JMS Step 1 - How to Create a Simple JMS Queue in Weblogic Server 11g

***By John-Brown.Evans on*** [***Nov 07, 2012***](https://blogs.oracle.com/soaproactive/entry/how_to_create_a_simple)

This example shows the steps to create a simple JMS queue in WebLogic Server 11g for testing purposes. For example, to use with the two sample programs QueueSend.java and QueueReceive.java which will be shown in later examples.

Additional, detailed information on JMS can be found in the following Oracle documentation:

*Oracle® Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server*   
*11g Release 1 (10.3.6)*   
*Part Number E13738-06*   
<http://docs.oracle.com/cd/E23943_01/web.1111/e13738/toc.htm>

1. Introduction and Definitions

A JMS queue in Weblogic Server is associated with a number of additional resources:

JMS Server

A JMS server acts as a management container for resources within JMS modules. Some of its responsibilities include the maintenance of persistence and state of messages and subscribers. A JMS server is required in order to create a JMS module.

JMS Module

A JMS module is a definition which contains JMS resources such as queues and topics. A JMS module is required in order to create a JMS queue.

Subdeployment

JMS modules are targeted to one or more WLS instances or a cluster. Resources within a JMS module, such as queues and topics are also targeted to a JMS server or WLS server instances. A subdeployment is a grouping of targets. It is also known as advanced targeting.

Connection Factory

A connection factory is a resource that enables JMS clients to create connections to JMS destinations.

JMS Queue

A JMS queue (as opposed to a JMS topic) is a point-to-point destination type. A message is written to a specific queue or received from a specific queue.

The objects used in this example are:

|  |  |  |
| --- | --- | --- |
| **Object Name** | **Type** | **JNDI Name** |
| TestJMSServer | JMS Server |  |
| TestJMSModule | JMS Module |  |
| TestSubDeployment | Subdeployment |  |
| TestConnectionFactory | Connection Factory | jms/TestConnectionFactory |
| TestJMSQueue | JMS Queue | jms/TestJMSQueue |

2. Configuration Steps

The following steps are done in the WebLogic Server Console, beginning with the left-hand navigation menu.

2.1 Create a JMS Server

* Services > Messaging > JMS Servers  
    
  
* Select **New**
* Name: **TestJMSServer**   
  Persistent Store: (none)
* Target: **soa\_server1**  (or choose an available server)
* **Finish**

The JMS server should now be visible in the list with Health OK.



2.2 Create a JMS Module

* Services > Messaging > JMS Modules
* Select **New**
* Name: **TestJMSModule**   
  Leave the other options empty
* Targets: **soa\_server1**  (or choose the same one as the JMS server)  
  Press **Next**
* Leave “Would you like to add resources to this JMS system module” unchecked and  press **Finish** .

2.3 Create a SubDeployment

A subdeployment is not necessary for the JMS queue to work, but it allows you to easily target subcomponents of the JMS module to a single target or group of targets. We will use the subdeployment in this example to target the following connection factory and JMS queue to the JMS server we created earlier.

* Services > Messaging > JMS Modules
* Select **TestJMSModule**
* Select the **Subdeployments**  tab and **New**
* Subdeployment Name: **TestSubdeployment**
* Press **Next**
* Here you can select the target(s) for the subdeployment. You can choose either Servers (i.e. WebLogic managed servers, such as the soa\_server1) or JMS Servers such as the JMS Server created earlier. As the purpose of our subdeployment in this example is to target a specific JMS server, we will choose the JMS Server option.   
  Select the **TestJMSServer** created earlier
* Press **Finish**

2.4  Create a Connection Factory

* Services > Messaging > JMS Modules
* Select **TestJMSModule**  and press **New**
* Select **Connection Factory**  and **Next**
* Name: **TestConnectionFactory**   
  JNDI Name: **jms/TestConnectionFactory**   
  Leave the other values at default
* On the Targets page, select the **Advanced Targeting**  button and select **TestSubdeployment**
* Press Finish

