Using the JMS JCA Wizard

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

Part No: 821–0454–10
September 2009

Copyright 2009 Sun Microsystems, Inc.   4150 Network Circle, Santa Clara, CA 95054 U.S.A.   All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun^{TM} Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

090917@22749

Contents

Using the JMS JCA Wizard

The following topics provide information for using the JMS JCA, along with instructions for configuring specific scenarios with the JMS JCA Wizard.

- "About the JMS JCA Wizard" on page 5
- "Receiving a JMS Text Message" on page 5
- "Sending a JMS Text Message" on page 15
- "Initiating a Request-Reply Transaction" on page 20

About the JMS JCA Wizard

The JMS JCA Wizard provides tools for Java EE users to easily connect to JMS message servers from their Java EE applications. The wizard is a NetBeans IDE plug-in module and provides GUI support for the JMS JCA inbound configuration and for code fragment generation through a drag-and-drop code palette. The wizard leverages the EJB 3.0 and JCA 1.5 APIs to simplify code creation. The runtime components are GlassFish Enterprise Server and the JMS JCA Adapter. The JMS JCA Adapter is a JCA 1.5 compliant resource adapter. The advantage of using the JMS JCA Adapter is that it allows you to connect transparently to the message servers from different vendors, such as WebSphere, WebLogic, JBoss, and Sun Java System Message Queue.

Receiving a JMS Text Message

This topic provides instructions for building a Message-Driven Bean (MDB) that will monitor a designated queue on a JMS destination (of the JMS Server) in order to receive JMS messages. Upon receipt of the a JMS message, the MDB will print out the content of the message if it is of the type TextMessage.

Perform the following steps to receive a JMS text message:

- "To Create an Admin Object Resource" on page 6
- "To Create the EJB Module Project" on page 7

- "To Create the JCA Message-Driven Bean" on page 9
- "To Test the Sample Code" on page 15

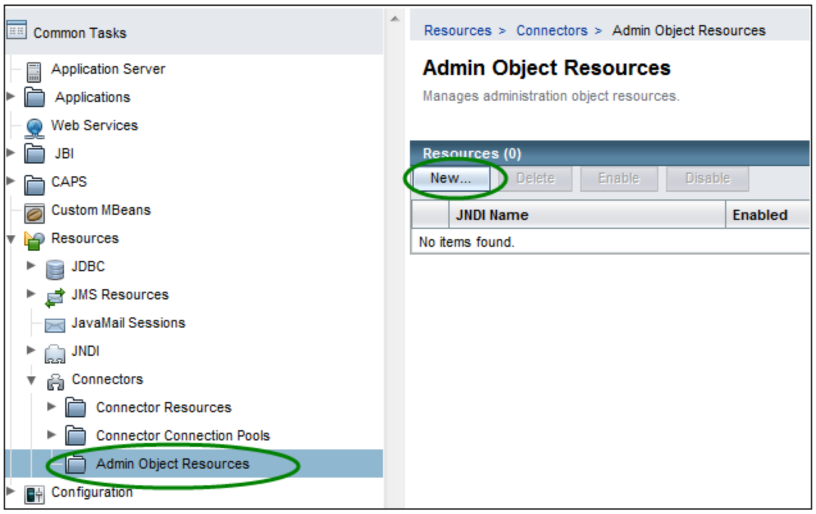
▼ To Create an Admin Object Resource

For this example, the message is being received from *Queue1*, so you need to create the corresponding JMS Queue object resource in GlassFish.

1 **Start the GlassFish server and use a browser to connect to the Admin Console.**

The URL for the Admin Console is `http://`*HostName*`:`*PortNumber*. The default port number is 4848.

2 **In the left navigation bar, expand Resources and Connectors and then select Admin Object Resources.**



Admin Object Resources

Manages administration object resources.

Resources (0)

	JNDI Name	Enabled
No items found.		

FIGURE 1 Admin Object Resources

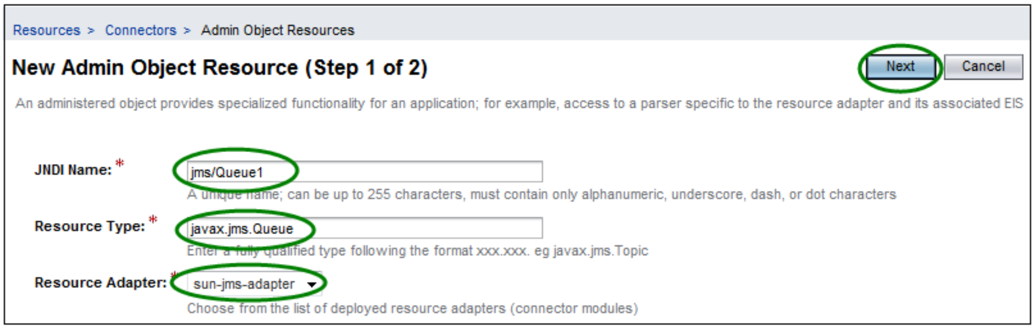
3 **Click New.**

The New Admin Object Resource window appears (Step 1 of 2).

4 **Fill in the required fields.**

For the purposes of this exercise, use the following values:

- **JNDI Name** = jms/Queue1
- **Resource Type** = javax.jms.Queue
- **Resource Adapter** = sun-jms-adapter


Resources > Connectors > Admin Object Resources
New Admin Object Resource (Step 1 of 2)
Next
Cancel

An administered object provides specialized functionality for an application; for example, access to a parser specific to the resource adapter and its associated EIS

JNDI Name: * jms/Queue1
A unique name; can be up to 255 characters, must contain only alphanumeric, underscore, dash, or dot characters

Resource Type: * javax.jms.Queue
Enter a fully qualified type following the format xxx.xxx. eg javax.jms.Topic

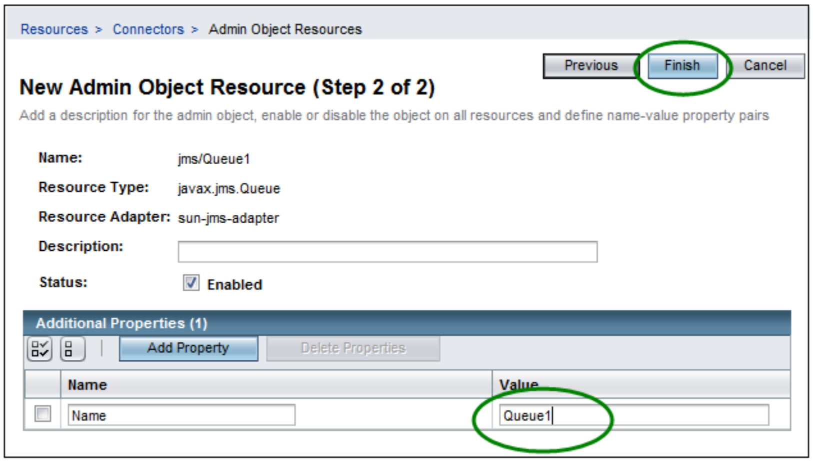
Resource Adapter: * sun-jms-adapter
Choose from the list of deployed resource adapters (connector modules)

FIGURE 2 New Admin Object Resources (Step 1 of 2)

5 **Click Next.**

6 **Enter a name for the resource.**

For this exercise, enter **`Queue1`**. This is the physical destination name of the resource.


