageID	Indicates whether message IDs are disabled for a message producer. The valid values are `true` and `false`. The default value is `false`. Sun Java System Message Queue ignores the `disableMessageID` attribute when it is a provider of messages.	
disableMessageTimeStamp	Indicates whether message timestamps are disabled for a message producer. The valid values are `true` and `false`. The default value is `false`.	
timeout	The timeout in milliseconds on a message consumer receive for a reply message. This attribute applies only to the provider request reply.	120000
clientID	Defines a unique client ID. The `durableName` will be used as the `clientID` if a durable subscriber is used but not set.	ClientID123
messageSelector	Enables you to filter messages. A message selector consists of a boolean expression, such as `Age > 30`.	JMSCorrelationID='88888888' AND JMSType='SUN'
subscriptionDurability	Determines the durability of the topic subscriber. The valid values are `Durable` and `NonDurable`. The default value is `NonDurable`.	
subscriptionName	The name that is used to denote the durable subscription. This attribute is used only with a durable subscriber (the destination is a topic and the `subscriptionDurability` attribute is set to `Durable`).	SunStockSubscriptionName
batchSize	If defined with a positive integer, this attribute specifies that the messages received will be in a batch. The number of messages in the batch could be less than or equal to the specified size.	20

Attribute	Description	Example
maxConcurrentConsumers	If defined with a positive integer and the destination type is Queue, this attribute specifies the maximum number of concurrent receivers that can process messages. The default value is 15 if the destination type is Queue.	15
concurrencyMode	Specifies the concurrency mode. The valid values are `sync` and `cc`. There is no concurrent processing for topics in `sync` mode.	
redeliveryHandling	Specifies what actions to take if an error occurs in processing the JMS message received from the JMS destination. For more information, see "Configuring Redelivery Handling" on page 27.	5:1000; 10:5000

The following example illustrates the JMS `operation` extensibility element.

```
...
<operation name="Operation1">
  <jms:operation destination="MyTopic"
                 destinationType="Topic"
                 messageSelector="JMSType='FOO.Type'"/>
  ...
</operation>
...
```

JMS `message` Element

The `messageType` and `use` attributes are required. The other attributes are optional.

Attribute	Description	Example
messageType	Defines the type of JMS messages being created and sent to the JMS destination. The valid values are `TextMessage` and `MapMessage`.	

Attribute	Description	Example
parts	If the `messageType`attribute is set to `TextMessage`, then this attribute defines the parts from the WSDL message that comprise the text payload. Each part in the list is delimited by a space.	msgPart1,msgPart2
use	Defines the use type that affects how the message is interpreted. This attribute is currently supported as a string literal.	literal
encodingStyle	This attribute is reserved for future use.	
correlationIdPart	References the message part that contains the value for the `JMSCorrelationID` header.	partCorrelationID
deliveryMode	Defines the static value for the `JMSDeliveryMode` header.	NON_PERSISTENT
deliveryModePart	References the message part that contains the value for the `JMSDeliveryMode` header.	partDeliveryMode
priority	Defines the static value for the `JMSPriority` header.	5
priorityPart	References the message part that contains the value for the `JMSPriority` header.	partPriority
type	Defines the static value for the `JMSType` header.	MyMessageType
typePart	References the message part that contains the value for the `JMSType` header.	partType

The following example illustrates the JMS `message` extensibility element.

```
...
<jms:binding></jms:binding>
  <operation name="operationOneWayOut">
    <jms:operation destination="MyTopic"
                   destinationType="Topic"
                   messageSelector="JMSType='FOO.Type'"/>
      <input name="input">
        <!—jms:message defines the WSDL message to/from jms message mappings -->
        <jms:message messageType="MapMessage"
                     use="literal"
                     correlationIdPart=”msgPart1”
                     deliveryModePart=”msgPart2”
                     priorityPart=”msgPart3”
                     typePart=”msgPart4”
                     messageIDPart=”msgPart5”
                     timestampPart=”msgPart6”>
          <jms:mapmessage>
            <jms:mappart part="partBoolean"
                         type="boolean"
                         name="BooleanMapEntry">
            </jms:mappart>
            <jms:mappart part="partChar"
                         type="char"
                         name="CharMapEntry">
            </jms:mappart>
          </jms:mapmessage >
        </jms:message>
      </input>
    </jms:operation>
  </operation>
</jms:binding>
...
```

JMS `properties` and `property` Elements

The JMS `properties` extensibility element is a collection of `property` elements. It is an optional child element of the JMS `message` extensibility element.