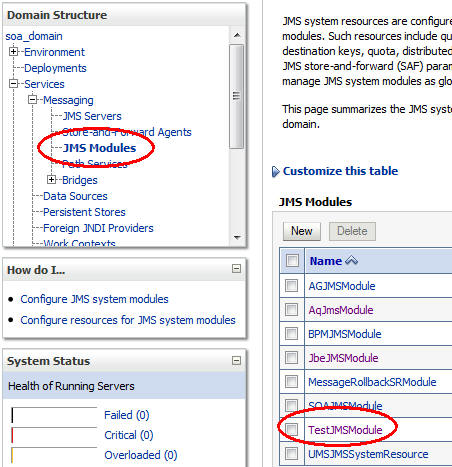
The connection factory should be listed on the following page with TestSubdeployment and TestJMSServer as the target.

2.5 Create a JMS Queue

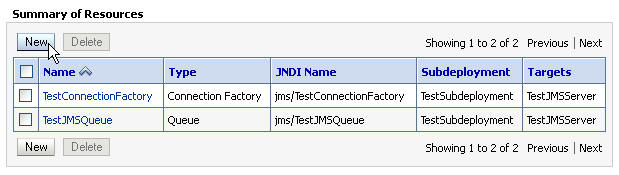
* Services > Messaging > JMS Modules
* Select **TestJMSModule**  and press **New**
* Select **Queue** and **Next**
* Name: TestJMSQueue  
  JNDI Name: jms/TestJMSQueue  
  Template: None  
  Press **Next**
* Subdeployments: **TestSubdeployment**
* **Finish**

The TestJMSQueue should be listed on the following page with TestSubdeployment and TestJMSServer.

Confirm the resources for the TestJMSModule. Using the Domain Structure tree, navigate to soa\_domain > Services > Messaging > JMS Modules then select TestJMSModule



You should see the following resources



The JMS queue is now complete and can be accessed using the JNDI names

jms/TestConnectionFactory and  
jms/TestJMSQueue.

In the following blog post in this series, I will show you how to write a message to this queue, using the WebLogic sample Java program QueueSend.java.

### JMS Step 2 - Using the QueueSend.java Sample Program to Send a Message to a JMS Queue

#### By John-Brown.Evans on [Nov 14, 2012](https://blogs.oracle.com/soaproactive/entry/jms_step_2_using_the)

This post is the second in a series of JMS articles which demonstrate how to use JMS queues in a SOA context.

In the previous post   
[JMS Step 1 - How to Create a Simple JMS Queue in Weblogic Server 11g](https://blogs.oracle.com/soaproactive/entry/how_to_create_a_simple)   
I showed you how to create a JMS queue and its dependent objects in WebLogic Server. In this article, we will use a sample program to write a message to that queue. Please review the previous post if you have not created those objects yet, as they will be required later in this example. The previous post also includes useful background information and links to the Oracle documentation for addional research.

The following post in this series will show how to read the message from the queue again.

## 1. Source code

The following java code will be used to write a message to the JMS queue. It is based on a sample program provided with the WebLogic Server installation. The sample is not installed by default, but needs to be installed manually using the WebLogic Server Custom Installation option, together with many, other useful samples. You can either copy-paste the following code into your editor, or install all the samples.

The knowledge base article in My Oracle Support:

[How To Install WebLogic Server and JMS Samples in WLS 10.3.x (Doc ID 1499719.1)](https://support.oracle.com/epmos/faces/ui/km/SearchDocDisplay.jspx?_afrLoop=7121038701894&type=DOCUMENT&id=1499719.1)

describes how to install the samples.