Resources > Connectors > Admin Object Resources
Previous
Finish
Cancel
New Admin Object Resource (Step 2 of 2)
Additional Properties (1)
Add Property
Delete Properties

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

Name: jms/Queue1
Resource Type: javax.jms.Queue
Resource Adapter: sun-jms-adapter
Description:
Status: ✓ Enabled

	Name	Value
☐	Name	Queue1

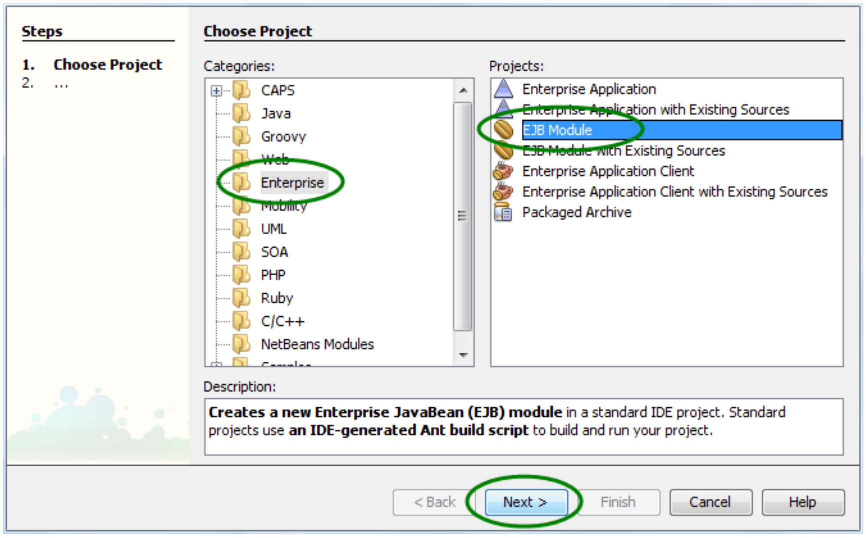
FIGURE 3 New Admin Object Resources (Step 2 of 2)

7 **Click Finish.**

▼ To Create the EJB Module Project

1 **Right–click in the Projects Panel of the NetBeans IDE and select New Project.**

2 On the New Project Wizard, select Java EE under Categories, and then select EJB Module under Projects.



Creates a new Enterprise JavaBean (EJB) module in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build and run your project.

FIGURE 4 Choose New Project

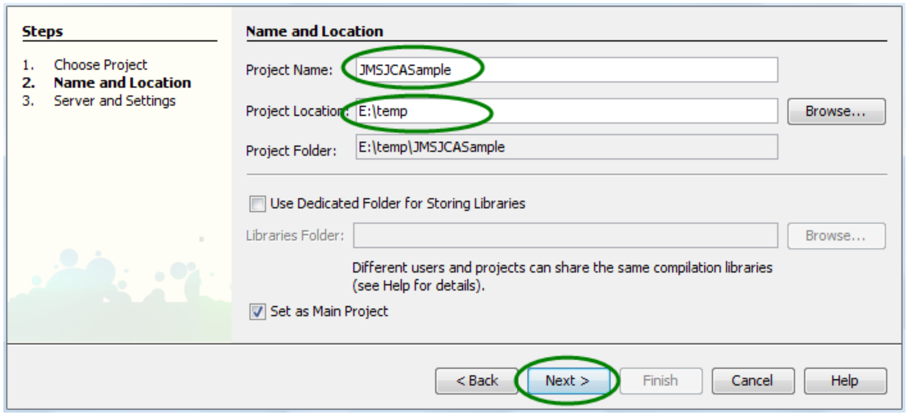
3 Click Next.

The Name and Location window appears.

4 Enter the Project Name and Location fields.

For the purposes of this exercise, enter the following values:

- **Project Name** = JMSJCASample
- **Project Location** = the location to store NetBeans project files



Project Name: JMSJCASample

Project Location: E:\temp

Project Folder: E:\temp\JMSJCASample

[ ] Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

[✓] Set as Main Project

FIGURE 5 EJB Module Project Name and Location

5 **Click Next.**

The Server and Settings window appears.

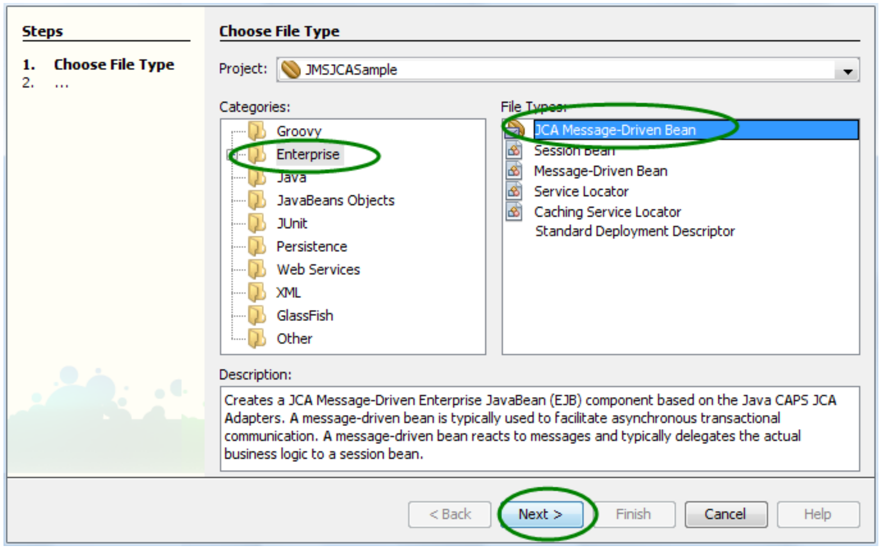
6 **In the Server and Settings window, keep the default values for all fields.**

7 **Click Finish.**

▼ To Create the JCA Message-Driven Bean

1 **Right-click on the Project node, and then select New->Other.**

2 **On the wizard, select Java EE under Categories, and select JCA Message-Driven Bean under File Types.**



Creates a JCA Message-Driven Enterprise JavaBean (EJB) component based on the Java CAPS JCA Adapters. A message-driven bean is typically used to facilitate asynchronous transactional communication. A message-driven bean reacts to messages and typically delegates the actual business logic to a session bean.