Each `property` element defines a mapping of a JMS message user property, either to or from a WSDL message part.

The JMS `property` element includes the following attributes, all of which are required.

Attribute	Description	Example
name	The name of the JMS property that is mapped to the message part.	JMSProp1
part	The name of the message part to which the JMS property is mapped.	msgPart1
propertyType	The type of the JMS property. The valid values are `boolean`, `short`, `int`, `long`, `float`, `double`, or `string`.	string

The following example illustrates the JMS `properties` and `property` extensibility elements.

```
...
<jms:message messageType="TextMessage" textPart="partBody">
  <jms:properties>
    <jms:property part="partPropString"
                  propertyType="string"
                  name="AppStringProperty">
    </jms:property>
  </jms:properties>
</jms:message>
...
```

JMS `mapmessage` and `mappart` Elements

When the exchange involves a JMS `MapMessage` type, the JMS `mapmessage` extensibility element is used to define the mapping of the JMS `MapMessage` to the WSDL message parts and vice versa.

This element is a child element the JMS `message` extensibility element. The `mapmessage` element is a collection of one or more `mappart` elements.

The JMS `mappart` element contains the following attributes, all of which are required.

Element Name	Description	Example
name	The name of the `mapmessage` property that is mapped to the message part.	JMSProp1
part	The name of the message part that is mapped to the `mapmessage` property.	msgPart1
type	The Java type of the JMS `mapmessage` property.	string

The following example illustrates the JMS `mapmessage` and `mappart` extensibility elements.

```
...
<jms:message messageType="MapMessage">
  <jms:mapmessage>
    <jms:mappart part="partBoolean"
                 type="boolean"
                 name="BooleanMapEntry">
    </jms:mappart>
    <jms:mappart part="partChar"
                 type="char"
                 name="CharMapEntry">
    </jms:mappart>
  </jms:mapmessage>
</jms:message>
...
```

JMS Binding Component Clustering

Mission-critical enterprise systems need to be fast, reliable, and scalable. In a world of JMS systems that require near-zero downtime, the messaging architecture must support the clustering of JMS resources such as queues and topics, while accounting for the load-balancing of distributed JMS message traffic.

The JBI Runtime Environment supports clusters of JMS Binding Components and JMS brokers. The clustering of JMS binding components requires deploying one instance of a JMS binding component to each JBI runtime in a clustered JBI Runtime Environment. The availability and method of configuring a cluster of JMS brokers depends on the JMS provider. The Enterprise Edition of Sun Java System Message Queue supports clustering of JMS brokers.

Note – Clustering is supported for queues only. Clustering is not supported for topics.

Clustering and the JMS Binding Component is described in the following categories:

- "JMS High Availability" on page 49
- "JMS Load Balancing" on page 50
- "JMS Performance" on page 50

JMS High Availability

High availability (or failover) clusters are designed to improve availability of services by removing any single points of failure. Nodes in the cluster also provide failover and redundancy capabilities. If one node in a cluster fails, then another node ensures that there is no loss of availability of the service. A logical front-end is responsible for configuring two or more nodes to act as redundant services.

To achieve high availability of JMS services, the JMS Binding Component is available in every instance of JBI runtime. Each instance of JBI runtime is also associated with one application server running in a clustered environment. The deployment of a service assembly is replicated to each JMS Binding Component instance. As a result, multiple consumers or producers are created on connections established by the JMS Binding Component instances to the same JMS broker. The JMS broker can be a standalone broker or a broker in a high-performance cluster of brokers.

Because JMS Binding Component instances are replicated, a crash to one instance does not result in a loss of service. The JMS message acknowledgement protocol guarantees at least once delivery of messages (no loss of messages). In conjunction with XA, exactly once semantics can be achieved. A crash to the JMS Binding Component during message delivery results in the JMS broker detecting a connection loss (lacking of message acknowledgement) and redelivering the message to the next available JMS Binding Component receiving from the same queue. With

XA, a crash that leaves behind any in-doubt transactions can be recovered when the JMS Binding Component performs XA recovery in coordination with the JBI runtime and the transaction manager. Therefore, the combination of JMS acknowledgement protocol and XA protocol can support failover and recovery.

