Demo on JMS Web service

This demo is to show the students how to use the Oracle JDeveloper 10*g* to develop JMS Web service.

Demo:
In the demo, you will first create a Message Driven Bean (MDB). MDB is used to process the incoming messages and outgoing messages. The MDB should have an onMessage() method which is invoked when a message is sent to or retrieved from a queue/topic. More explanation about the MDB is provided along with the code.

Generate The JMS Web service, which is mainly used to put messages on and take messages off JMS destinations. When the client supplies an XML element, which has to be put to a queue or topic, the Web service takes the message and enqueues the appropriate JMS queue. When the message is put on the queue the onMessage method of the MDB is invoked. Hence, the Web service and the MDB have to be associated with the appropriate queues. In the example used for the demo the MDB picks up the message from one queue and places it on another queue from where it can be consumed by a message consumer.

Follow these steps to create an MDB:

1. Create a Workspace, `jmsws.jws`, and create a project `Project1.jpr`.
2. Right click on project and select New. Expand the Business Tier node in the Categories. Select Enterprise JavaBeans from Categories and select Message Driven Bean from Items. Click the OK button.
3. You will see the Enterprise JavaBean Wizard. Click the Next button to proceed with the wizard. Provide the name, `jmsmdb`, for EJB Name. Click the Next button. Click the Next button again and then click the Finish button to generate the skeleton for MDB. Observe that the `jmsmdbBean` is generated and is shown in the System-Navigator.
4. In JDeveloper, right click on project and select Project Properties. Click on Libraries and include Oracle XML Parser v2.
5. Edit the MDB to import the packages.
   ```
   import javax.naming.*;
   import javax.jms.*;
   import oracle.xml.parser.v2.XMLElement;
   ```
6. Add the following code to the onMessage() method of MDB.
   NOTE: The commented lines need not be added, you can use them for explanation. For your convenience mdb.txt file in the same folder contains the code without commented lines if you want to copy and paste it.
   ```
   try
     {
   // The onMessage() method should have one argument of type javax.jms.Message.
   // The JMS Web service only supports messages of type ObjectMessage, so the
   // you should cast the incoming JMS Web Service message to an ObjectMessage.
      XMLElement el = (XMLElement) ((ObjectMessage) msg).getObject();
   ```

