Write Java Message Service programs using WebSphere MQ Version 6 and Rational Application Developer Version 6, Part 3: Using JMS and WebSphere MQ for the test environment

Willy Farrell

Use Java Message Service (JMS) 1.1 unified domain programming interfaces for point-to-point and publish/subscribe applications. And configure and use IBM® WebSphere® MQ as the Java Message Service provider for the IBM Rational® Application Developer WebSphere Test Environment.

View more content in this series

Before you start
Learn what to expect from this tutorial, and how to get the most out of it.

About this tutorial
This tutorial provides an overview of the unified domain interfaces which are the preferred method of client programming for Java Message Service (JMS) 1.1. Sample applications along with instructions for running those applications are included to demonstrate and test the capabilities and use of these interfaces. The tutorial also shows how to configure IBM WebSphere MQ as the JMS provider for the Rational Application Developer (Application Developer) WebSphere Test Environment, along with instructions for running applications with that configuration.

Objectives
After completing this tutorial, you will know how to:

- Use JMS unified domain interfaces
- Configure WebSphere MQ as the JMS provider for the Application Developer WebSphere Test Environment

Prerequisites
This tutorial is intended for Java programmers who need to understand how to write and test JMS programs using WebSphere MQ and Rational Application Developer. Intermediate knowledge of
Java and introductory knowledge of JMS is assumed. The instructions in this tutorial also assume that you have read and carried out the instructions in Part 1 and Part 2, since they build on the configurations and sample code created and tested in those tutorials.

System requirements

The instructions in this tutorial are for use in a Windows environment, though readers knowledgeable in other operating systems can probably adapt them for use in their operating system.

The system requirements for the products used in the tutorial can be found through the following links:

- WebSphere MQ
- Rational Application Developer

Introduction

In Part 1 of this series, you get a brief overview of enterprise messaging systems and JMS. It provides descriptions of the latest tools from IBM for JMS development and testing, and gives detailed instructions about obtaining, installing, and configuring those tools. Sample programs and instructions on running them are provided in order to exercise the tools and understand how they work.

In Part 2 of this series, you learn about WebSphere MQ client transport and remote queuing, and JMS publish/subscribe applications. It gives step-by-step instructions for configuring these capabilities, and provides sample programs to demonstrate their use.

This installment (Part 3) picks up where those tutorials left off, to further explore WebSphere MQ and JMS programming with Rational Application Developer. The instructions in this tutorial assume that you have read and carried out the instructions in the first two tutorials, since Part 3 builds on the configurations and sample code created and tested in those tutorials.

JMS 1.1 unified domain common interfaces

JMS 1.0 provides high-level interfaces that are parents to both point-to-point and publish/subscribe domain-specific interfaces. These interfaces contain only those functions common to both domains, and JMS providers did not provide implementations of these interfaces. In JMS 1.1, the high-level interfaces are now considered common interfaces and contain all of the functionality of both domains; JMS providers must provide implementations of these interfaces. Although the common interfaces are still parents to the domain-specific interfaces, they are now the preferred method for JMS client programming, and the domain-specific interfaces are provided only for backward compatibility. The common interfaces unified the JMS domains, providing for easier programming and more flexible applications.

The following table shows the common interfaces and the domain-specific interfaces inherited from each.
Table 1. Common interfaces with corresponding domain-specific interfaces

<table>
<thead>
<tr>
<th>JMS common interface</th>
<th>Point-to-point domain</th>
<th>Publish/subscribe domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFactory</td>
<td>QueueConnectionFactory</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>Connection</td>
<td>QueueConnection</td>
<td>TopicConnection</td>
</tr>
<tr>
<td>Destination</td>
<td>Queue</td>
<td>Topic</td>
</tr>
<tr>
<td>Session</td>
<td>QueueSession</td>
<td>TopicSession</td>
</tr>
<tr>
<td>MessageProducer</td>
<td>QueueSender</td>
<td>TopicPublisher</td>
</tr>
<tr>
<td>MessageConsumer</td>
<td>QueueReceiver, QueueBrowser</td>
<td>TopicSubscriber</td>
</tr>
</tbody>
</table>

Review the common interfaces sample code

To better illustrate unified domain programming, I've modified the sample code from the previous tutorials to use only the common interfaces. In this section, you will import this sample code and review portions of it to gain a better understanding of using the common interfaces.

