Oracle® Java CAPS JMS JCA Wizard User's Guide

Part No: 821–2652
March 2011

ORACLE

Copyright © 2009, 2011, Oracle and/or its affiliates. All rights reserved.

License Restrictions Warranty/Consequential Damages Disclaimer

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

Warranty Disclaimer

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Restricted Rights Notice

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

Hazardous Applications Notice

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group in the United States and other countries.

Third Party Content, Products, and Services Disclaimer

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

120126@25097

Contents

Using the JMS JCA Wizard 5
About the JMS JCA Wizard 5
Receiving a JMS Text Message 5
▼ To Create an Admin Object Resource 6
▼ To Create the EJB Module Project 7
▼ To Create the JCA Message-Driven Bean 9
▼ To Test the Sample Code 15
Sending a JMS Text Message 15
▼ To Create an Admin Object Resource 16
▼ To Define a JMS Session Instance 16
▼ Create a Reference to the Destination Object 18
▼ To Test the Sample Code 20
Initiating a Request-Reply Transaction 20
▼ To Create the EJB Module Project 21
▼ To Create a Message Driven Bean 21
▼ To Create a JCA Message-Driven Bean for the Destination 27

Using the JMS JCA Wizard

The following topics provide information for using the JMS JCA Wizard, 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 so on.

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

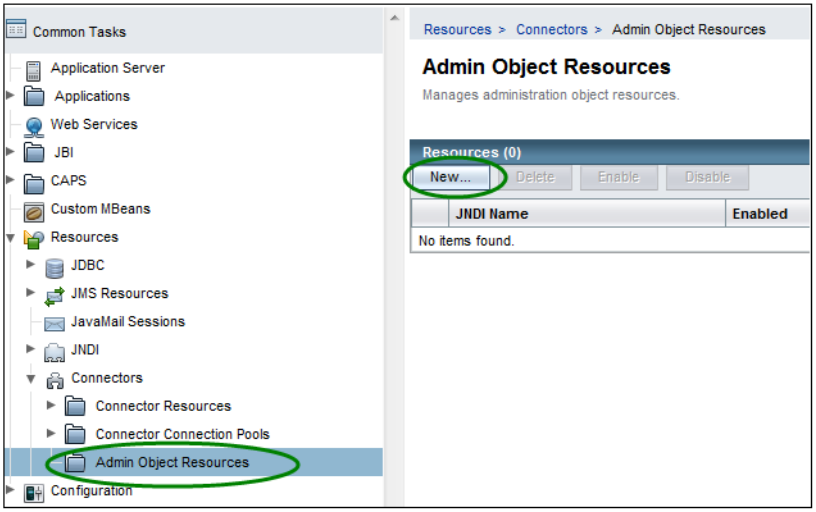
For this example, the message is being received from *Queue1*, so you need to create the corresponding JMS Queue object resource in GlassFish.

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 , expand Connectors, and then select Admin Object Resources.

FIGURE 1 Admin Object Resources



3 Click New.

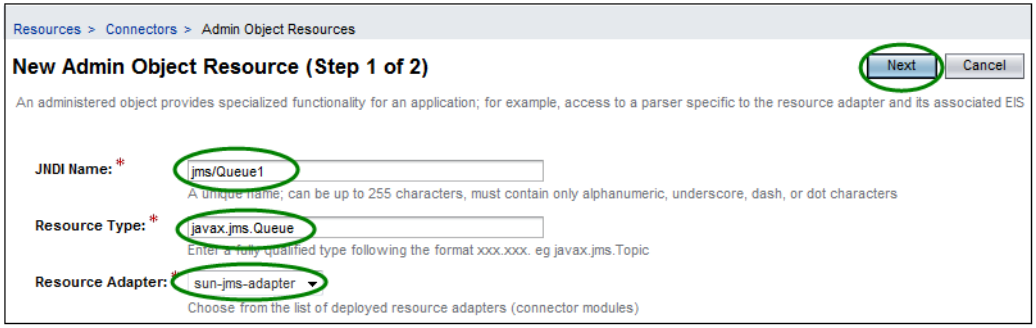
The New Admin Object Resource window appears (Step 1 of 2).