FIGURE 6 Choose JCA Message-Driven Bean

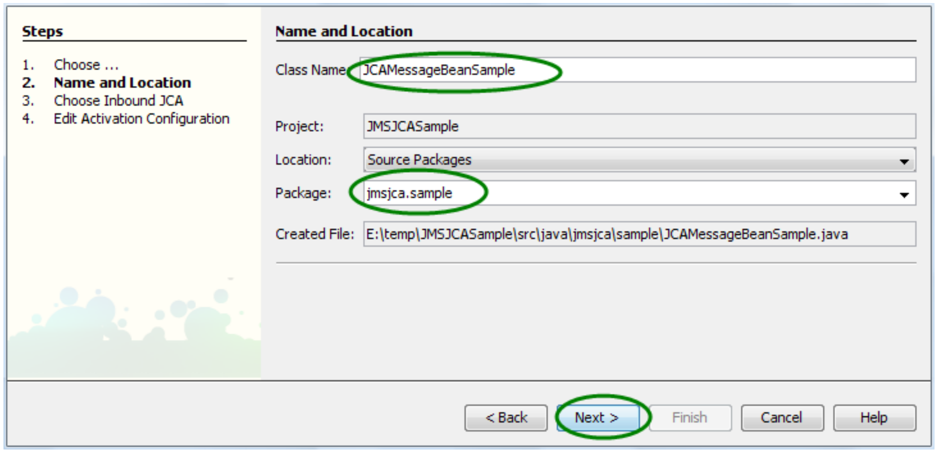
3 Click Next.

The JCA Message-Driven Bean Name and Location window appears.

4 Enter the Name and Location fields.

For this exercise, enter the following values:

- **Class Name** = JCAMessageBeanSample
- **Package** = jmsjca.sample



Class Name: JCAMessageBeanSample

Project: JMSJCASample

Location: Source Packages

Package: jmsjca.sample

Created File: E:\temp\JMSJCASample\src\java\jmsjca\sample\JCAMessageBeanSample.java

FIGURE 7 JCA Message-Driven Bean Name and Location

5 Click Next.

The Choose Inbound JCA window appears.

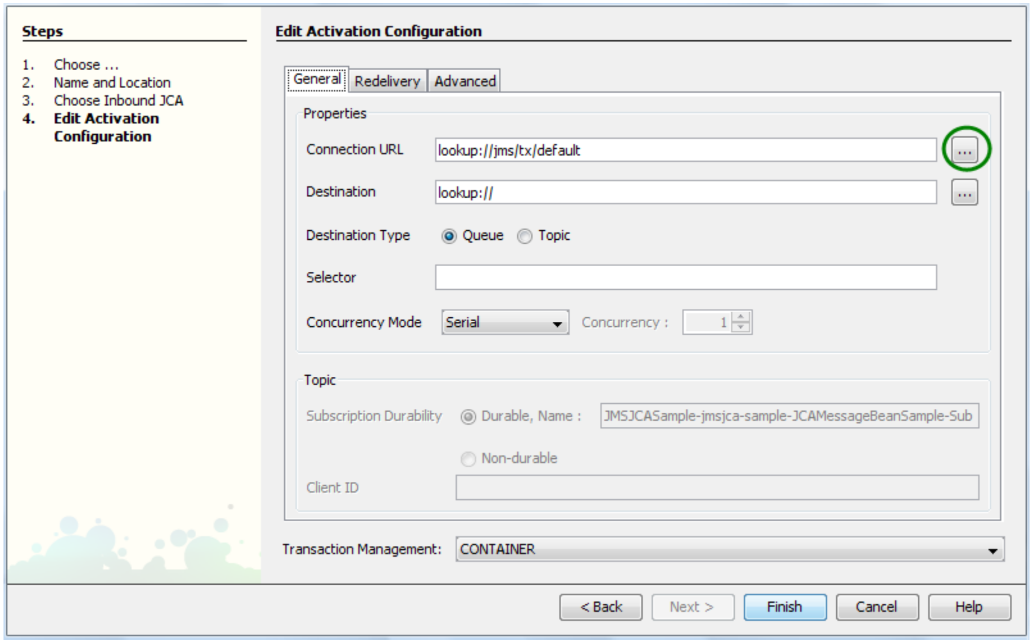
6 Select JMS Adapter and click Next.

Note – Currently only JMS Adapter can be selected in the window.

The Edit Activation Configuration window appears.

7 Configure the Inbound JMS connection by clicking on the ellipsis button next to the Connection URL box (as shown below).

You can configure many different options for the Inbound JMS connection, such as the JNDI name of the JMS connection resource or the JNDI name of the JMS destination. You can also configure the more advanced options such as message re-delivery, selector, concurrency mode, and so on. In this simple case, only the Connection URL and Destination options for our sample code to work.


Steps
1. Choose ...
2. Name and Location
3. Choose Inbound JCA
4. **Edit Activation Configuration**

Edit Activation Configuration

General | Redelivery | Advanced

Properties

Connection URL	lookup://jms/tx/default
Destination	lookup://
Destination Type	● Queue ○ Topic
Selector	
Concurrency Mode	Serial; Concurrency : 1

Topic

Subscription Durability	● Durable, Name : JMSJCASample-jmsjca-sample-JCAMessageBeanSample-Sub
	○ Non-durable
Client ID	

Transaction Management: CONTAINER

FIGURE 8 Edit Activation Configuration

8 **Expand the tree node all the way and select `jms/tx/jmq1` (as shown below).**

This resource connects the embedded Sun MQ JMS server inside the GlassFish server and is created by default with the installer. The default connection URL is `mq://localhost:7676`.

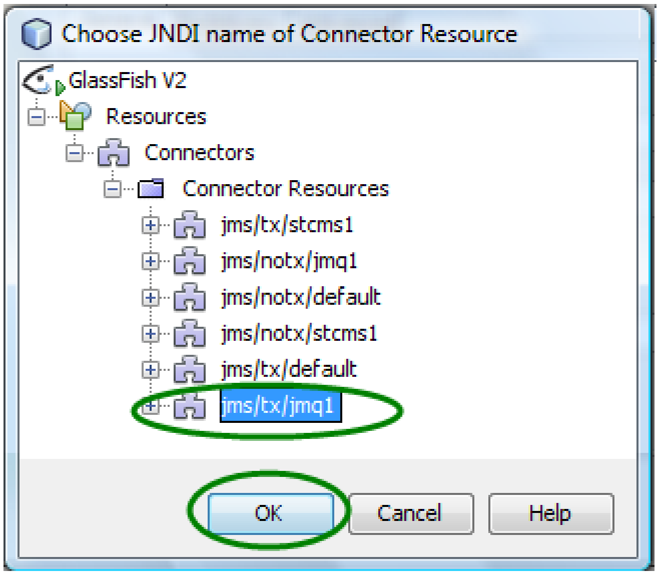


FIGURE 9 Connector Resource — Connection URL

9 **Click on the ellipsis button next to Destination box.**

The Connector Resource dialog box for the Destination appears.

10 **Expand the tree node all the way and select `jms/Queue1` (as shown below).**

This is the Admin Object Resource created earlier for the Queue1 destination using the GlassFish Admin Console.