All of the code has been packaged in application client modules in order to run it using the WebSphere Test Environment.

Install the sample code

1. Download the sample code file named i-mqrad3code.zip. (Click Downloadable resources for a link). On my system, I downloaded it to a folder named i-mqrad3code on my Windows desktop.
2. If not already started, launch Rational Application Developer from the Windows Start menu. (From this point on the instructions assume Application Developer is running.)
3. In Application Developer, from the main menu select File > Import.
4. Select Project Interchange as the import source and click Next.
5. In the From zip file field, enter the location of the i-mqrad3code.zip file.
6. Click Select All, and click Finish.

QSender class

The first class to look at is QSender in the EQSenderCommon project. Start by looking at the JMS instance variables as shown in Listing 1.

Listing 1. JMS instance variables

```java
public class QSender {
    private Connection connection;
    private Session session;
    private MessageProducer sender;
}
```

As you can see, I've completely replaced the variables of point-to-point interfaces types used in the QSender class in the EQSender project with variables of common interfaces types.
Now take a look at the `setConnection()` method, shown in **Listing 2**.

**Listing 2. setConnection() method**

```java
public void setConnection(String connectionName) throws JMSException,
NamingException, Throwable {
    try {
        close();
        ConnectionFactory factory = (ConnectionFactory) getInitContext()
                .lookup(connectionName);
        connection = factory.createConnection();
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    } catch (Throwable e) {
        setExceptionMessage(e);
        throw e;
    }
}
```

This code is equivalent to the `setConnection()` method in the `QSender` class in the MQP2P project, but only uses the common interfaces.

The last method to look at in the `QSender` class is `setQueue()`, shown in **Listing 3**.

**Listing 3. setQueue() method**

```java
public void setQueue(String queueName) throws JMSException,
NamingException, Throwable {
    try {
        if (connection != null)
            connection.stop();
        if (sender != null)
            sender.close();
        Destination queue = (Destination) getInitContext()
                .lookup(queueName);
        sender = session.createProducer(queue);
        sender.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        sender.setPriority(4);
        sender.setTimeToLive(0);
        connection.start();
    } catch (Throwable e) {
        setExceptionMessage(e);
        throw e;
    }
}
```

And, once again, only common interfaces types are used. Note, though, that I have kept the variable names the same because, semantically, this program is intended for use with point-to-point messaging. However, the use of the common interfaces provides greater flexibility at run time, as you will see when you run this code.

**QReceiver class**

By now you should be getting the point -- this new sample code only uses the unified domain common interfaces. But take a look at the `setQueue()` method in the `QReceiver` class in the EQRceiverCommon project (shown in **Listing 4**) just to emphasize the point.
Listing 4. QReceiver method

```java
public void setQueue(String queueName) throws JMSException, 
    NamingException, Throwable {
    try {
        if (connection != null) 
            connection.stop();
        if (receiver != null) 
            receiver.close();
        Destination queue = (Destination) getInitContext()
            .lookup(queueName);
        receiver = session.createConsumer(queue);
        receiver.setMessageListener(this);
        connection.start();
    } catch (Throwable e) {
        setExceptionMessage(e);
        throw e;
    }
}
```

Although the variable is named `queue`, it is actually of type `Destination`, the parent interface for `Queue`. Also, the `receiver` variable is a `MessageConsumer`, the parent interface for `QueueReceiver`.

ChatMonitor class

Take a look at one more method in the `ChatMonitor` class in the `EChatMonitorCommon` project, the `onMessage()` method, shown in Listing 5.

Listing 5. onMessage() method

```java
public void onMessage(Message message) {
    String msgText = null;
    try {
        //msgText = 
        //   (((Topic) message.getJMSDestination()).getTopicName()
        // + ": " 
        // + ((TextMessage) message).getText());
        msgText = ((TextMessage) message).getText();
    } catch (Throwable e) {
        setExceptionMessage(e);
        msgText = getExceptionMessage();
    } finally {
        setMessage(msgText);
    }
}
```

Notice the code that is commented out. In the version of this code from Part 2 of this series, I could cast the `Destination` received from the call to `message.getJMSDestination()` to a `Topic`, because the message is coming from a topic, since I wrote the code using the domain-specific interfaces for publish/subscribe. Now that I've changed the code to use the common interfaces, I can't make that assumption. If I decided that I definitely wanted to show the topic name when the message did come from a topic, I could attempt the cast and catch the `ClassCastException` when the message is from a queue, substituting some appropriate text for the topic name.