### QueueSend.java

|  |
| --- |
| package examples.jms.queue;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Hashtable;  import javax.jms.\*;  import javax.naming.Context;  import javax.naming.InitialContext;  import javax.naming.NamingException;  /\*\* This example shows how to establish a connection  \* and send messages to the JMS queue. The classes in this  \* package operate on the same JMS queue. Run the classes together to  \* witness messages being sent and received, and to browse the queue  \* for messages. The class is used to send messages to the queue.  \*  \* @author Copyright (c) 1999-2005 by BEA Systems, Inc. All Rights Reserved.  \*/  public class QueueSend  {  // Defines the JNDI context factory.  public final static String JNDI\_FACTORY="weblogic.jndi.WLInitialContextFactory";  // Defines the JMS context factory.  public final static String JMS\_FACTORY="jms/TestConnectionFactory";  // Defines the queue.  public final static String QUEUE="jms/TestJMSQueue";  private QueueConnectionFactory qconFactory;  private QueueConnection qcon;  private QueueSession qsession;  private QueueSender qsender;  private Queue queue;  private TextMessage msg;  /\*\*  \* Creates all the necessary objects for sending  \* messages to a JMS queue.  \*  \* @param ctx JNDI initial context  \* @param queueName name of queue  \* @exception NamingException if operation cannot be performed  \* @exception JMSException if JMS fails to initialize due to internal error  \*/  public void init(Context ctx, String queueName)  throws NamingException, JMSException  {  qconFactory = (QueueConnectionFactory) ctx.lookup(JMS\_FACTORY);  qcon = qconFactory.createQueueConnection();  qsession = qcon.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);  queue = (Queue) ctx.lookup(queueName);  qsender = qsession.createSender(queue);  msg = qsession.createTextMessage();  qcon.start();  }  /\*\*  \* Sends a message to a JMS queue.  \*  \* @param message message to be sent  \* @exception JMSException if JMS fails to send message due to internal error  \*/  public void send(String message) throws JMSException {  msg.setText(message);  qsender.send(msg);  }  /\*\*  \* Closes JMS objects.  \* @exception JMSException if JMS fails to close objects due to internal error  \*/  public void close() throws JMSException {  qsender.close();  qsession.close();  qcon.close();  }  /\*\* main() method.  \*  \* @param args WebLogic Server URL  \* @exception Exception if operation fails  \*/  public static void main(String[] args) throws Exception {  if (args.length != 1) {  System.out.println("Usage: java examples.jms.queue.QueueSend WebLogicURL");  return;  }  InitialContext ic = getInitialContext(args[0]);  QueueSend qs = new QueueSend();  qs.init(ic, QUEUE);  readAndSend(qs);  qs.close();  }  private static void readAndSend(QueueSend qs)  throws IOException, JMSException  {  BufferedReader msgStream = new BufferedReader(new InputStreamReader(System.in));  String line=null;  boolean quitNow = false;  do {  System.out.print("Enter message (\"quit\" to quit): \n");  line = msgStream.readLine();  if (line != null && line.trim().length() != 0) {  qs.send(line);  System.out.println("JMS Message Sent: "+line+"\n");  quitNow = line.equalsIgnoreCase("quit");  }  } while (! quitNow);  }  private static InitialContext getInitialContext(String url)  throws NamingException  {  Hashtable env = new Hashtable();  env.put(Context.INITIAL\_CONTEXT\_FACTORY, JNDI\_FACTORY);  env.put(Context.PROVIDER\_URL, url);  return new InitialContext(env);  }  } |