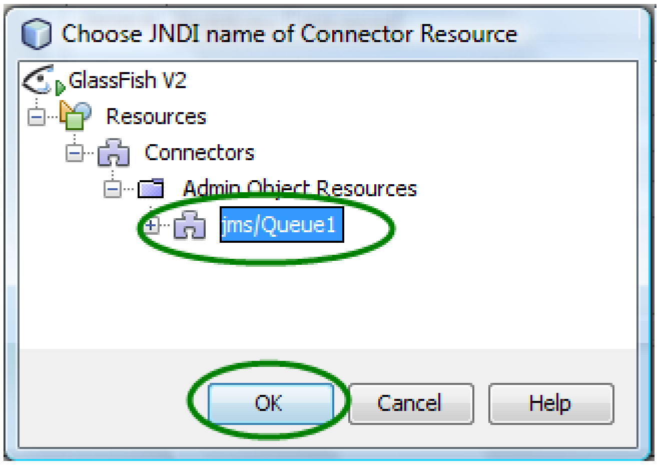
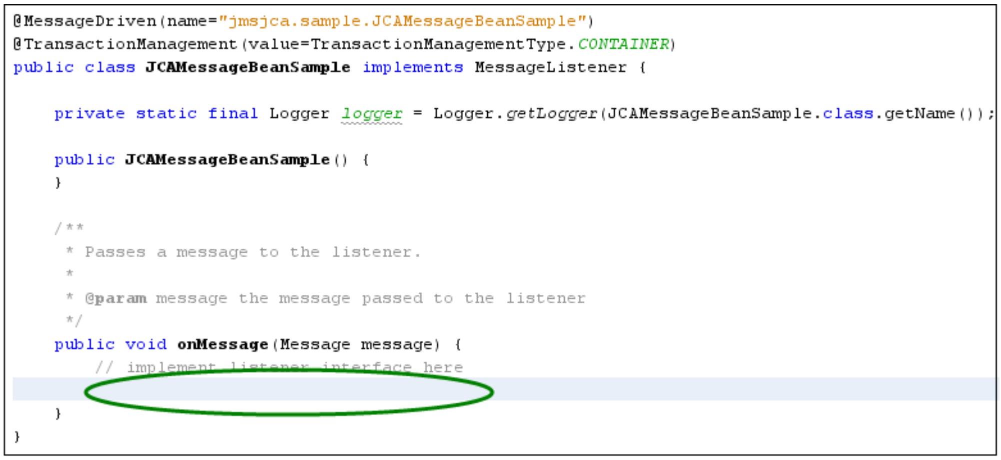


FIGURE 10 Connector Resource — Destination

11 Click Finish.

A Java source file is created and opened in the editor view. The source file is a skeleton file with most of the boilerplate code already generated, as shown below.



```
@MessageDriven(name="jmsjca.sample.JCAMessageBeanSample")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class JCAMessageBeanSample implements MessageListener {

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

    public JCAMessageBeanSample() {
    }

    /**
     * Passes a message to the listener.
     *
     * @param message the message passed to the listener
     */
    public void onMessage(Message message) {
        // implement listener interface here

    }
}
```

FIGURE 11 Java Source Code

Any JMS messages sent to the Queue1 destination are passed to the `onMessage(...)` method in this Java file. The login can be processed inside the `onMessage()` method as needed. Because the purpose of this task is to simply print out the message content of the JMS message (if the message is of type `javax.jms.TextMessage`), the implementation code would be similar to the following:

```
public void onMessage(Message message) {
    try {
        if (message instanceof javax.jms.TextMessage) {
            logger.log(Level.INFO, "JMS message conecnt is: " +
              ((javax.jms.TextMessage) message).getText());
        }
    } catch (JMSException ex) {
        Logger.getLogger(JCAMessageBeanSample.class.getName()).log(Level.SEVERE,
                         null, ex);
        return;
    }
}
```

Note – The above code has been wrapped to fit the page.

12 **Click Save when you are done editing the file.**

▼ To Test the Sample Code

1 **Right-click the Project node and select Build.**

2 **When the build process is complete, right-click the Project node and select Undeploy and Deploy.**

3 **Use a JMS client to send a text message to Queue1 on the to JMS server (located at `mq://localhost:7676`, by default).**

The contents of the message is logged in the server log file.

Sending a JMS Text Message

This topic provides instructions for sending a JMS message to a destination (*Queue2*). For purposes of this exercise, the message content to Queue2 is "Hello " concatenated with the message content received from the `onMessage()` method from Queue1. For more information about receiving JMS messages, see "Receiving a JMS Text Message" on page 5.

Perform the following steps to send a JMS text message:

- "To Create an Admin Object Resource" on page 16
- "To Define a JMS Session Instance" on page 16
- "Create a Reference to the Destination Object" on page 18
- "To Test the Sample Code" on page 20

▼ To Create an Admin Object Resource

1 **Start the GlassFish server and use a browser to connect to the Admin Console.**

The URL for the Admin Console is `http://`*HostName*`:`*PortNumber*. The default port number is 4848.

2 **In the left navigation bar, expand Resources and Connectors, and then select Admin Object Resources.**

3 **Click New.**

4 **For this exercise, enter the following values:**

- **JNDI Name** = jms/Queue2
- **Resource Type** = javax.jms.Queue
- **Resource Adapter** = sun-jms-adapter

5 **Click Next.**

6 **Enter `Queue2` in the Name property.**

This is the physical destination name of the resource.

7 **Click Finish.**

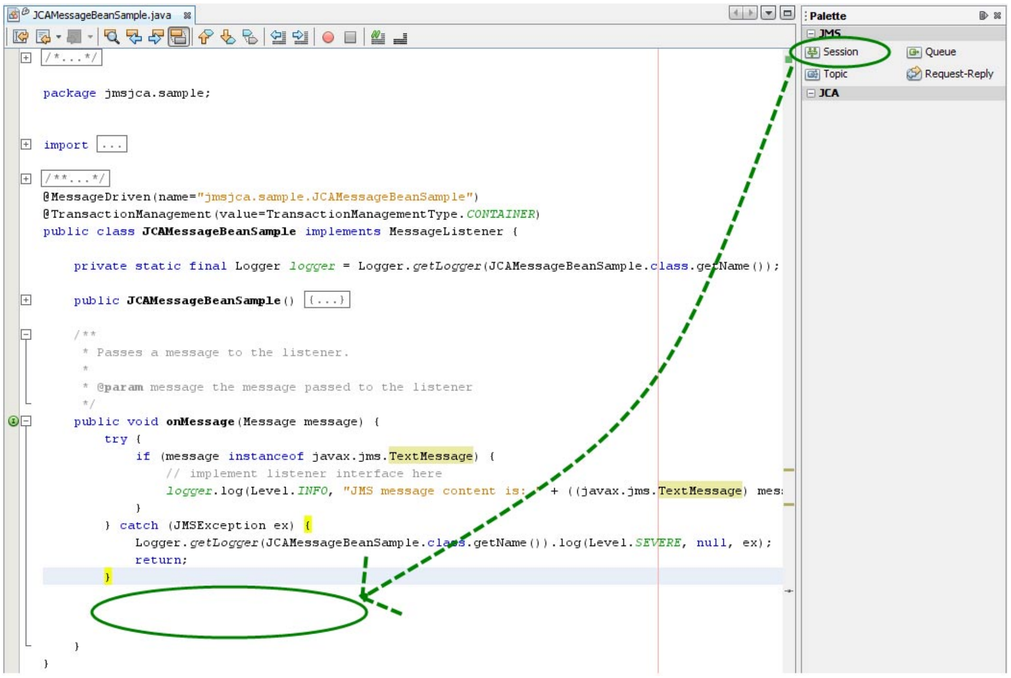
▼ To Define a JMS Session Instance

You need to create a JMS message, object, or message producer to send a message to Queue2 once the JMS message is received inside the MDB file of the *onMessage()* method.