JMS Load Balancing

Load balancing clusters are designed to distribute the workload across multiple nodes in the cluster. The clusters accomplish this by having one logical front-end to the cluster that distributes any requests for work to any node in the cluster. The main purpose of this type of cluster is to provide improved performance, although they often include some high-availability features.

JMS supports load balancing by allowing multiple receivers on a queue. Replication of deployed service assemblies results in multiple JMS Binding Component instances receiving from the same queue. Messages are delivered in a point-to-point fashion, with only one receiver getting one message at a time. The JMS broker balances the load by delivering messages from the queue to any available receivers on that queue. The algorithm used for load balancing depends on the JMS provider.

Note – Load balancing is only valid if queues are used, not topics.

JMS Performance

High-performance clusters are a special type of load balancing cluster. A logical front-end to the cluster partitions the work into tiny chunks and distribute each chunk to nodes in the cluster. When the work is done, the logical front-end reassembles the result and then returns the completed result to the client of the cluster.

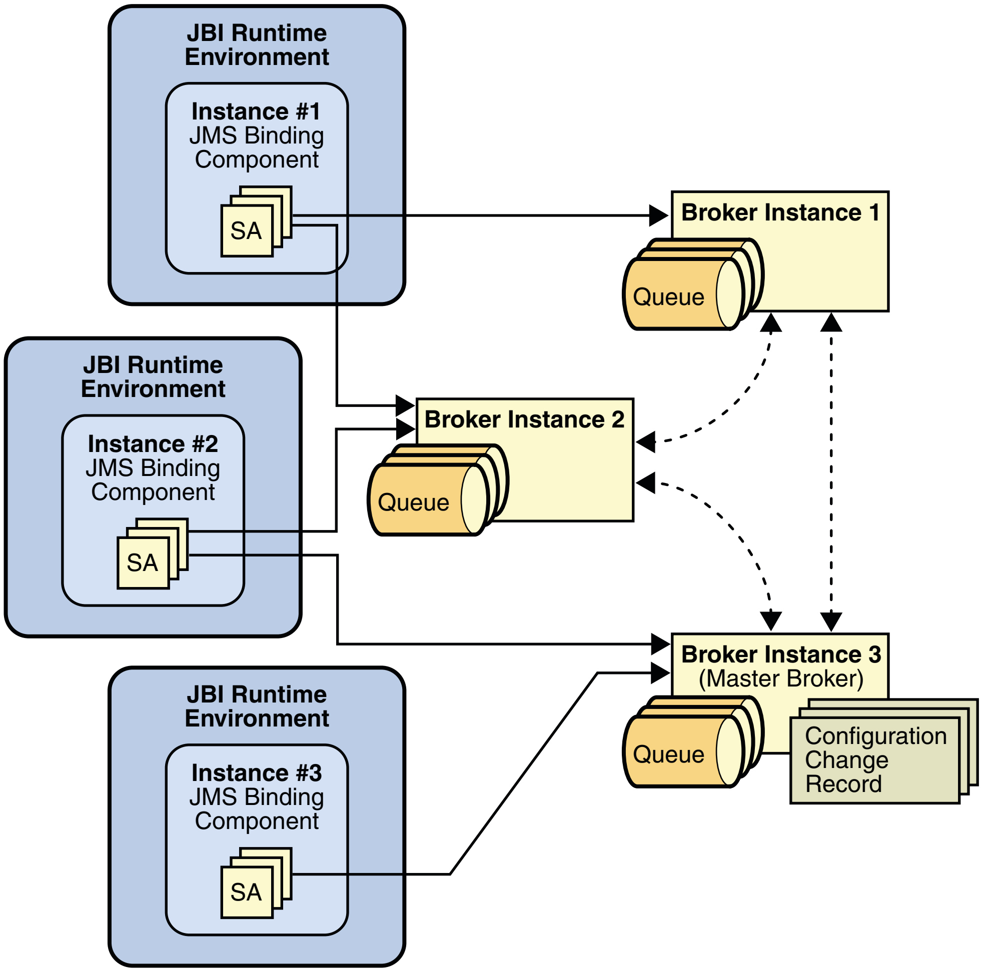
Replication of a JMS Binding Component results in multiple receivers on a queue in a clustered or nonclustered broker setup. Load is distributed across the JBI instances as a result of having multiple receivers on one or more queues.