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

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

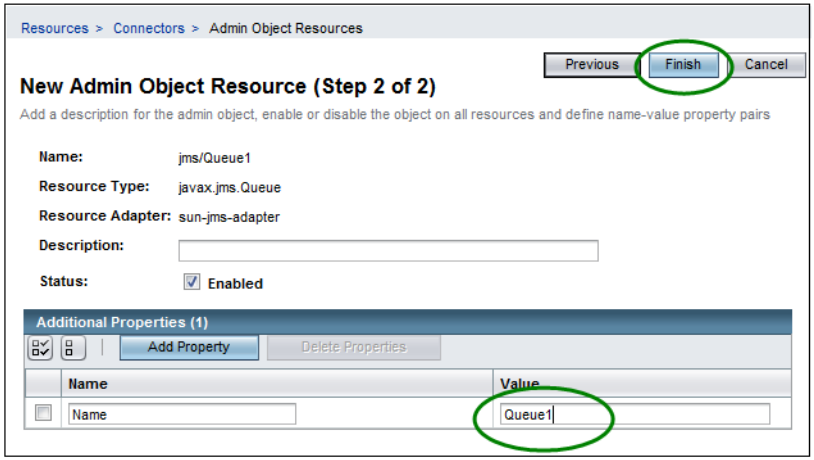


5 **Click Next.**

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

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

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



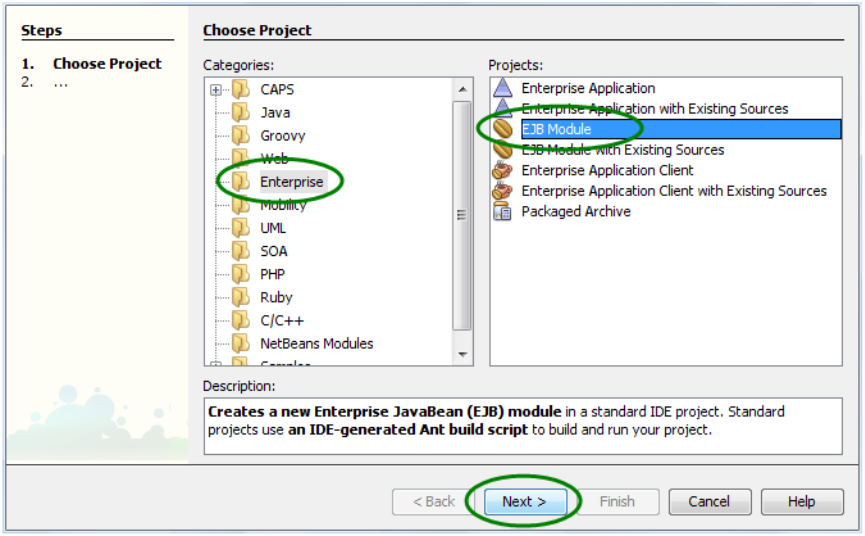
7 **Click Finish.**

▼ 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.

FIGURE 4 Choose New Project



3 Click Next.

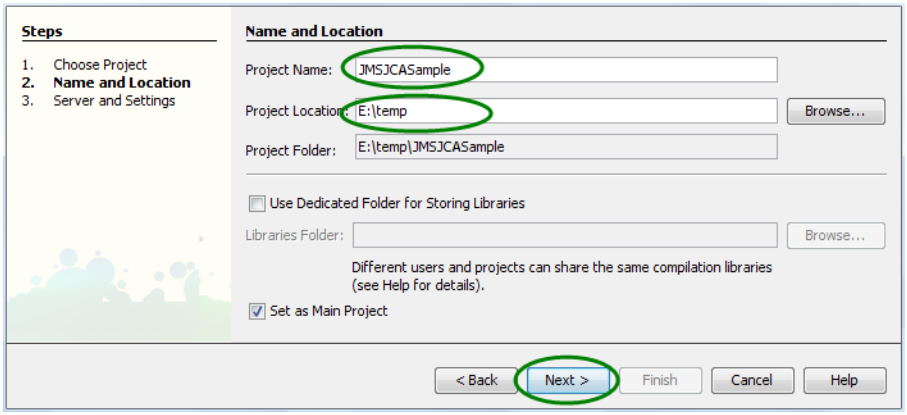
The Name and Location window appears.