I won't go over any more code in this tutorial, but I encourage you to examine all of the new sample code to see the uses of the common interfaces and gain greater understanding of them.
Run the common interfaces sample code
In this section, you will run the point-to-point programs written with common interfaces to see that they work as before, and you will run the publish/subscribe programs written with common interfaces and verify that they behave as the publish/subscribe programs behaved in the previous tutorial. Then, you will run all of the programs together and see the real power and flexibility of using the unified domain common interfaces.

Run the point-to-point programs
Run EQReceiverCommon
1. In Application Developer, ensure you are in the J2EE perspective.
2. Select the Servers view, at the bottom of the window.
3. Right-click WebSphere Application Server v6.0 and select Start.
4. Wait for the Status of the server to change to Started.
5. Right-click EQReceiverCommon and select Run > Run.
6. Select WebSphere v6.0 Application Client in the Configurations list.
7. Click New.
8. Change the Name field to EQReceiverCommon.
9. Click Run.
10. Type jms/BeanQCF in the Connection field and press Enter or click Set.
11. Type jms/MyBeanQ in the Queue field and press Enter or click Set.

Run EQSenderCommon
1. Back in Application Developer, right-click EQSenderCommon and select Run > Run.
2. Select WebSphere v6.0 Application Client in the Configurations list.
3. Click New.
4. Change the Name field to EQSenderCommon.
5. Click Run.
6. Type jms/BeanQCF in the Connection field and press Enter or click Set.
7. Type jms/MyBeanQ in the Queue field and press Enter or click Set.

Test the programs
1. Make both the Queue Sender and Queue Receiver windows visible.
2. Type text into the Message field of Queue Sender and press Enter or click Set.
3. You should see the message text appear in the Queue Receiver window.
4. When you are finished, keep both windows open.

Run the publish/subscribe programs
Let’s try the publish/subscribe programs.

Run EChatCommon
1. Right-click EChatCommon and select Run > Run.
2. Select WebSphere v6.0 Application Client in the Configurations list.
3. Click New.
4. Type EChatCommon in the Name field.
5. Click Run.
6. Type user1 in the User Name field of the Chat window and press Enter or click Set.
7. Type \texttt{jms/BeanTCF} in the \textbf{Connection} field and press Enter or click Set.
8. Type \texttt{jms/Java} in the \textbf{Topic} field and press Enter or click Set.
9. You will see the message: "User1 has entered the chat room".
10. In Application Developer, from the main menu select \textbf{Run > Run History > EChatCommon}.
11. In this second Chat window, type \texttt{user2} in the \textbf{User Name} field and press Enter or click Set.
12. Type \texttt{jms/BeanTCF} in the \textbf{Connection} field and press Enter or click Set.
13. Type \texttt{jms/Java} in the \textbf{Topic} field and press Enter or click Set.
14. You will see the message "User2 has entered the chat room" in both Chat windows.
15. In the \textbf{Message} field of either window, type any text and press Enter or click Send. The message text will display in both windows, prepended with the name of the user that published the message.
16. Try changing the topic in each of the windows to one of the other topics created with JMSAdminGUI. Don't try jms/AllComputers or jms/AllBooks, however, because publishing is not allowed to wildcard (*) topics, only subscribing.
17. Also, try running additional instances of \texttt{EChatCommon} with different user names.
18. Change the topic on one of the Chat windows to Java before proceeding with the next steps.