1 **Launch the NetBeans IDE and open the Message-Driven Bean file you created in "To Create the JCA Message-Driven Bean" on page 9.**

The file is located in the Enterprise Beans node of the JMSJCASample project.

2 **Drag-and-drop the Session icon from the Palette panel on the right side to the inside of the `onMessage()` method, as shown in the figure below:**



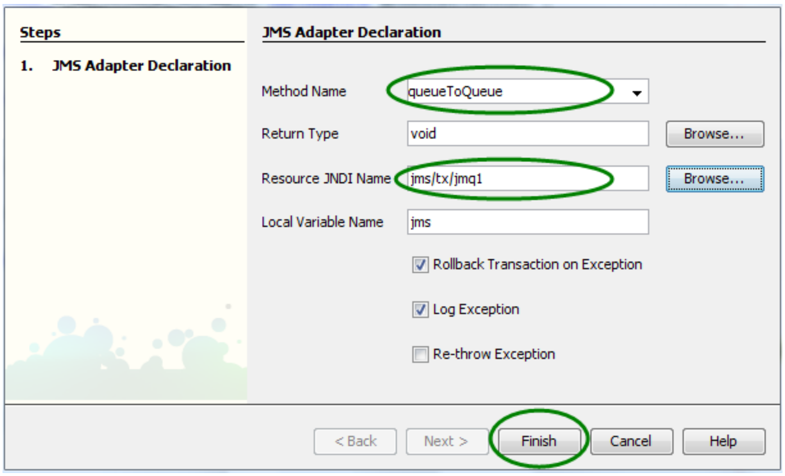
```
/*...*/

package jmsjca.sample;


import ...

/**...*/
@MessageDriven(name="jmsjca.sample.JCAMessageBeanSample")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class JCAMessageBeanSample implements MessageListener {

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

    public JCAMessageBeanSample() {...}

    /**
     * Passes a message to the listener.
     *
     * @param message the message passed to the listener
     */
    public void onMessage(Message message) {
        try {
            if (message instanceof javax.jms.TextMessage) {
                // implement listener interface here
                logger.log(Level.INFO, "JMS message content is: " + ((javax.jms.TextMessage) mes
            }
        } catch (JMSException ex) {
            Logger.getLogger(JCAMessageBeanSample.class.getName()).log(Level.SEVERE, null, ex);
            return;
        }



    }
}
```

FIGURE 12 JCA Message Bean Sample — Session

The JCA Wizard dialog box appears.

3 For this exercise, enter the following values:

- **Method Name** = queueToQueue
- **Resource JNDI Name** = jms/tx/jmq1


Steps

1. JMS Adapter Declaration

JMS Adapter Declaration

Method Name: queueToQueue

Return Type: void

Resource JNDI Name: jms/tx/jmq1

Local Variable Name: jms

[✓] Rollback Transaction on Exception

[✓] Log Exception

[ ] Re-throw Exception

FIGURE 13 JCA Adapter Declaration

4 **Click Finish.**

Several Java code fragments is generated as a result, in particular the `queueToQueue(...)` method, which can be implemented to process the incoming message.

5 **Save the MDB file.**

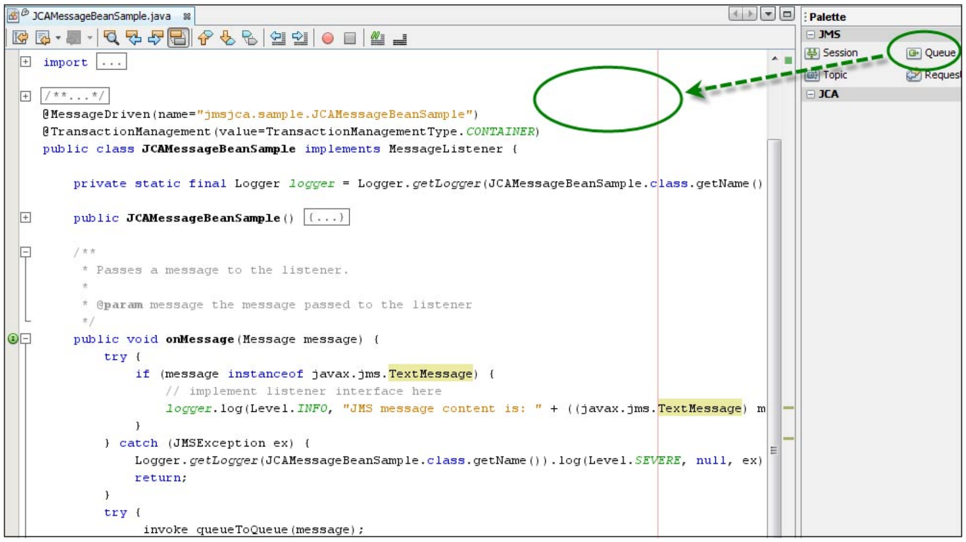
▼ Create a Reference to the Destination Object

Creating a reference to the destination object allows a message to be sent to the destination object in the Java code. For this exercise, the destination object is Queue2.

1 **In the NetBeans IDE, open the Message-Driven Bean file you created in "To Create the JCA Message-Driven Bean" on page 9.**

The file is located in the Enterprise Beans node of the JMSJCASample project.

2 **Drag-and-drop the Queue icon from the Palette panel on the right to any place in Java editor, as shown below.**



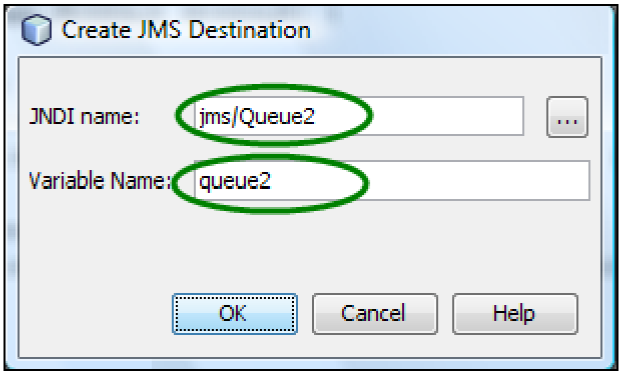
```
import ...

/**...*/
@MessageDriven(name="jmsjca.sample.JCAMessageBeanSample")
@TransactionManagement(value=TransactionManagementType.CONTAINER)
public class JCAMessageBeanSample implements MessageListener {

    private static final Logger logger = Logger.getLogger(JCAMessageBeanSample.class.getName()

    public JCAMessageBeanSample() {...}

    /**
     * Passes a message to the listener.
     *
     * @param message the message passed to the listener
     */
    public void onMessage(Message message) {
        try {
            if (message instanceof javax.jms.TextMessage) {
                // implement listener interface here
                logger.log(Level.INFO, "JMS message content is: " + ((javax.jms.TextMessage) m
            }
        } catch (JMSException ex) {
            Logger.getLogger(JCAMessageBeanSample.class.getName()).log(Level.SEVERE, null, ex)
            return;
        }
        try {
             invoke queueToQueue(message);
```

FIGURE 14 JCA Message Bean Sample — Queue

The Create JMS Destination dialog box appears.