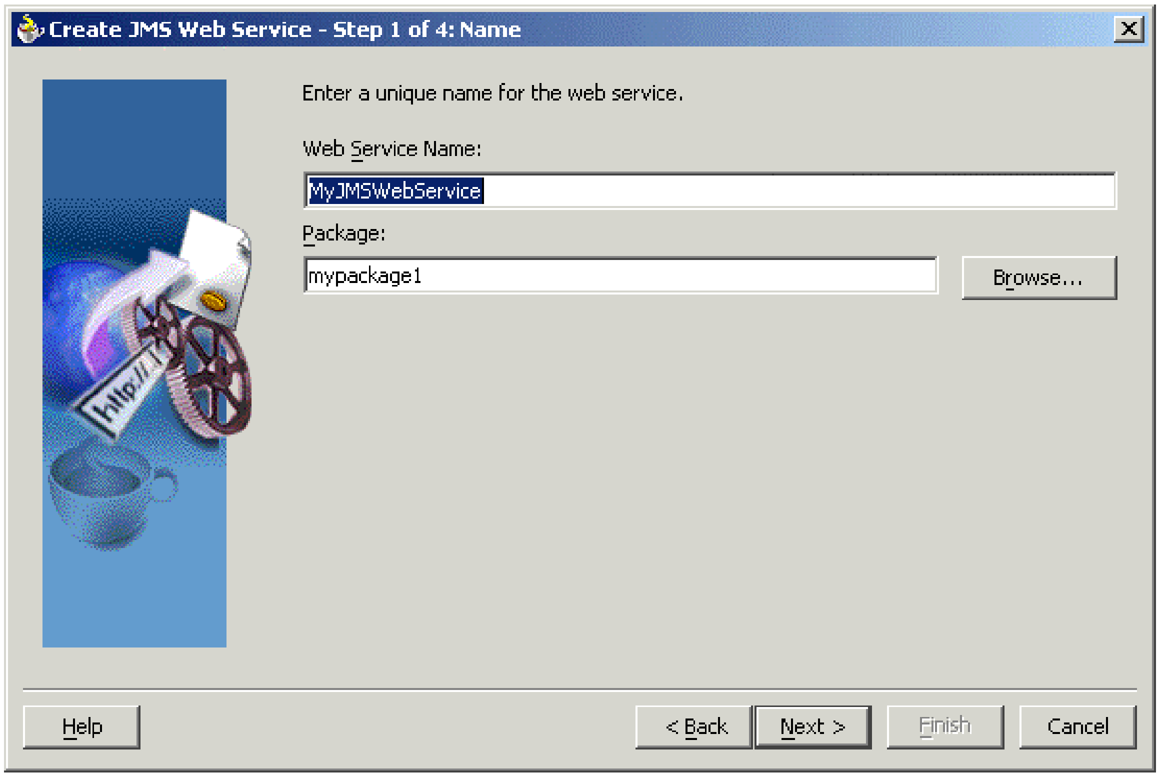
```
 System.out.println("onMessage method invoked: printing the message");
  el.print(System.out);

//Obtain an InitialContext object to look up for the connection factory.
//Connection factory is an object that encapsulates a set of configuration
//parameters that has been defined for the JMS destinations.
  Context ctx = new InitialContext();

// Get the queue to which the message has to be put and display the name of
// the queue. The information about the queue and the connection factory is
// provided in the jms.xml file.
  Queue queue = (Queue) msg.getJMSReplyTo();
  System.out.println("queue name is ");
  System.out.println(queue.getQueueName());

// Get the connection factory name and display it
  String qcfName =
  msg.getStringProperty("OC4J_REPLY_TO_FACTORY_NAME");
  System.out.println("queue connection factory name is ");
  System.out.println(qcfName);

// Lookup for the connection factory from JNDI and obtain the
// QueueConnectionFactory object. Start the connection, obtain a session if the
// session doesn't already exist.

  QueueConnectionFactory qcf =
  (QueueConnectionFactory) ctx.lookup(qcfName);
  QueueConnection conn = qcf.createQueueConnection();
  conn.start();
  QueueSession session = conn.createQueueSession
    (
      true,
      Session.AUTO_ACKNOWLEDGE
    );

//Obtain a QueueSender object and send the message to the queue.
  QueueSender sender = session.createSender(queue);
  sender.send(msg);

  session.commit();
  sender.close();
  session.close();
  conn.stop();
  conn.close();
}
```

```
catch (Exception ex)
 {
  ex.printStackTrace();
 }
```

7. Edit the jms.xml file in <*jdev_home*>\j2ee\home\config
```
<queue name="First Queue" location="jms/firstQueue">
     <description>First queue</description>
</queue>
<queue name="Second Queue" location="jms/secondQueue">
     <description>Second queue</description>
</queue>
<queue-connection-factory
location="jms/QueueConnectionFactory" username="admin"
password="welcome" />
```
8. Start OS command prompt and issue the command `echo %PATH%` to check if the `PATH` variable includes <*jdev_home*>\jdk\bin. Change directory to <*jdev_home*>\j2ee\home. Start the OC4J server with the following command
```
 java -jar oc4j.jar
```
9. Associate the Queue with the MDB. Edit the orion-ejb-jar.xml file, look for the message-driven-deployment tag and modify it to include the connection factory information.
   ```
   <message-driven-deployment name="jmsmdb" max-instances="-1" min-
   instances="0" connection-factory-location="jms/QueueConnectionFactory"
   destination-location="jms/firstQueue" />
   ```
10. Create the deployment profile for the MDB created. Right click on the ejb-jar.xml file in the System-Navigator and select Create EJB JAR Deployment Profile. Name the deployment profile as jmsmdb.deploy. Click the OK button. Once again click the OK button.

Follow these steps to create the JMS Web service.

1. Right click on the Project and select New. From the Business Tier node of Categories select Web Services and choose JMS Web Service from Items.
2. Click the OK button. You will see the wizard. Click the Next button.
   Enter MyJMSWebService for Web Service name. Click the Next button.