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

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.**

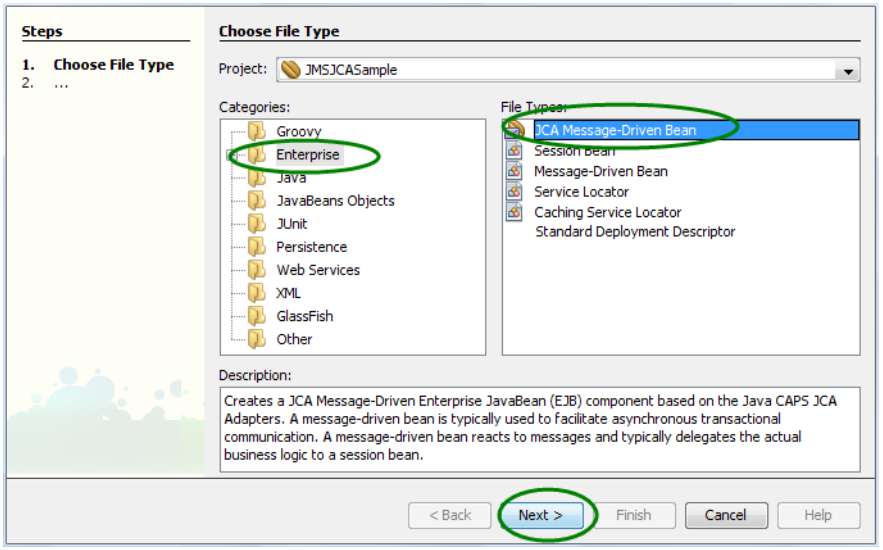
7 **Click Finish.**

▼ 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.**

FIGURE 6 Choose JCA Message-Driven Bean



3 Click Next.

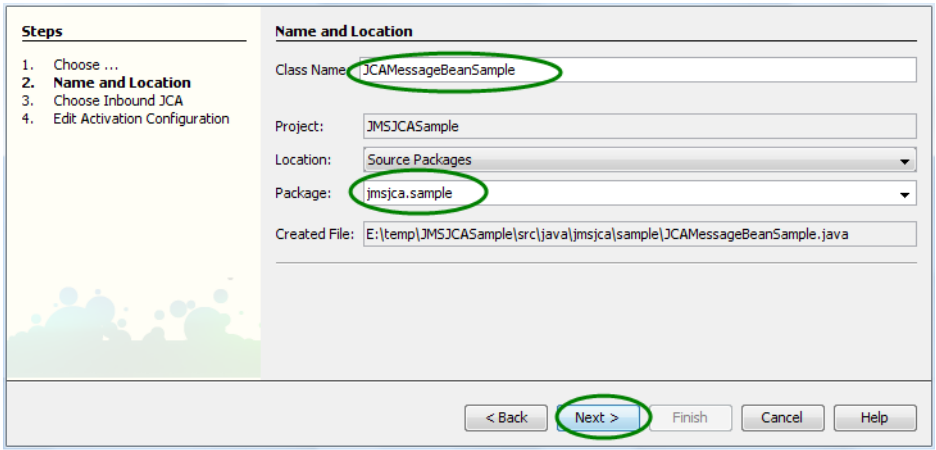
The JCA Message-Driven Bean Name and Location window appears.

4 Enter the Name and Location fields.

For this exercise, enter the following values:

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

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.**

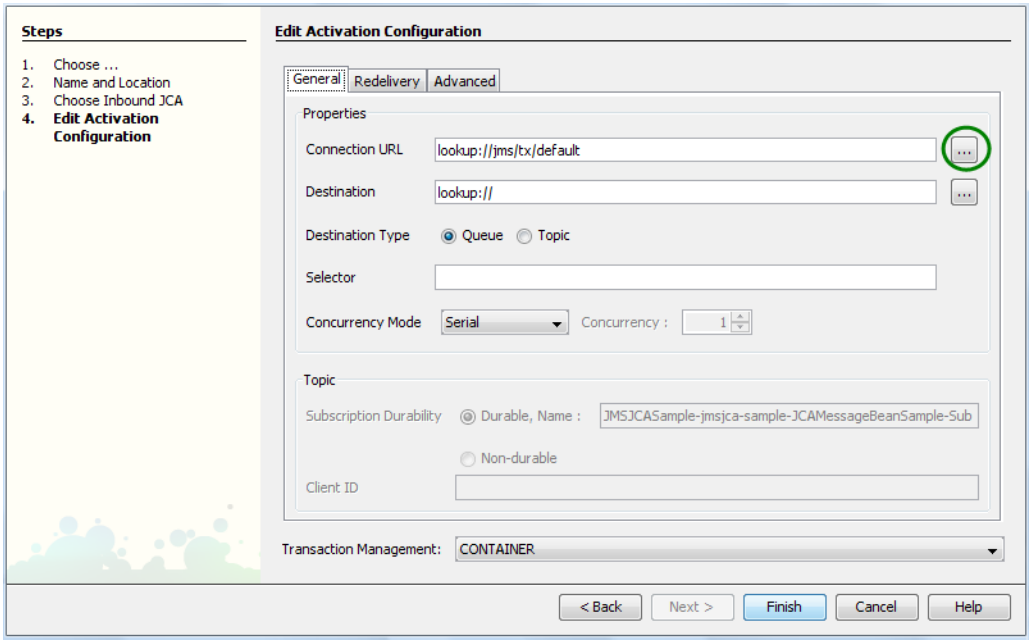
Note – Currently only JMS Adapter can be selected in the window.