3 **For this exercise, enter the following information into the fields:**

- **JNDI Name** = jms/Queue2 (You can select this value by clicking the ellipsis button and expanding the tree.)
- **Variable Name** = queue2



JNDI name: jms/Queue2

Variable Name: queue2

FIGURE 15 Create JMS Destination

4 **Click OK.**

5 **Write the actual code to create a new JMS message and send it to Queue2.**

The code fragment inside the `queueToQueue(...)` method should be similar to the example shown below:

```
private void queueToQueue(Message message, javax.jms.Session jmsSession)
           throws java.lang.Exception {
    if (message instanceof javax.jms.TextMessage)  {
        String oldContent = ((javax.jms.TextMessage) message).getText();
        javax.jms.TextMessage newMessage = jmsSession.createTextMessage("Hello "
               + oldContent);
        jmsSession.createProducer(queue2).send(newMessage);
    }
}
```

Note – The above code has been wrapped to fit onto the page.

6 **Save the changes.**

▼ To Test the Sample Code

To test that JMS messages are being properly passed from Queue1 to Queue2, complete the following steps.

1 **Right-click on the Project node and select Build.**

2 **After the build process is complete, right-click on the Project node, and then select Undeploy and Deploy.**

3 **Use your preferred JMS client to send a text message to Queue1 (located at `mq://localhost:7676,`).**

4 **Use another JMS client (or the same client) to receive a text message from Queue2 in the JMS server (located at `mq://localhost:7676,`).**

Initiating a Request-Reply Transaction

JMS messaging solutions need to satisfy the requirements of operating on a fire-and-forget or a store-and-forward basis. This messaging infrastructure is used to deliver each message to the intended recipient whether that recipient is active at the time of send or not. In a request-reply pattern, messages are delivered to the messaging system, which immediately acknowledges that it has taken the responsibility for delivery to the ultimate recipient. That delivery might take some time if the recipient is not active for a period or might not take place at all if the recipient never appears.

Perform the following steps to initiate a request-reply transaction:

- "To Create the EJB Module Project" on page 21
- "To Create a Message Driven Bean" on page 21
- "To Create a JCA Message-Driven Bean for the Destination" on page 27

▼ To Create the EJB Module Project

1 **From the File menu, select New Project.**

The New Project dialog box appears.

2 **Select Java EE under Categories, and then select EJB Module under Projects.**

3 **Click Next.**

The Name and Location window appears.

4 **Enter a unique Project Name and the location to store the project files.**

5 **Click Next.**

The Server and Settings window appears.

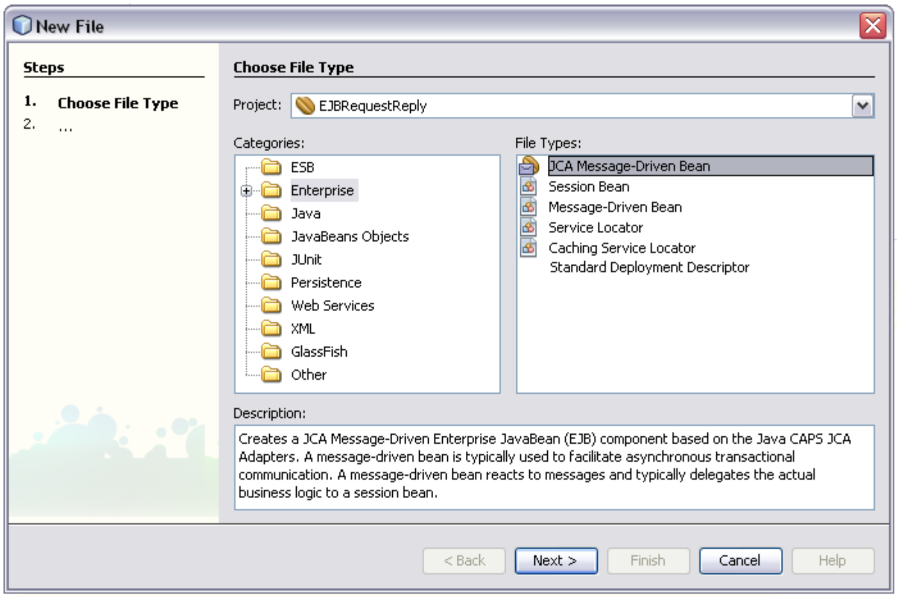
6 **Accept the default settings for the server and click Finish.**

The new project is created.

▼ To Create a Message Driven Bean

1 **Right-click the new project, and then select New -> Other.**

The New File Wizard appears.



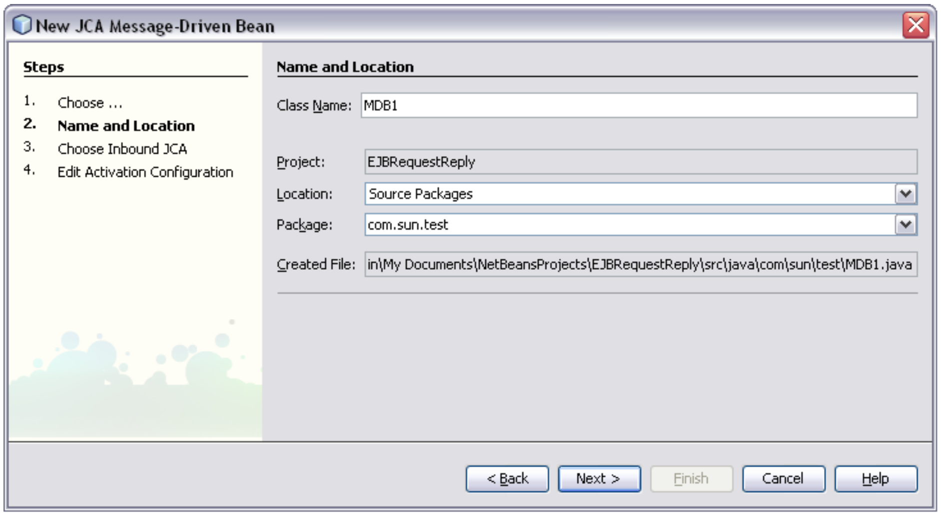
Creates a JCA Message-Driven Enterprise JavaBean (EJB) component based on the Java CAPS JCA Adapters. A message-driven bean is typically used to facilitate asynchronous transactional communication. A message-driven bean reacts to messages and typically delegates the actual business logic to a session bean.