To achieve even higher performance, the JMS brokers should be clustered. Broker clusters enable a message server to scale its operations with the volume of message traffic by distributing client connections among multiple brokers. For Sun Java System Message Queue, in a clustered broker environment, each broker within a cluster is directly connected to all the others. Each client (JMS Binding Component consumer or producer) has a single home broker with which it communicates directly, sending and receiving messages as if that broker were the only one on the server.

Behind the scenes, the home broker works with the other brokers in the cluster to share the load of providing delivery services for all connected clients. One broker within the cluster can be designated as the master broker. The master broker maintains a configuration change record in which changes to the cluster's persistent entities (destinations and durable subscriptions) are recorded. This record is used to propagate such change information to brokers that were offline at the time the changes occurred.

For detailed information about clustering for Sun Java System Message Queue, see the Sun Java System Message Queue documentation.

The following diagram illustrates how a cluster of JBI/JMS Binding Components and a cluster of Sun Java System Message Queue brokers appears.


JBI Runtime Environment
Instance #1 JMS Binding Component
SA
Broker Instance 1
Queue
JBI Runtime Environment
Instance #2 JMS Binding Component
SA
Broker Instance 2
Queue
Broker Instance 3 (Master Broker)
Queue
Configuration Change Record
JBI Runtime Environment
Instance #3 JMS Binding Component
SA

Publishing and Subscribing to Multiple WebSphere Queues

WebSphere MQ allows you to specify publish and subscribe to multiple queues, which means that each subscriber retrieves all of its messages from a specific queue that is assigned to that subscriber. JMS creates a new dynamic queue for each subscriber, and each subscriber has exclusive use of the dynamic queue assigned to it.

When you use the multiple queue approach, you do not define an explicit queue name. Instead, you define a prefix name on which all dynamic queue names are based. The prefix name needs to end in an asterisk (*); for example, "SYSTEM.JMS.D.PAYROLL.*". All dynamic queues that are associated with this subscription will have names that start with "SYSTEM.JMS.D.PAYROLL".

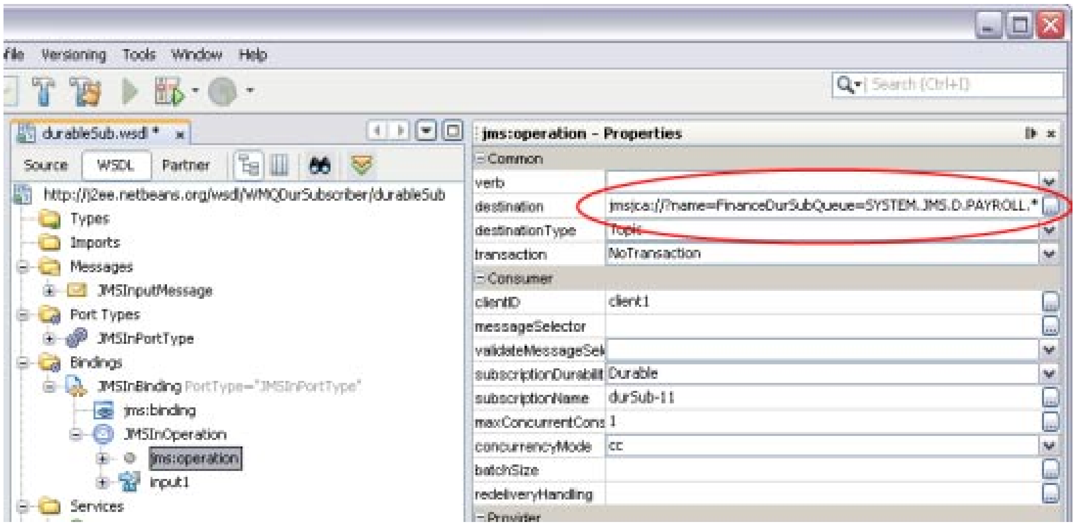
For more detailed information about WebSphere queues and configuring multiple queues, see Subscriber Options.