\textbf{Run EChatMonitorCommon}

1. Return to Application Developer.
2. From the main menu, select \textbf{Run > Run}.
3. Right-click \texttt{EChatCommon} in the \textbf{Configurations} list and select \textbf{Duplicate}.
4. Rename \texttt{EChatCommon (1)} to \texttt{EChatMonitor}.
5. Change the \textbf{Application client module} field to \texttt{EChatMonitorCommon}.
6. Click \textbf{Run}.
7. Type \texttt{jms/BeanTCF} in the \textbf{Connection} field of the Chat Monitor window and press Enter or click Set.
8. Type \texttt{jms/AllComputers} in the \textbf{Topic} field and press Enter or click Set.
9. Switch to the Chat window with the jms/Java topic in use, ensuring that the Chat Monitor window is visible and send a message.
10. The message text will display in the Chat Monitor window, prepended with the user that published the message.
11. Switch to the Chat window with the jms/Java topic in use, and type \texttt{jms/Linux} in the \textbf{Topic} field, then press Enter or click Set.
12. Send a message from that Chat window.
13. The message text will display in the Chat Monitor window.
14. In the Chat window with the jms/Linux topic in use, type \texttt{jms/Cyberpunk} and press Enter or click Set.
15. Send a message from that Chat window.
16. You will \texttt{not} see the message in the Chat Monitor window.
17. When you are finished, close the Chat window using jms/Cyberpunk as its topic by clicking the \texttt{X} in the upper-right corner. Leave one Chat window and the Chat Monitor window open. Ensure the open Chat window is using jms/Java as its topic.

\textbf{Mix and match}

At this point, you are probably thinking "Ho hum, whoop dee doo, I get it. I can use the common interfaces to get the same behavior as the domain-specific interfaces." But wait, there's more.
Let's have some fun. In the previous parts of this series, the point-to-point programs could not communicate with the publish/subscribe programs, because they were written for specific domains. The new sample programs don't have that limitation, as you will discover shortly.

1. Switch to the Queue Receiver window, and type `jms/Java` in the **Queue** field, and press **Enter** or click **Set**.
2. Switch to the Queue Sender window, and type `jms/BeanTCF` in the **Connection** field, and press **Enter** or click **Set**.
3. Still in the Queue Sender window, type `jms/Java` in the **Queue** field, and press **Enter** or click **Set**.
4. Send a message from the Queue Sender window. All three of the other windows should receive that message.
5. Switch to the Chat window, and send a message. The Chat Monitor and the Queue Receiver should receive that message.
6. Still in the Chat window, type `jms/MyBeanQ` in the **Topic** field and press **Enter** or click **Set**.
7. Switch to the Queue Receiver window and type `jms/MyBeanQ` in the **Queue** window and press **Enter** or click **Set**.
8. Switch to the Chat window and send a message. The Queue Receiver should receive the message.
9. When you are finished, close all windows by clicking the **X** in the upper-right corner, and stop the server.

You were probably surprised when the instructions told you to type the name of a TopicConnectionFactory in a point-to-point program or the name of a queue in a publish/subscribe program. But that's the point. AConnectionFactory is a ConnectionFactory, whether it's a QueueConnectionFactory or a TopicConnectionFactory. And a Destination is a Destination, whether it's a Topic or a Queue.

From that demonstration, you should now begin to really understand the power, flexibility, and versatility of the unified domain common interfaces.

**WebSphere MQ as the WebSphere Test Environment JMS provider**

Up to this point, you have used WebSphere MQ directly for JMS programming, with the help of the SupportPacs that provide Java Naming and Directory Interface (JNDI) capabilities within WebSphere MQ, and you have used the built-in JMS messaging capability of the WebSphere Test Environment. In this section, you will configure WebSphere MQ to be the JMS provider for the WebSphere Test Environment. Since the WebSphere Test Environment is, in fact, WebSphere Application Server V6, these instructions apply to using WebSphere MQ as the JMS provider for WebSphere Application Server V6.

**WebSphere MQ Fix Pack 6.0.1.1**

When I was developing and testing this portion of the tutorial, I ran into several errors that, quite frankly, had me befuddled. I did some research and realized that I wasn't making mistakes, but that there was a known problem with what I was trying to do. Fortunately, that problem has been
corrected with WebSphere MQ Fix Pack 6.0.1.1. Before proceeding, you must install this fix pack on your system. (See Downloadable resources for a link.) Follow the instructions that accompany the fix pack for installing it. You do not have to install the updated Java SDK that is included with the fix pack.

As soon as the fix pack is installed, you may have to reboot your system. If so, restart Application Developer. Whether you reboot or not, restart the broker with the `strmqbrk -m End` command, as described in Part 2 of this tutorial.