FIGURE 16 New JCA Message-Driven Bean

2 Select Java EE under Categories, and then select JCA Message-Driven Bean under File Types .

3 Click Next.

The Name and Location window appears.



Class Name: MDB1

Project: EJBRequestReply

Location: Source Packages

Package: com.sun.test

Created File: in\My Documents\NetBeansProjects\EJBRequestReply\src\java\com\sun\test\MDB1.java

FIGURE 17 Configuring the Message-Driven Bean

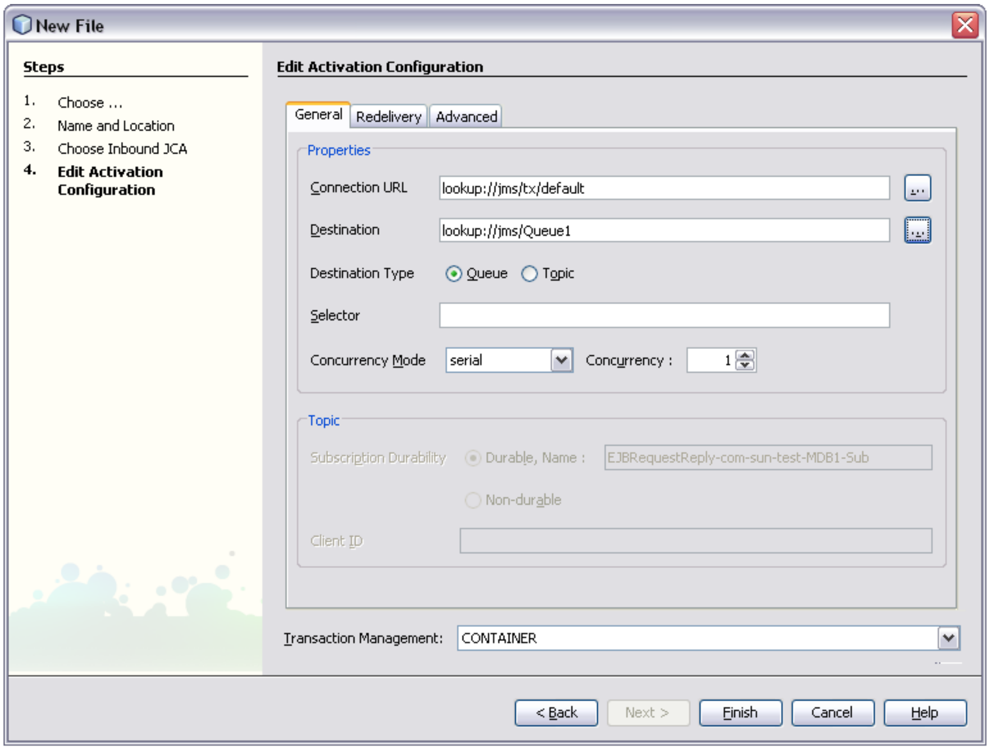
4 **Enter a unique Class Name and a valid Package name.**

5 **Click Next.**

The Choose Inbound JCA window appears.

6 **Select the JMS Adapter and click Next.**

The Edit Activation Configuration window appears.



Edit Activation Configuration

Connection URL: lookup://jms/tx/default

Destination: lookup://jms/Queue1

Destination Type: [✓ Queue] Topic

Selector:

Concurrency Mode: serial

Concurrency : 1

Subscription Durability: [✓ Durable, Name : EJBRequestReply-com-sun-test-MDB1-Sub] Non-durable

Client ID:

Transaction Management: CONTAINER

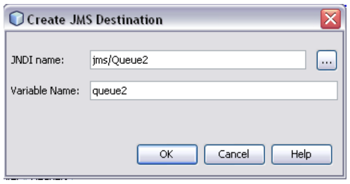
FIGURE 18 Edit Activation Configuration

7 Set the Destination lookup to the JNDI Name of the Queue and click Finish.

A new Message-Driven Bean is created.

8 Drag a Queue rom the Palette panel on the right into the Java Editor.

The Create JMS Destination dialog box appears.



JNDI name: jms/Queue2

Variable Name: queue2

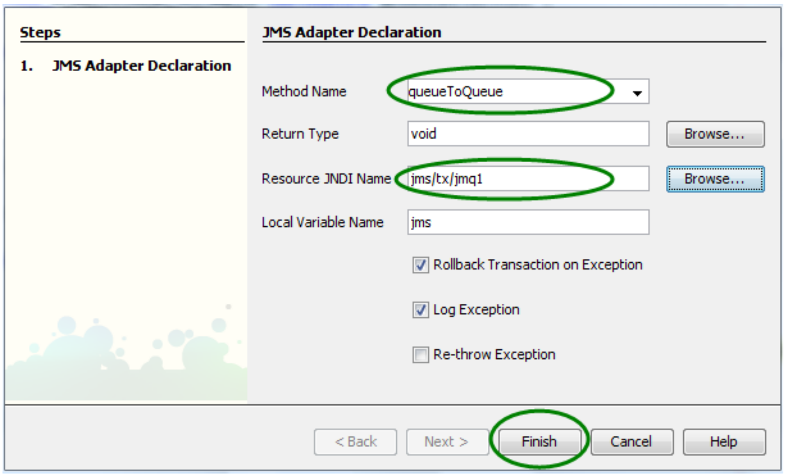
FIGURE 19 JMS Destination

9 **Enter a valid JNDI Name, Variable Name, and click OK.**

The Java code for the Queue instance is populated into the Java Editor. Repeat the above steps for as many Queues that are needed.

10 **Drag a Session from the Palette panel in the right into the `onMessage()` method in the Java Editor.**

The JMS Adapter Declaration dialog box appears.