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 are needed for our sample code to work.

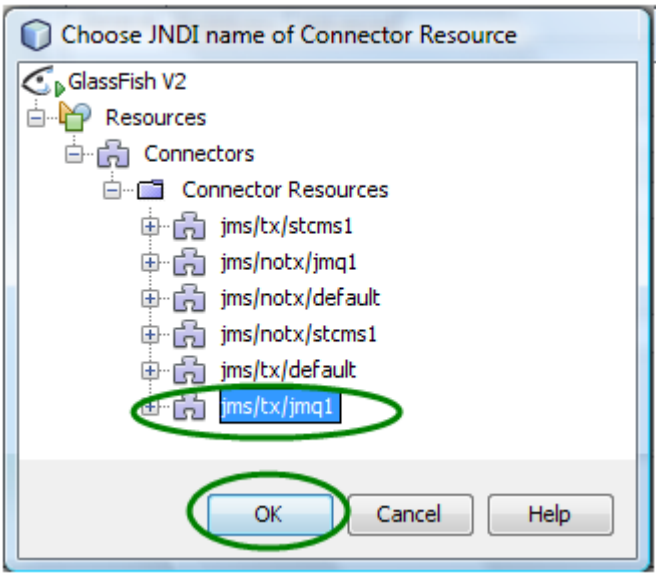
FIGURE 8 Edit Activation Configuration



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

This resource connects the embedded 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



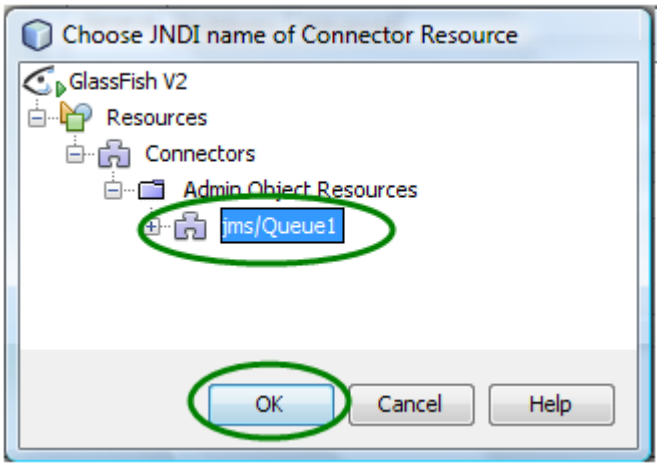
9 **Click on the ellipsis button next to Destination box.**

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.

FIGURE 11 Java Source Code

```
@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

    }
}
```

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 content 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

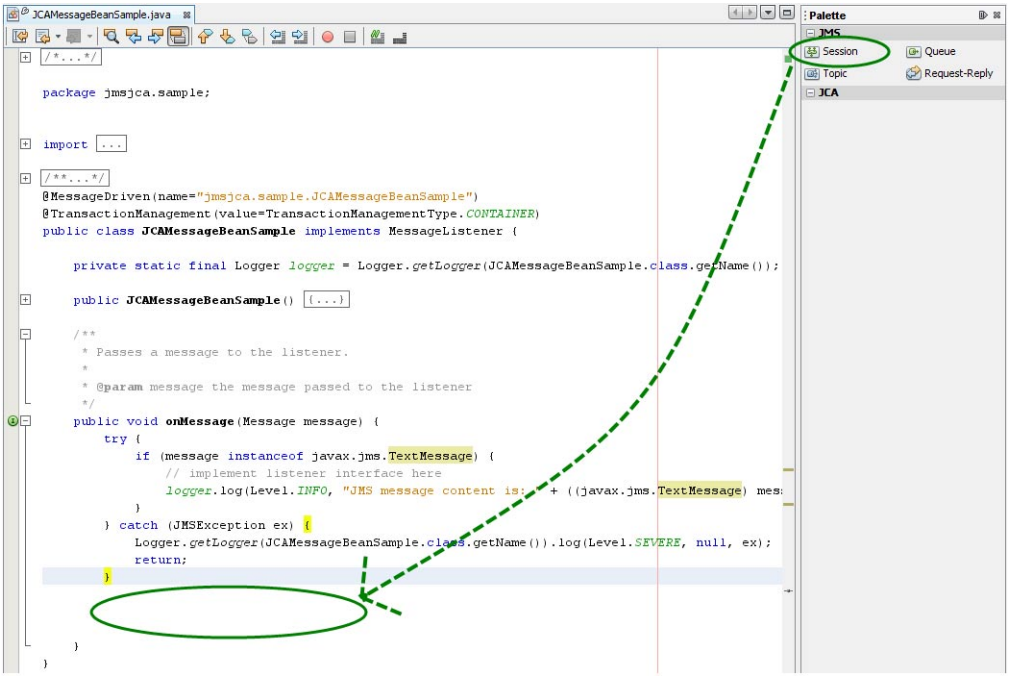
You need to create a JMS message, object, or message producer to send a message to Queue2 once the JMS message is received inside the MDB file of the *onMessage()* method.