## 2. How to Use This Class

### 2.1 From the file system on UNIX/Linux

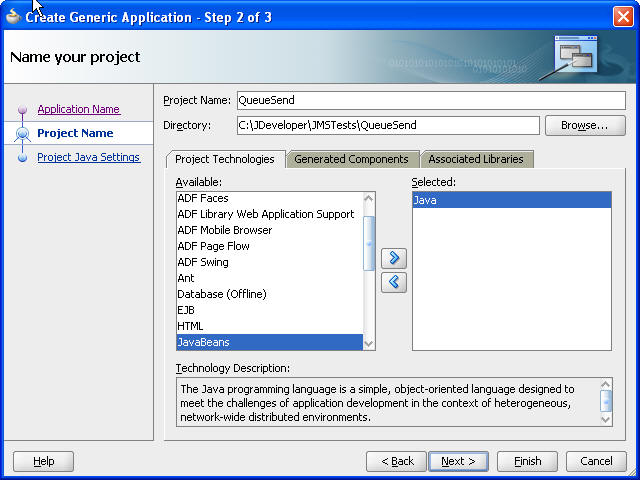
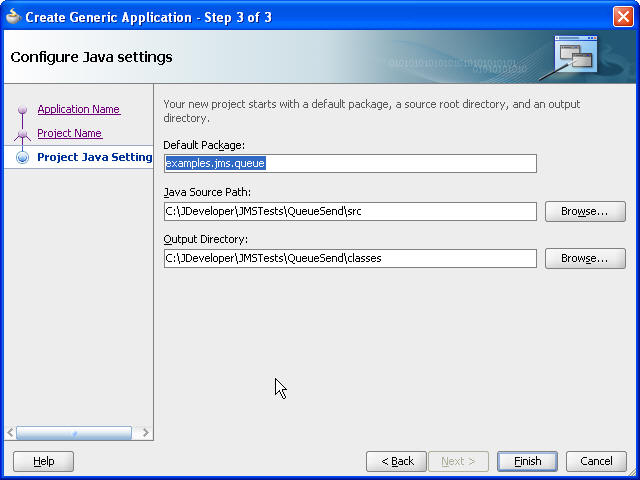
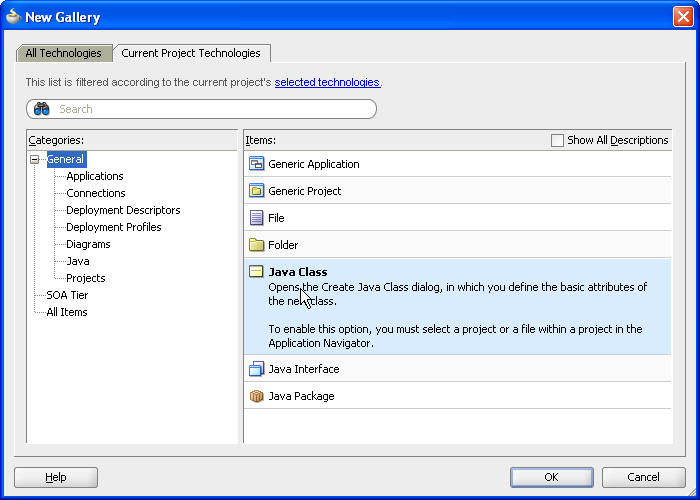
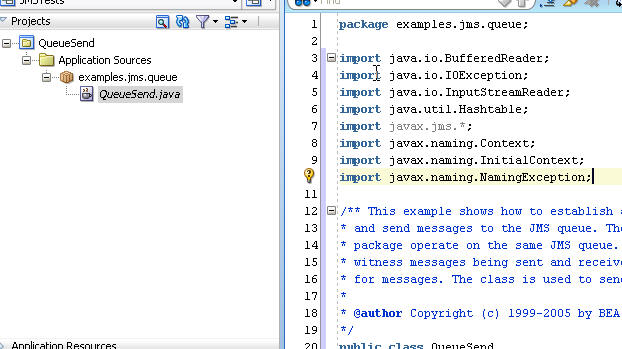
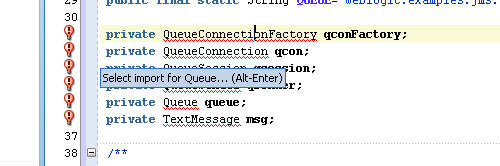
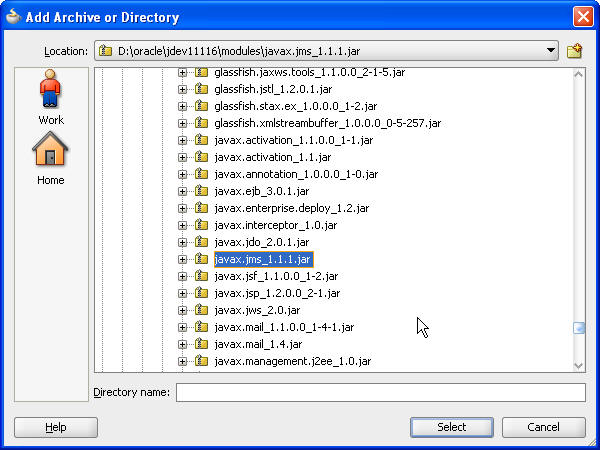
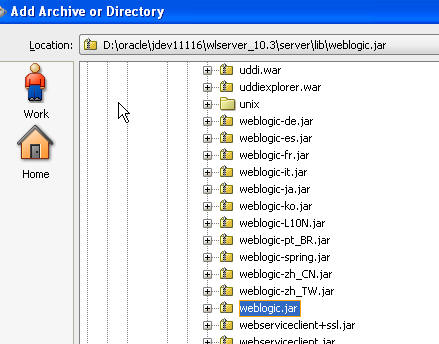
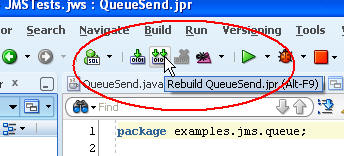
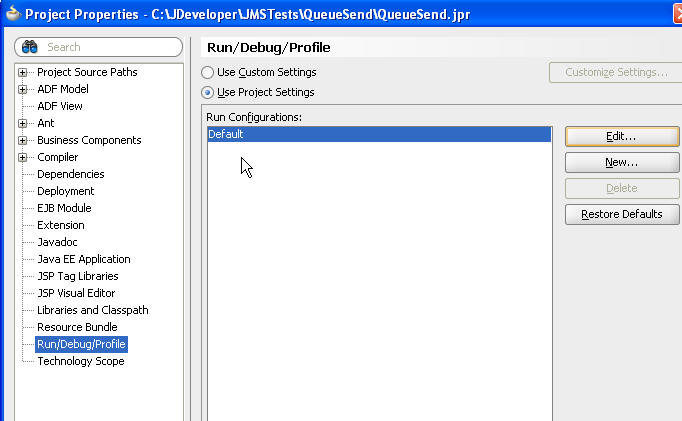
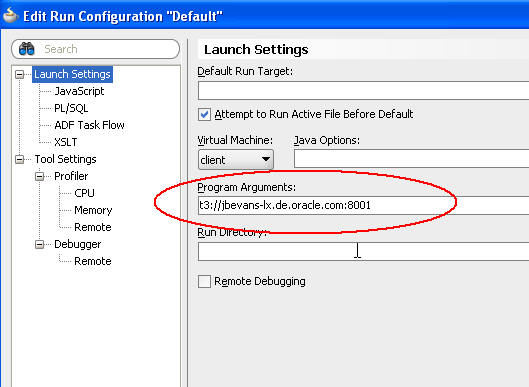
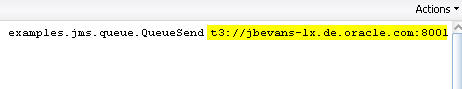
Log in to a machine with a WebLogic installation and create a directory to contain the source and code matching the package name, e.g. $HOME/examples/jms/queue. Copy the above QueueSend.java file to this directory.