**Configuring WebSphere MQ**

**Changing the WebSphere MQ install variable**

As installed, the WebSphere Test Environment uses WebSphere MQ and JMS libraries provided with Application Developer. For the test environment to use WebSphere MQ properly, it must use the libraries provided by WebSphere MQ. This is a simple matter of changing a variable in the test environment.

1. In Application Developer, ensure you are in the J2EE perspective.
2. Select the Servers view.
3. Right-click **WebSphere Application Server v6.0** and select **Start**.
4. Wait for the status of the server to change to Started.
5. Right-click the server and select **Run administrative console**.
6. When the console appears, log in with any user ID.
7. Click **Environment** from the navigation bar on the left.
8. Click **WebSphere Variables**.
9. Click the **MQ_INSTALL_ROOT** link in the list of variables.
10. Change the **Value** field to `C:\WSMQ` (see Figure 1).

![Figure 1. Changing MQ_INSTALL_ROOT variable](image)

---

Write Java Message Service programs using WebSphere MQ
Version 6 and Rational Application Developer Version 6, Part 3:
Using JMS and WebSphere MQ for the test environment
11. Click **OK**.
12. Click **Save** at the top of the page, then click the **Save** button.

### Create JMS administered objects

1. On the navigation bar to the left, click **Resources > JMS Providers > WebSphere MQ**.
2. Click **WebSphere MQ connection factories** in the **Additional Properties** list.
3. Click **New**.
4. Type **WigginsCF** in the **Name** field.
5. Type **.jms/WigginsCF** in the **JNDI name** field.
6. Type **Ender** in the **Queue manager** field (See **Figure 2**).

![Figure 2. Configuring WebSphere MQ connection factory](image)

7. These next two changes are required when using the built-in message broker in WebSphere MQ. Different values would be used when using a separate message broker, such as WebSphere Event Broker.
   8. Change **Broker version** to **Basic**.
   9. Change **Broker message selection** to **Client**. (See **Figure 3**).
Figure 3. Configuring WebSphere MQ connection factory (continued)

10. Click **OK**.
11. Click **WebSphere MQ** on the navigation bar to the left.
12. Click **WebSphere MQ queue destinations** in the **Additional Properties** list.
13. Click **New**.
14. Type **WigginsQ** in the **Name** field.
15. Type **jms/MyWigginsQ** in the **JNDI name** field.
16. Type **EnderQ** in the **Base queue name** field. (See Figure 4).
17. Click **OK**.
18. Click **New** again.
19. Type `DemosthenesQ` in the **Name** field.
20. Type `jms/MyDemosthenesQ` in the **JNDI name** field.
21. Type `RemotePetraQ` in the **Base queue name** field. (See Figure 4.)
22. Click **OK**.
23. Click **WebSphere MQ** on the navigation bar to the left.
24. Click **WebSphere MQ topic destinations** in the **Additional Properties** list.
25. Click **New**.
26. Type `Locke` in the **Name** field.
27. Type `jms/Locke` in the **JNDI name** field.
28. Type `computers/java` in the **Base topic name** field.
29. Click **OK**.
30. Click **Save** at the top of the page, then click the **Save** button.
31. Exit the admin console.
32. Right-click **WebSphere Application Server v6.0** in the **Servers** view and select **Restart > Start**.

**Run the sample code**

Now you can run the sample programs to test the configuration. Just as you had to change the WebSphere Test Environment to use the WebSphere MQ libraries, you also have to change the application client modules to use those same libraries.
Run the publish/subscribe sample programs

1. In Application Developer, from the main menu select **Run > Run**.
2. Select **EChatCommon** in the **Configurations** list.
3. Click the **Arguments** tab.
4. Change the entry in the `-Dws.ext.dirs` argument from `C:\RAD\runtimes\base_v6\lib\WMQ\java\lib` to `C:\WSMQ\Java\lib`. (See Figure 5.)