Steps

1. JMS Adapter Declaration

JMS Adapter Declaration

Method Name: queueToQueue

Return Type: void

Resource JNDI Name: jms/tx/jmq1

Local Variable Name: jms

[✓] Rollback Transaction on Exception

[✓] Log Exception

[ ] Re-throw Exception

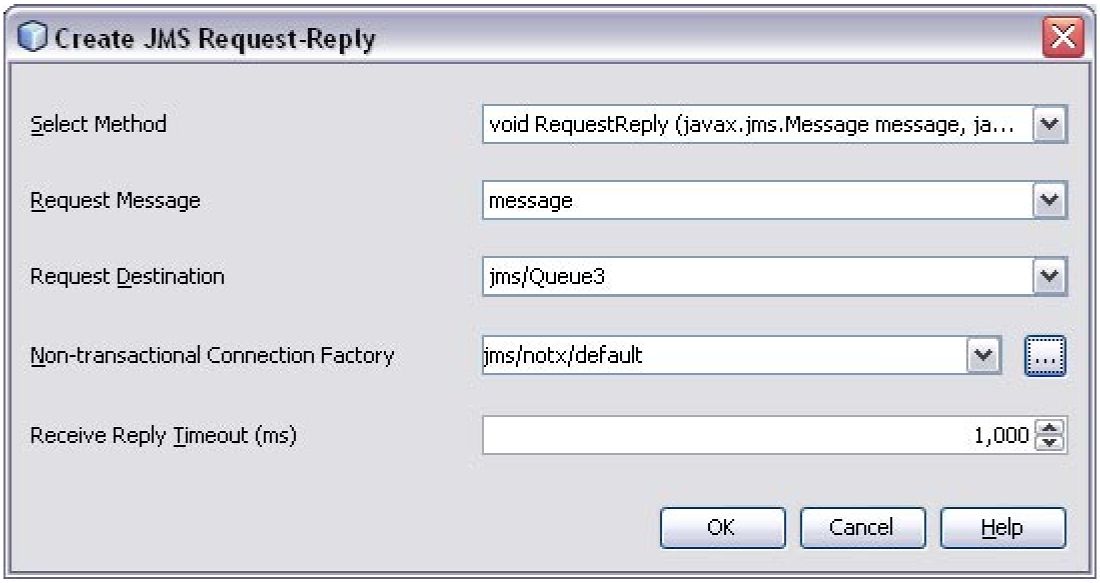
FIGURE 20 JMS Adapter Declaration

11 **Enter a valid method name, such as `RequestReply` and click Finish.**

The Java code for the JMS Session is populated into the Java Editor.

12 **Drag a Request-Reply from the Palette panel on the right into the new method.**

The Create JMS Request-Reply dialog box appears.



Select Method: void RequestReply (javax.jms.Message message, ja...
Request Message: message
Request Destination: jms/Queue3
Non-transactional Connection Factory: jms/notx/default
Receive Reply Timeout (ms): 1,000

FIGURE 21 JMS Request-Reply

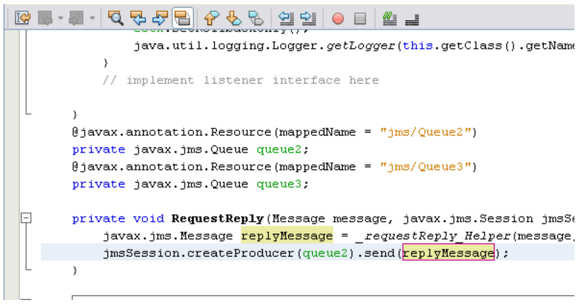
13 Select following values for the fields:

- **Select Method** – Select the method you specified earlier on the JMS Adapter Declaration dialog box.
- **Request Message** – Select `message`.
- **Request Destination** – Select the JMS queue or topic you created for the adapter.
- **Non-transactional Connection Factory** – Select a connection factory that contains “notx” in the name.

14 Click OK.

15 In the Request-Reply method, enter the following code beneath the first line of code:

```
jmsSession.createProducer(queue2).send(replyMessage);
```



```
            java.util.logging.Logger.getLogger(this.getClass().getName
        }
        // implement listener interface here

    }
    @javax.annotation.Resource(mappedName = "jms/Queue2")
    private javax.jms.Queue queue2;
    @javax.annotation.Resource(mappedName = "jms/Queue3")
    private javax.jms.Queue queue3;

    private void RequestReply(Message message, javax.jms.Session jmsSe
        javax.jms.Message replyMessage = _requestReply_Helper(message,
        jmsSession.createProducer(queue2).send(replyMessage);
    }
```

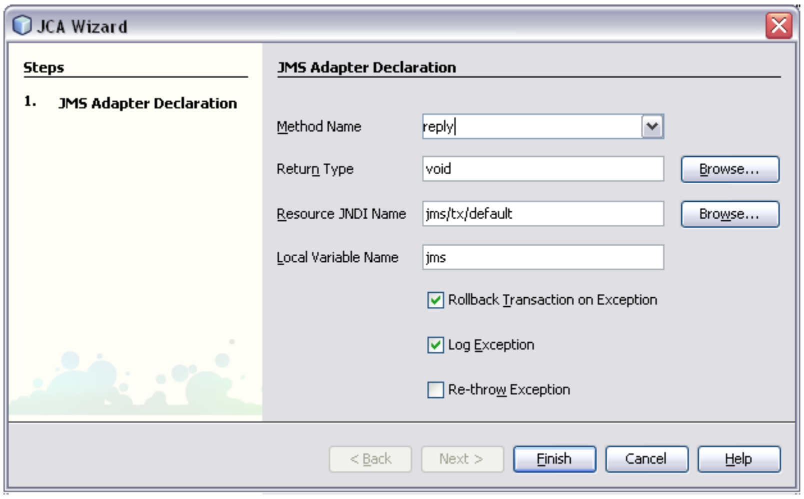
FIGURE 22 Request-Reply Method

16 Save the file.

▼ To Create a JCA Message-Driven Bean for the Destination

1 From the NetBeans Palette window, drag an instance of the JMS Session into the `onMessage()` method in the Java Editor.

The JMS Adapter Declaration dialog box appears.



JCA Wizard

Steps

1. JMS Adapter Declaration

JMS Adapter Declaration

Method Name: reply

Return Type: void

Resource JNDI Name: jms/tx/default

Local Variable Name: jms

[✓] Rollback Transaction on Exception

[✓] Log Exception

[ ] Re-throw Exception

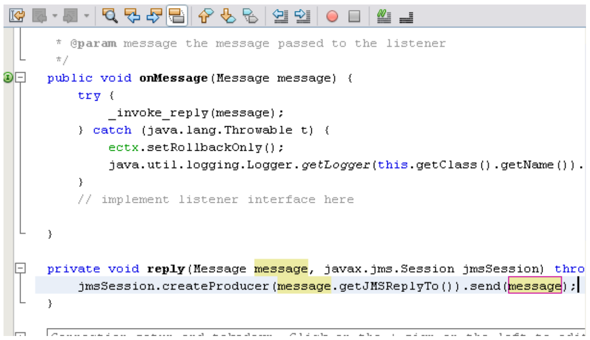
FIGURE 23 Reply Method

2 Enter `reply` as the method name and click Finish.

The Java code for the JMS Session is populated into the Java Editor.

3 **In the reply method enter the following code:**

```
jmsSession.createProducer(message.getJMSReplyTo()).send(message);
```



```
     * @param message the message passed to the listener
     */
    public void onMessage(Message message) {
        try {
            _invoke_reply(message);
        } catch (java.lang.Throwable t) {
            ectx.setRollbackOnly();
            java.util.logging.Logger.getLogger(this.getClass().getName()).
        }
        // implement listener interface here

    }

    private void reply(Message message, javax.jms.Session jmsSession) thro
        jmsSession.createProducer(message.getJMSReplyTo()).send(message);
    }
```

FIGURE 24 Reply Method in the Java Editor

This code sends the incoming message to the reply destination.

4 **Save the file.**

5 **Build and deploy the project.**