* Set the CLASSPATH and environment to match the WebLogic server environment.   
  Go to $MIDDLEWARE\_HOME/user\_projects/domains/base\_domain/bin  and execute  
    
  . ./setDomainEnv.sh
* Collect the following information required to run the script:
* The JNDI name of a JMS queue to use   
    
  In the Weblogic server console > Services > Messaging > JMS Modules > (Module name, e.g. TestJMSModule) > (JMS queue name, e.g. TestJMSQueue)  
  Select the queue and note its JNDI name,   
  e.g. jms/TestJMSQueue
* The JNDI name of a connection factory to connect to the queue   
    
  Follow the same path as above to get the connection factory for the above queue, e.g.  
  TestConnectionFactory and its JNDI name  
  e.g. jms/TestConnectionFactory
* The URL and port of the WebLogic server running the above queueCheck the JMS server for the above queue and the managed server it is targeted to, for example soa\_server1. Now find the port this managed server is listening on, by looking at its entry under Environment > Servers in the WLS console,   
  e.g. 8001   
  The URL for the server to be given to the QueueSend program in this example will therefore be t3://host.domain:8001   
  e.g. t3://jbevans-lx.de.oracle.com:8001
* Edit QueueSend.java and enter the above queue name and connection factory respectively under

|  |
| --- |
| ... public final static String  JMS\_FACTORY=" jms/TestConnectionFactory ";  ...  public final static String QUEUE=" jms/TestJMSQueue ";  ... |