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:**

FIGURE 12 JCA Message Bean Sample — Session

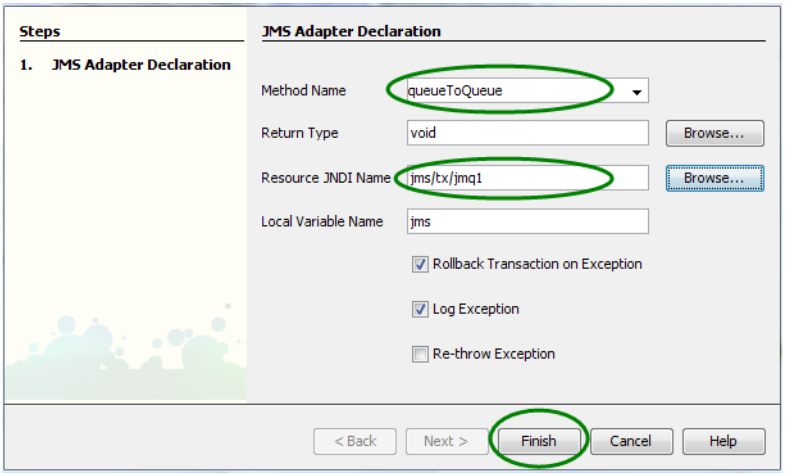


The JCA Wizard dialog box appears.

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

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

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

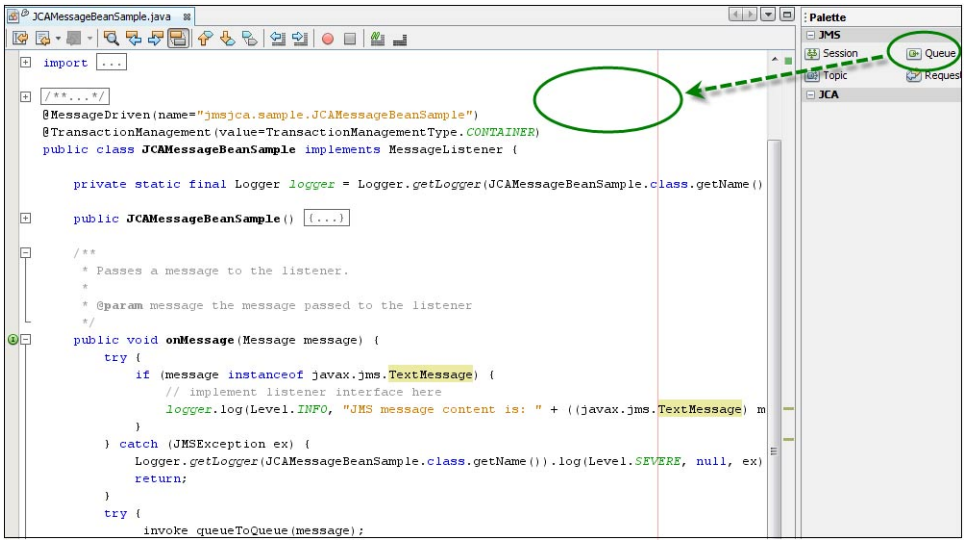
Creating a reference to the destination object allows a message to be sent to the destination object in the Java code. For this exercise, the destination object is Queue2.

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.**

FIGURE 14 JCA Message Bean Sample — Queue

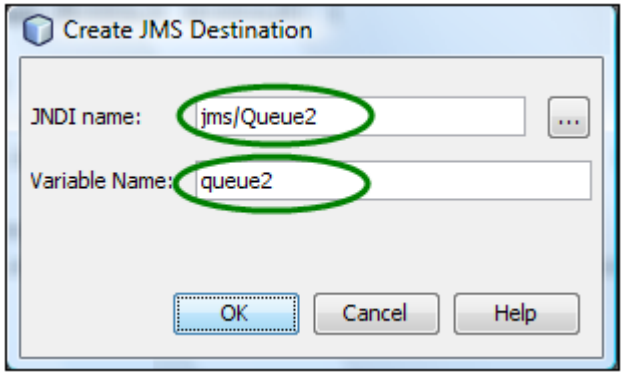


The Create JMS Destination dialog box appears.