There are two ways to define publishing and subscribing to multiple queues for the JMS BC:

- Enter the destination name in the WSDL. The destination name provides only the prefix as described above, and does not include the full queue name. In this case, the JMSJCA parses the destination name to determine the actual destination name and properties. For example:

  ```
  <operation name="JMSInOperation">
      <jms:operation
         destination="jmsjca://?name=FinanceDurSubQueue=SYSTEM.JMS.D.PAYROLL.*"
         destinationType="Topic" ... subscriptionName="durSub-11">
           <jms:options/>
      </jms:operation>
  ```

  This is how the same destination name would appear in the WSDL Properties panel.

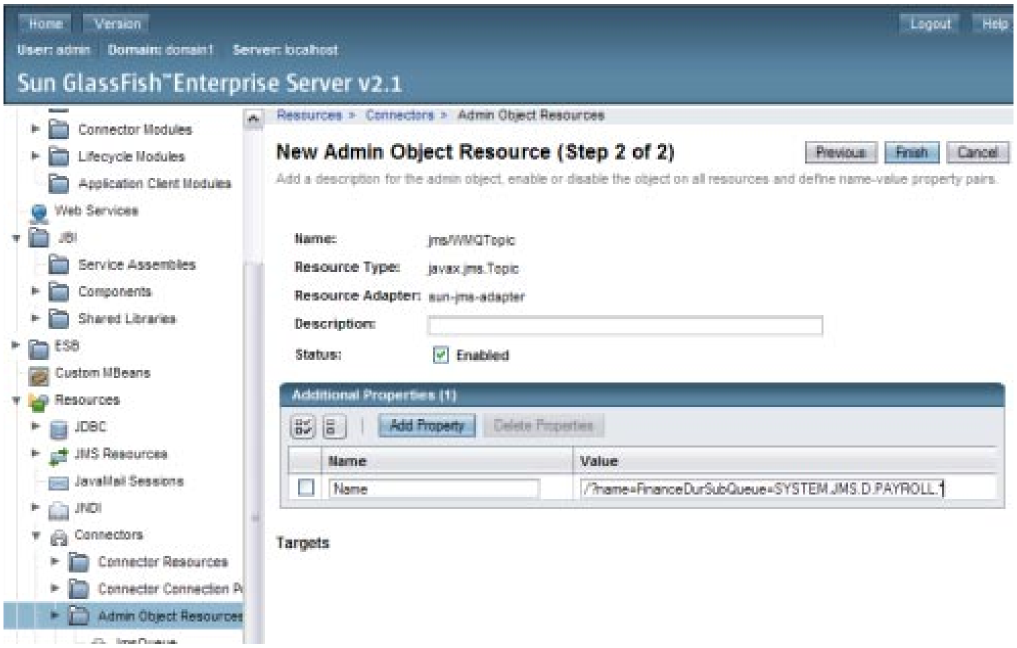


jms:operation - Properties

Common	
verb	
destination	jmsjca://?name=FinanceDurSubQueue=SYSTEM.JMS.D.PAYROLL.*
destinationType	Topic
transaction	NoTransaction
Consumer	
clientID	client1
messageSelector	
validateMessageSel	
subscriptionDurabilit	Durable
subscriptionName	durSub-11
maxConcurrentCons	1
concurrencyMode	cc
batchSize	
redeliveryHandling	
Provider	

- Create an Admin Object Resource on the GlassFish Admin Console and use a JNDI lookup in the WSDL. To create the Admin Object Resource, do the following:
  1. Launch the GlassFish Admin Console and expand Resources > Connectors > Admin Object Resources.
  2. Create a new resource, and add a `Name` property using the same naming convention as above.

     The following figure illustrates an Admin Object Resource:



New Admin Object Resource (Step 2 of 2)

Add a description for the admin object, enable or disable the object on all resources and define name-value property pairs.

Name: jms/WMQTopic

Resource Type: javax.jms.Topic

Resource Adapter: sun-jms-adapter

Description:

Status: ✓ Enabled

Additional Properties (1)

	Name	Value
☐	Name	/?name=FinanceDurSubQueue=SYSTEM.JMS.D.PAYROLL.1

3. Enter the JNDI lookup in the WSDL destination property, as shown below:

```
<operation name="JMSInOperation">
   <jms:operation destination="lookup://jms/WMQTopic" destinationType="Topic"
                  subscriptionName="durSub-11" ....>
        <jms:options/>
   </jms:operation>
```

After you configure the JMS BC to use multiple WebSphere queues, be sure to enable publish/subscribe on the WebSphere MQ broker. By default, it is not enabled in version 5.3 and 6.0.