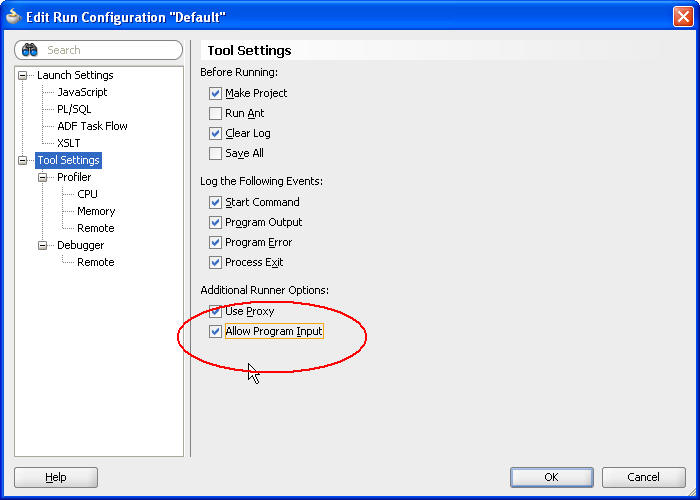
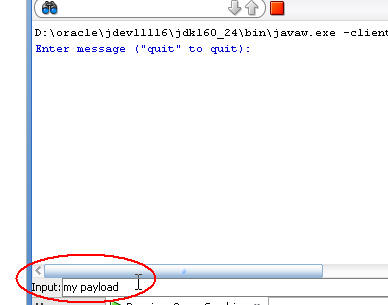
* Compile QueueSend.java using  
    
  javac QueueSend.java
* Go to the source’s top-level directory and execute it using  
    
  java examples.jms.queue.QueueSend t3://jbevans-lx.de.oracle.com:8001
* This will prompt for a text input or “quit” to end.
* In the WLS console, go to the queue and select Monitoring to confirm that a new message was written to the queue.

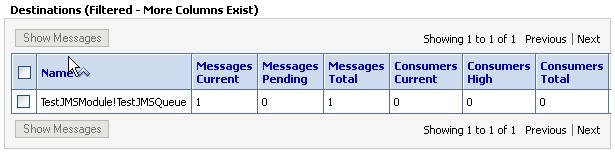
### 2.2 From JDeveloper

* Create a new application in JDeveloper, called, for example JMSTests.
* When prompted for a project name, enter QueueSend and select Java as the technology  
    
  
* Default Package = examples.jms.queue (but you can enter anything here as you will overwrite it in the code later).   
  Leave the other values at their defaults.   
    
  
* Press Finish
* Create a new Java class called QueueSend and use the default values  
    
    
    
  This will create a file called QueueSend.java.
* Open QueueSend.java, if it is not already open and replace all its contents with the QueueSend java code listed above  
    
  
* Some lines might have warnings due to unfound objects.   
  These are due to missing libraries in the JDeveloper project.   
    
  
* Add the following libraries to the JDeveloper project:
* right-click the QueueSend  project in the navigation menu and select Libraries and Classpath , then Add JAR/Directory
* Go to the folder containing the JDeveloper installation and find/choose the file javax.jms\_1.1.1.jar , e.g. at D:\oracle\jdev11116\modules\javax.jms\_1.1.1.jar  
    
  Do the same for the weblogic.jar file located, for example in D:\oracle\jdev11116\wlserver\_10.3\server\lib\weblogic.jar   
    
    
  
* Now you should be able to compile the project, for example by selecting the Make or Rebuild icons  
    
  
* If you try to execute the project, you will get a usage message, as it requires a parameter pointing to the WLS installation containing the JMS queue, for example t3://jbevans-lx.de.oracle.com:8001 . You can automatically pass this parameter to the program from JDeveloper by editing the project’s Run/Debug/Profile.   
  Select the project properties, select Run/Debug/Profile and edit the Default run configuration  
    
    
    
  and add the connection parameter to the Program Arguments field  
    
    
    
  If you execute it again, you will see that it has passed the parameter to the start command  
    
  
* If you get a ClassNotFoundException for the class weblogic.jndi.WLInitialContextFactory , then check that the weblogic.jar file was correctly added to the project in one of the earlier steps above.