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

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

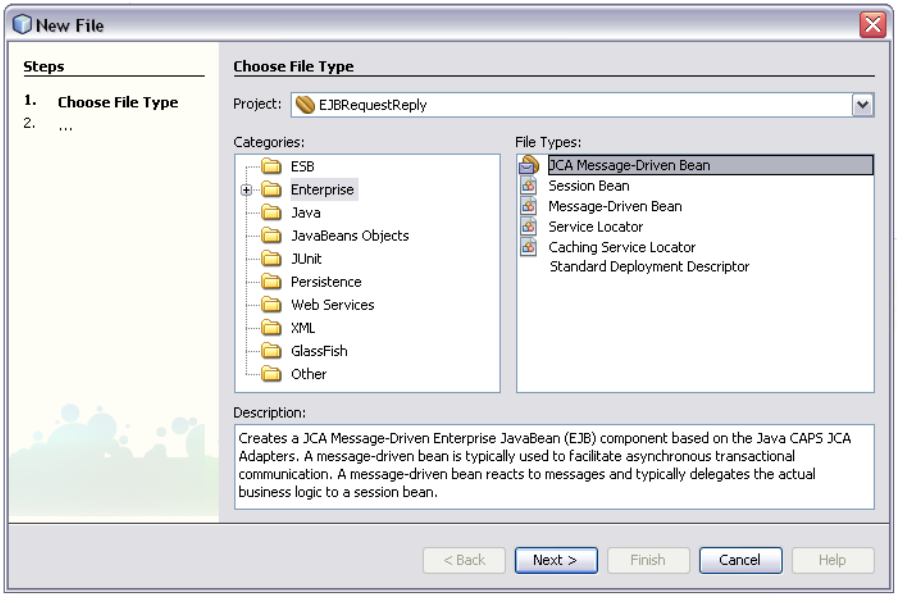
▼ To Create the EJB Module Project

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.

FIGURE 16 New JCA Message-Driven Bean

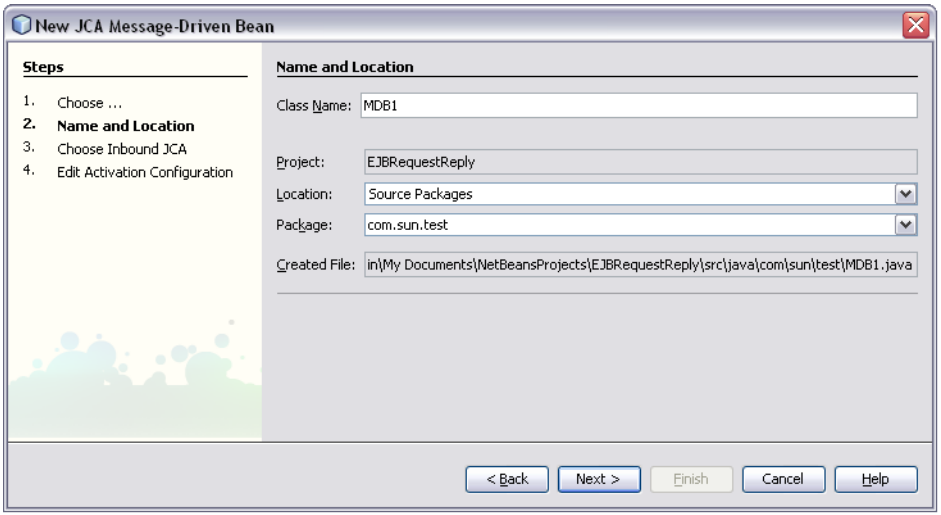


2 **Select Java EE under Categories, and then select JCA Message-Driven Bean under File Types .**

3 **Click Next.**

The Name and Location window appears.