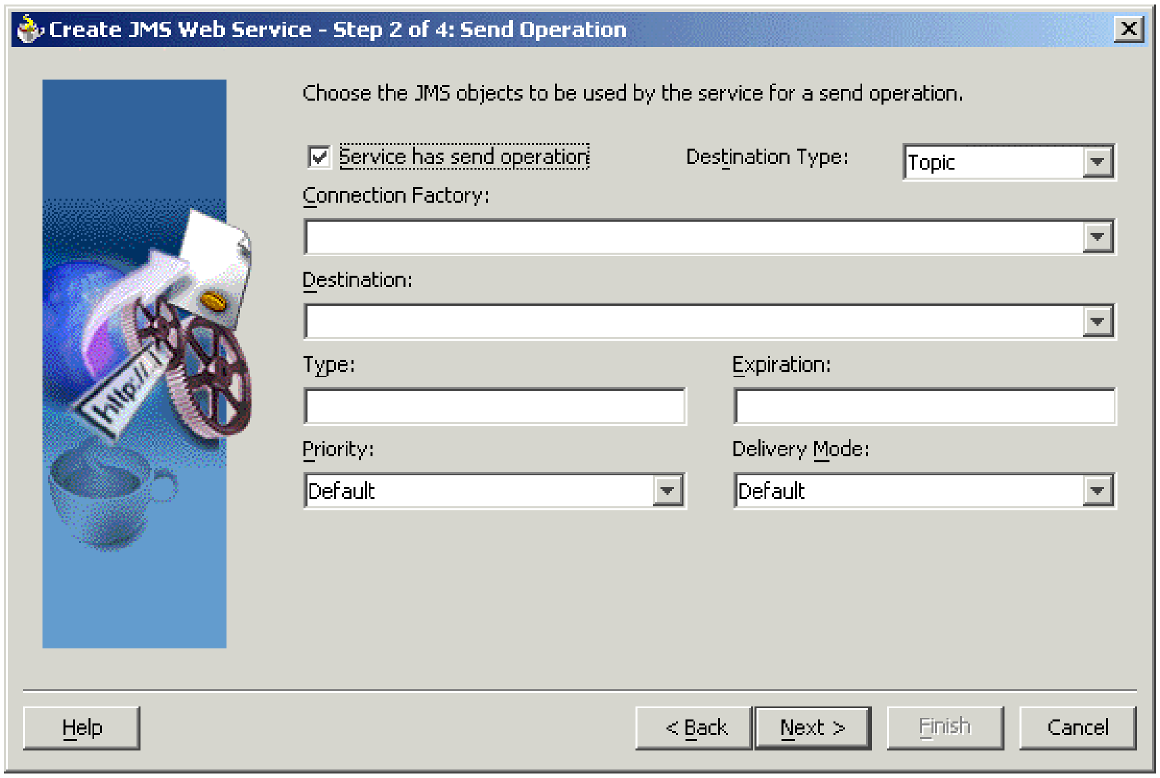

Create JMS Web Service - Step 1 of 4: Name

Enter a unique name for the web service.

Web Service Name: MyJMSWebService

Package: mypackage1

3. You will see the following screen.


Create JMS Web Service - Step 2 of 4: Send Operation

Choose the JMS objects to be used by the service for a send operation.

☑ Service has send operation

Destination Type: Topic

Connection Factory:

Destination:

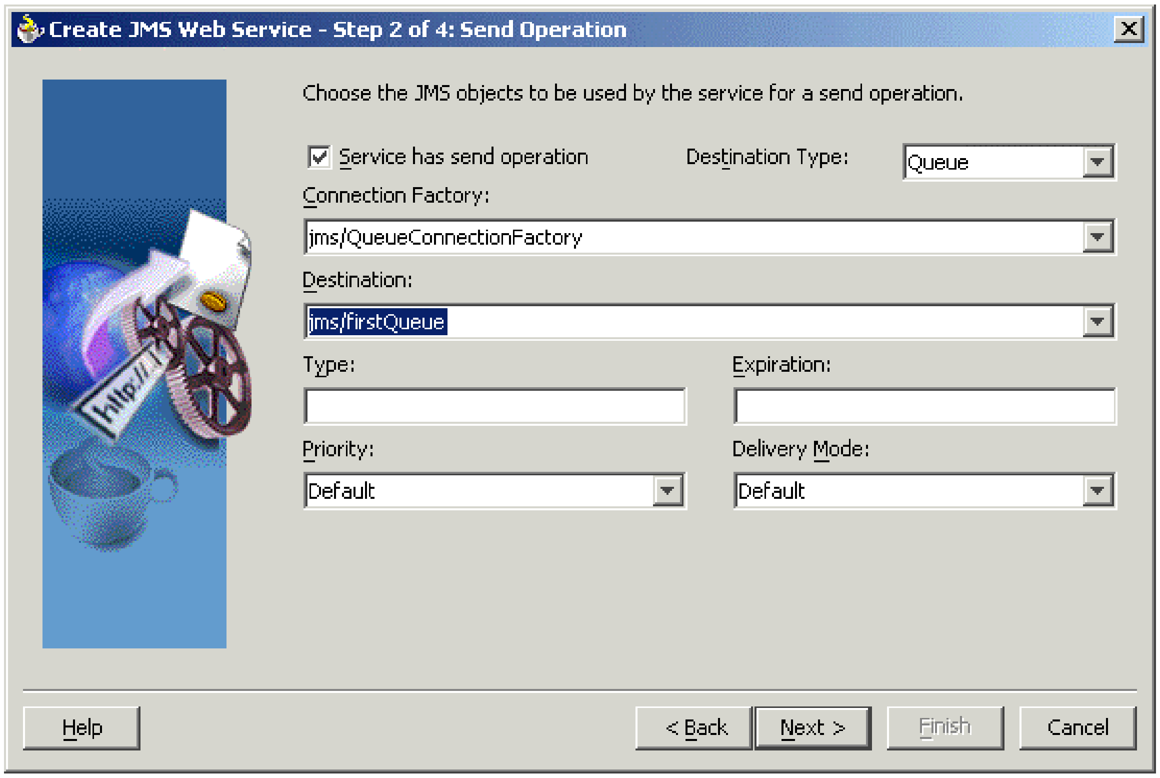
Type:

Expiration:

Priority: Default

Delivery Mode: Default

Provide the JMS Connection Factory and destination information as shown below


Create JMS Web Service - Step 2 of 4: Send Operation

Choose the JMS objects to be used by the service for a send operation.

☑ Service has send operation

Destination Type: Queue

Connection Factory: jms/QueueConnectionFactory

Destination: jms/firstQueue

Type:

Expiration:

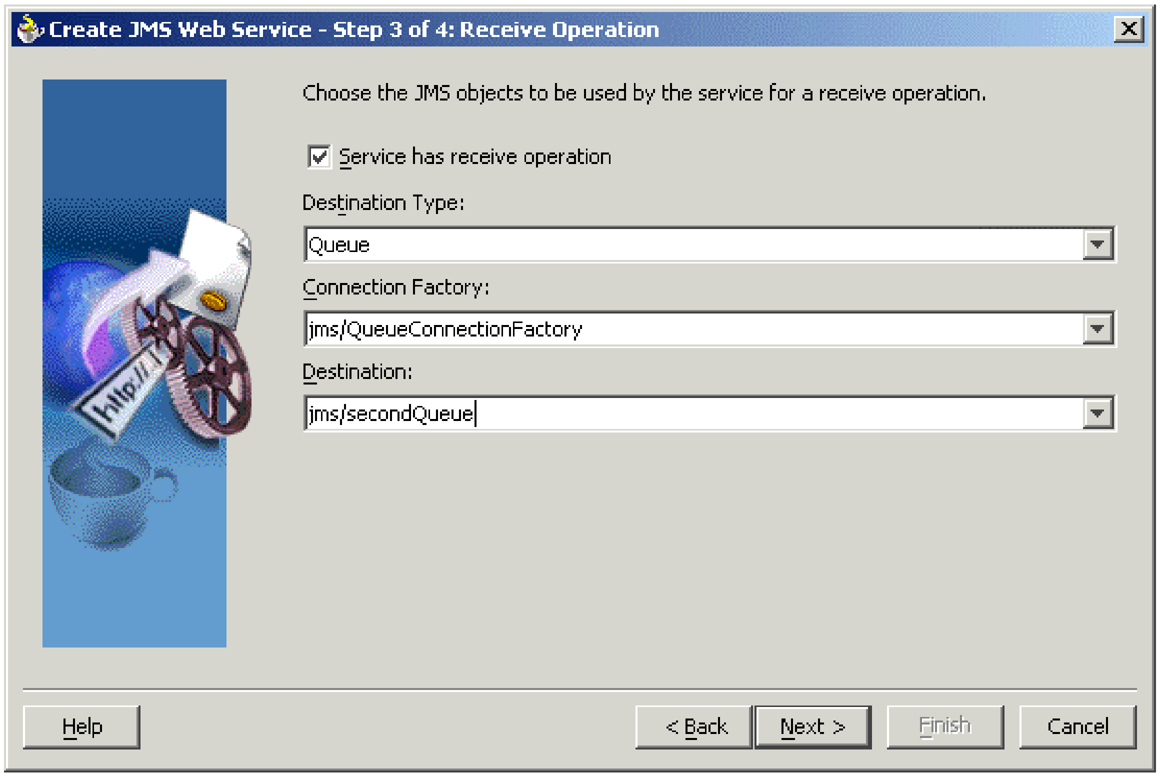
Priority: Default

Delivery Mode: Default

Type is a header property used to identify the kind of message. It is just an identifier for the message.
Expiration is the lifetime of the sent message in milliseconds.
Priority by default is 4. You can select a priority from 1 to 9.
Delivery Mode by default is Persistent.

Click the 'Next' button.

Provide information about the destination queue.
Note that the fields will be disabled until you check the 'Service has receive operation' checkbox. Select Queue as destination type. Select the connection factory. Edit the Destination field to specify jms/secondQueue. Click the Next button.


Create JMS Web Service - Step 3 of 4: Receive Operation

Choose the JMS objects to be used by the service for a receive operation.