   **Figure 5. Configuring EChatCommon VM arguments**

   ![Figure 5. Configuring EChatCommon VM arguments](image)

5. Click **Run**.
6. Type **User1** in the **User Name** field of the Chat window and press **Enter** or click **Set**.
7. Type **jms/WigginsCF** in the **Connection** field and press **Enter** or click **Set**.
8. Type **jms/Locke** in the **Topic** field and press **Enter** or click **Set**.
9. You will see the message "User1 has entered the chat room".
10. In Application Developer, from the main menu select **Run > Run History > EChatCommon**.
11. In this second Chat window, type **user2** in the **User Name** field and press **Enter** or click **Set**.
12. Type **jms/WigginsCF** in the **Connection** field and press **Enter** or click **Set**.
13. Type **jms/MyWigginsQ** in the **Queue** field and press **Enter** or click **Set**.
14. You will see the message "User2 has entered the chat room" in both Chat windows.
15. In the **Message** field of either window, type any text and press **Enter** or click **Send**. The message text will display in both windows, prepended with the name of the user that published the message.
16. When you are finished, close all windows by clicking the **X** in the upper-right corner.

Run the point-to-point sample code

1. From the main menu select **Run > Run**.
2. Select **EQReceiverCommon** in the **Configurations** list.
3. Click the **Arguments** tab.
4. Change the entry in the `-Dws.ext.dirs` argument from `C:\RAD\runtimes\base_v6\lib\WMQ\java\lib` to `C:\WSMQ\Java\lib`.
5. Click **Run**.
6. Type **jms/WigginsCF** in the **Connection** field and press **Enter** or click **Set**.
7. Type **jms/MyWigginsQ** in the **Queue** field and press **Enter** or click **Set**.
8. From the main menu select Run > Run.
9. Select EQSenderCommon in the Configurations list.
10. Click the Arguments tab.
11. Change the entry in the -Dws.ext.dirs argument from C:\RAD\runtimes\base_v6\lib\WMQ\java\lib to C:\WSMQ\Java\lib.
12. Click Run.
13. Type jms/WigginsCF in the Connection field and press Enter or click Set.
14. Type jms/MyWigginsQ in the Queue field and press Enter or click Set.
15. Make both the Queue Sender and Queue Receiver windows visible.
16. Type text into the Message field of Queue Sender and press Enter or click Set.
17. You should see the message text appear in the Queue Receiver window.
18. When you are finished, close both windows.

Mix and match again

And, finally, we'll run programs using different JNDI environments and remote queuing, just to show off a little of the power of WebSphere MQ.

1. Start WebSphere MQ Explorer from the Windows Start menu.
2. Select Channels under the Ender queue manager.
3. Right-click Ender_2_Petra and select Start.
4. In Application Developer, from the main menu, select Run > Run.
5. Select QReceiverUI and click Run.
6. Type Petra in the Connection field and press Enter or click Set.
7. Type PetraQ in the Queue field and press Enter or click Set.
8. From the main menu, select Run > Run History > EQSenderCommon.
9. Type jms/WigginsCF in the Connection field and press Enter or click Set.
10. Type jms/MyDemosthenesQ in the Queue field and press Enter or click Set.
11. Make both the Queue Sender and Queue Receiver windows visible.
12. Type text into the Message field of Queue Sender and press Enter or click Set.
13. You should see the message text appear in the Queue Receiver window.
14. When you are finished, close both windows.

Summary

In this tutorial, you picked up where Part 2 left off and explored some additional topics in JMS programming: JMS 1.1 unified domain programming interfaces and using WebSphere MQ as the JMS provider for the Application Developer WebSphere Test Environment.

While we haven't covered everything there is to know about JMS programming with IBM tools, you should now have a good foundation for using IBM products for enterprise messaging and JMS programming, and can begin using these technologies in your own applications.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample programs</td>
<td>i-mqrad3code.zip</td>
<td>3KB</td>
</tr>
</tbody>
</table>
Related topics

- Download a free trial version of WebSphere MQ Version 6.0.
- Download a free trial version of Rational Application Developer Version 6.0.
- Download WebSphere MQ V6.0 Fix Pack 6.0.1.1. This fix pack is required to run the sample code in this tutorial.
- Build your next development project with IBM trial software, available for download directly from developerWorks.
- In "Introducing the Java Message Service" (developerWorks, June 2004), learn the basics for writing JMS applications. This tutorial covers the common interfaces for unified domain programming introduced in JMS 1.1.
- Learn more about Java Message Service at Sun’s site.
- In "Using WebSphere MQ with WebSphere Application Server Community Edition" (developerWorks, Jan 2006), learn how to configure WebSphere Application Server Community Edition to use WebSphere MQ as its JMS provider.

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)