FIGURE 17 Configuring the Message-Driven Bean



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

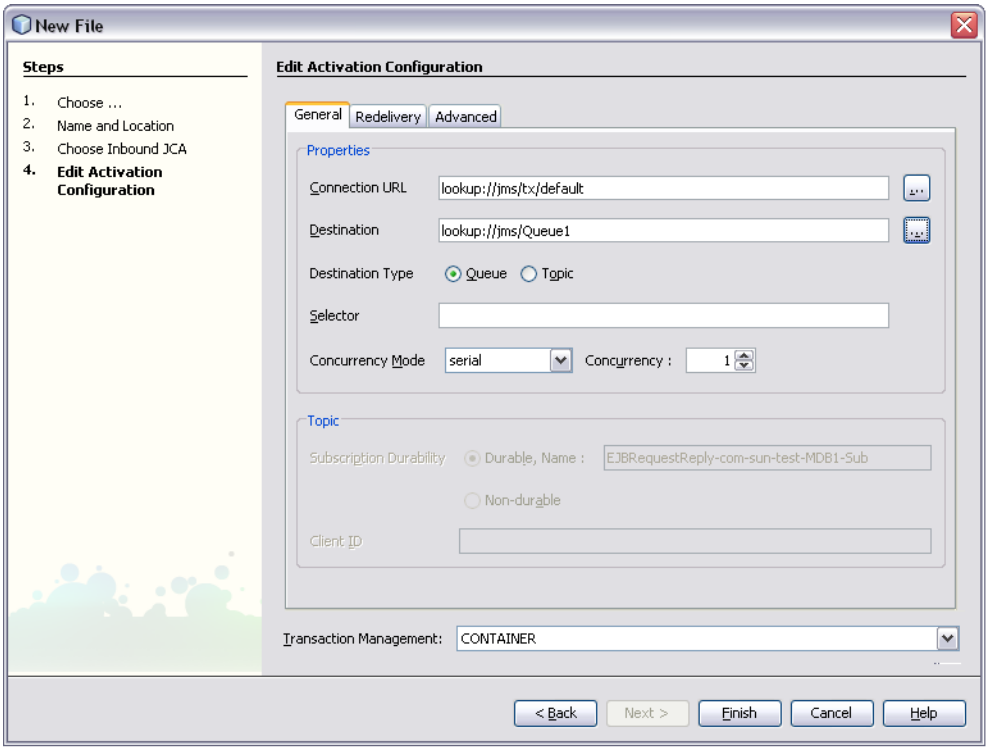
5 Click Next.

The Choose Inbound JCA window appears.

6 Select the JMS Adapter and click Next.

The Edit Activation Configuration window appears.

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 from the Palette panel on the right into the Java Editor.

The Create JMS Destination dialog box appears.

FIGURE 19 JMS Destination

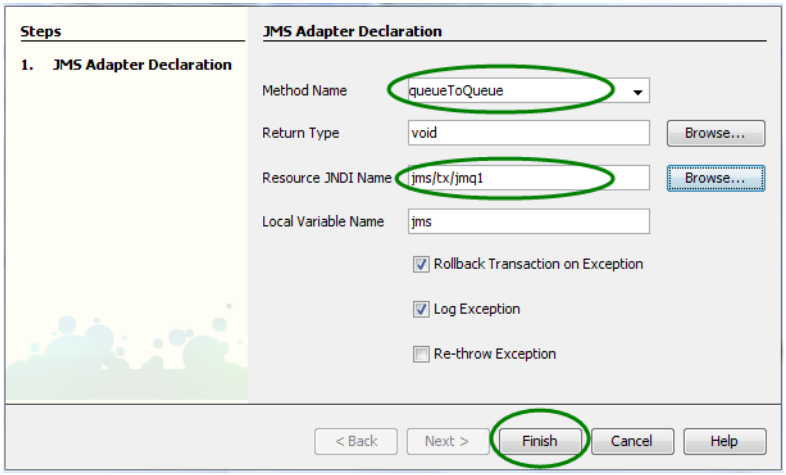
9 Enter a valid JNDI Name, Variable Name, and click OK.

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.