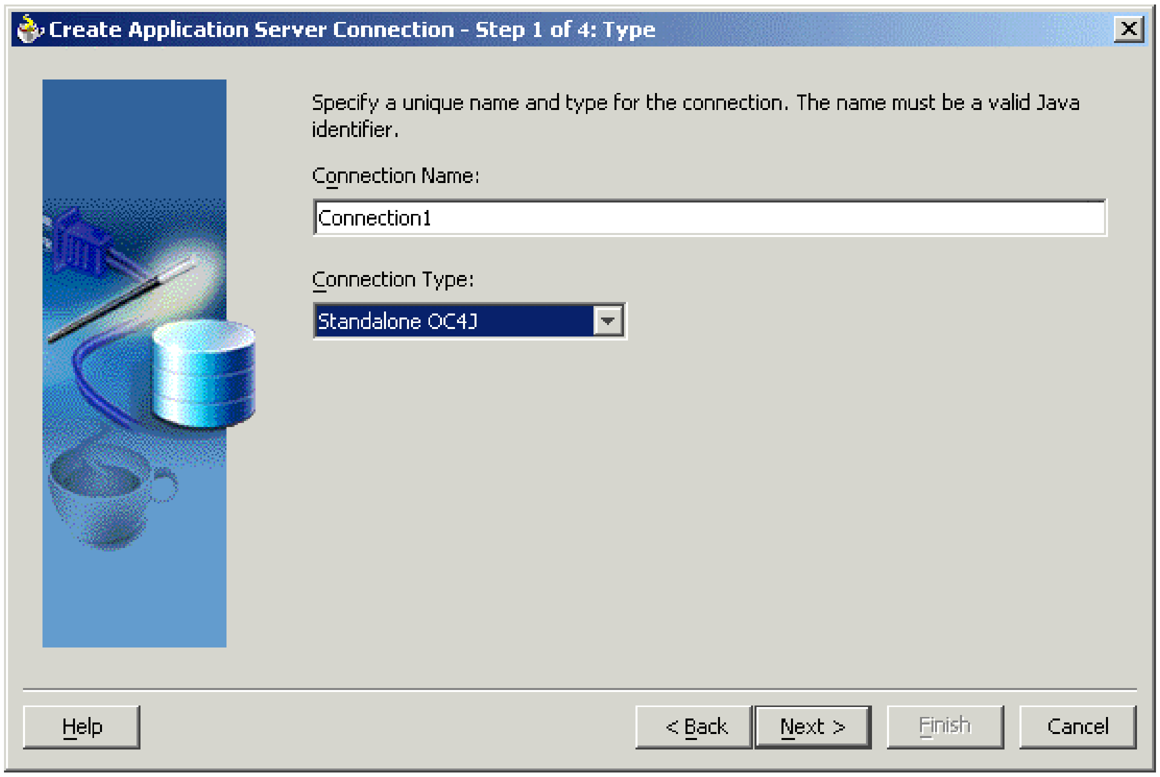
☑ Service has receive operation

Destination Type: Queue

Connection Factory: jms/QueueConnectionFactory

Destination: jms/secondQueue

4. Create a New connection to the OC4J, which you started in step 7 of MDB creation. Call this connection Connection1.

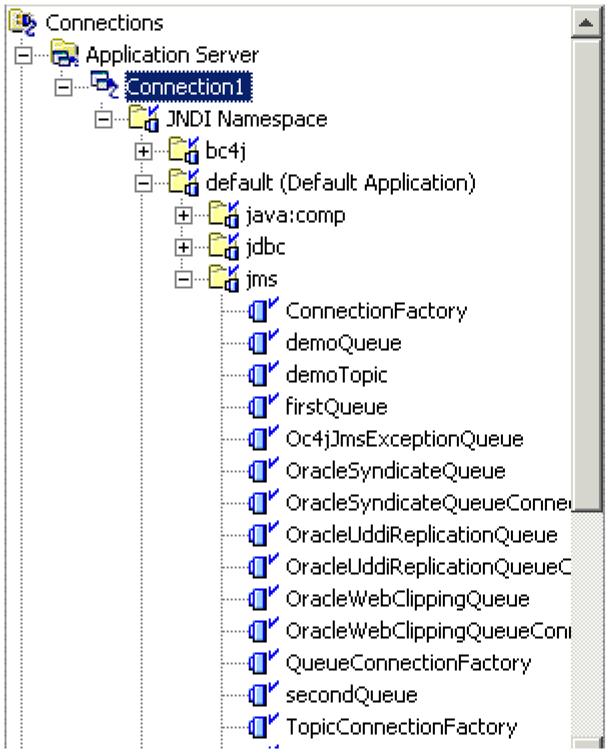


Specify a unique name and type for the connection. The name must be a valid Java identifier.

Connection Name: Connection1

Connection Type: Standalone OC4J

5. Click the Finish button and observe that the generated Web service and the associated files are shown in the System-Navigator.
   Expand connection 1 to check out the topics and queues in JMS as shown



Follow these steps to generate the stub file and test the JMS Web service.

1. In the system-navigator right click on the MyJMSWebService and select Generate Web Service Stub….
2. You will see the wizard. Click the Next button , check the Generate Main Method Into Stub checkbox and click the Next button and click the Finish button.
3. Edit the stub file to import oracle.xml.parser.v2 package and add the following lines of code to the main method

```
System.out.println("sending an XML Element");
XMLDocument doc = new XMLDocument();
Element main = doc.createElement("Message");
Element sub = doc.createElement("content");
sub.appendChild(doc.createTextNode("SENDING A
DOCUMENT"));
main.appendChild(sub);
stub.send(main);
System.out.println("sent\n\n");
System.out.println("Receiving the XML Element");
Vector res = stub.receive();
XMLElement el = (XMLElement) res.elementAt(0);
el.print(System.out);
```

Follow these steps to deploy and run the service.

1. Right click on the jmsmdb.deploy file and select Deploy to Connection1. Check the messages displayed in the log window to ensure that the deployment was successful.
2. Right click on WebServices.deploy file and select Deploy to Connection1.
3. Right click on the MyJMSWebServiceStub.java file and select Run.

You should see the following output in the log window.

```
sending an XML Element
sent
Receiving the XML Element
<Message>
   <content>SENDING A DOCUMENT</content>
</Message>
```

Also observe the messages displayed in the OS prompt from where you started the OC4J server.

NOTE: The complete workspace for the demo jmsws is provided in the demos/lesson19 folder. If you are using the workspace that is already provided, then check the host and port details for the string _endpoint in `MyJMSWebServiceStub.java`.