FIGURE 20 JMS Adapter Declaration



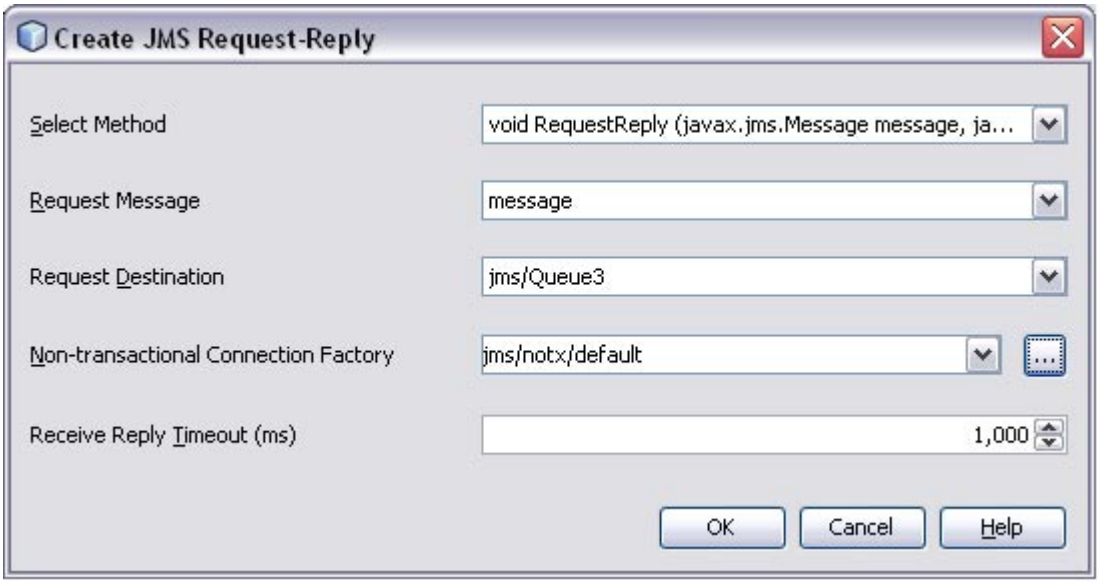
11 Enter a valid method name, such as `RequestReply` and click Finish.

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.

FIGURE 21 JMS Request-Reply



13 Select following values for the fields:

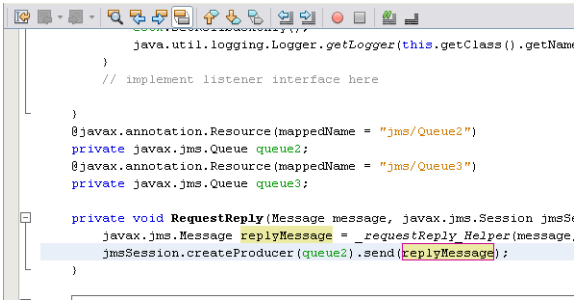
- **Select Method** – Select the method you specified earlier on the JMS Adapter Declaration dialog box.
- **Request Message** – Select **`message`**.
- **Request Destination** – Select the JMS queue or topic you created for the adapter.
- **Non-transactional Connection Factory** – Select a connection 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);
```

FIGURE 22 Request-Reply Method



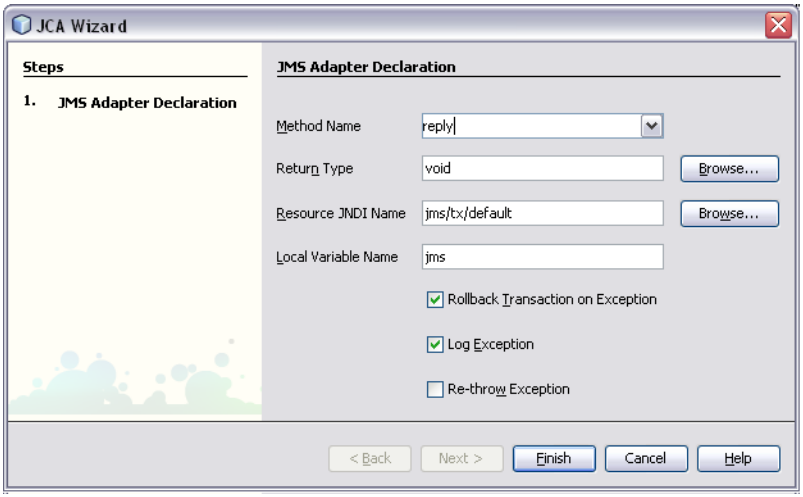
16 **Save the file.**

▼ 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.

FIGURE 23 Reply Method



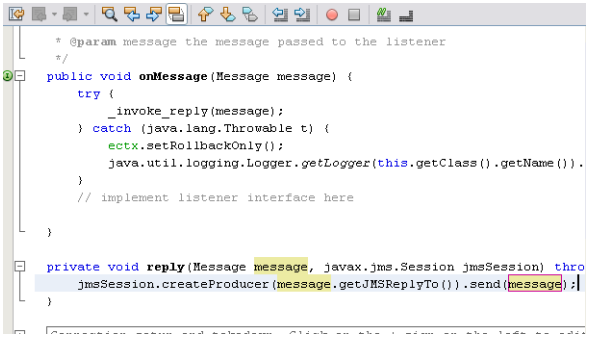
2 **Enter `reply` as the method name and click Finish.**

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);
```

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.**