Set the values of JMS\_FACTORY and QUEUE the same way as described above in the description of how to use this from a Linux file system, i.e.

|  |
| --- |
| ... public final static String  JMS\_FACTORY=" jms/TestConnectionFactory ";  ...  public final static String QUEUE=" jms/TestJMSQueue ";  ... |

* You need to make one more change to the project. If you execute it now, it will prompt for the payload for the JMS message, but you won’t be able to enter it by default in JDeveloper. You need to enable program input for the project first.  
    
  Select the project’s properties, then Tool Settings, then check the Allow Program Input checkbox at the bottom and Save.  
    
  
* Now when you execute the project, you will get a text entry field at the bottom into which you can enter the payload. You can enter multiple messages until you enter “quit”, which will cause the program to stop.  
    
  

The following screen shot shows the TestJMSQueue’s Monitoring page, after a message was sent to the queue:   


This concludes the sample. In the following post I will show you how to read the message from the queue again.

### JMS Step 3 - Using the QueueReceive.java Sample Program to Read a Message from a JMS Queue

#### By John-Brown.Evans on [Nov 21, 2012](https://blogs.oracle.com/soaproactive/entry/jms_step_3_using_the)

This post continues the series of JMS articles which demonstrate how to use JMS queues in a SOA context.

In the first post,   
[JMS Step 1 - How to Create a Simple JMS Queue in Weblogic Server 11g](https://blogs.oracle.com/soaproactive/entry/how_to_create_a_simple)   
we looked at how to create a JMS queue and its dependent objects in WebLogic Server.

In the previous post,   
[JMS Step 2 - Using the QueueSend.java Sample Program to Send a Message to a JMS Queue](https://blogs.oracle.com/soaproactive/entry/jms_step_2_using_the)   
I showed how to write a message to that JMS queue using the QueueSend.java sample program.

In this article, we will use a similar sample, the QueueReceive.java program to read the message from that queue. Please review the previous posts if you have not already done so, as they contain prerequisites for executing the sample in this article.

## 1. Source code

The following java code will be used to read the message(s) from the JMS queue. As with the previous example, it is based on a sample program shipped with the WebLogic Server installation. The sample is not installed by default, but needs to be installed manually using the WebLogic Server Custom Installation option, together with many, other useful samples. You can either copy-paste the following code into your editor, or install all the samples.

The knowledge base article in My Oracle Support:

[How To Install WebLogic Server and JMS Samples in WLS 10.3.x (Doc ID 1499719.1)](https://support.oracle.com/epmos/faces/ui/km/SearchDocDisplay.jspx?_afrLoop=7121038701894&type=DOCUMENT&id=1499719.1)

describes how to install the samples.

### QueueReceive.java

|  |
| --- |
| package examples.jms.queue;  import java.util.Hashtable;  import javax.jms.\*;  import javax.naming.Context;  import javax.naming.InitialContext;  import javax.naming.NamingException;  /\*\*  \* This example shows how to establish a connection to  \* and receive messages from a JMS queue. The classes in this  \* package operate on the same JMS queue. Run the classes together to  \* witness messages being sent and received, and to browse the queue  \* for messages. This class is used to receive and remove messages  \* from the queue.  \*  \* @author Copyright (c) 1999-2005 by BEA Systems, Inc. All Rights Reserved.  \*/  public class QueueReceive implements MessageListener  {  // Defines the JNDI context factory.  public final static String JNDI\_FACTORY="weblogic.jndi.WLInitialContextFactory";  // Defines the JMS connection factory for the queue.  public final static String JMS\_FACTORY="jms/TestConnectionFactory";  // Defines the queue.  public final static String QUEUE="jms/TestJMSQueue";  private QueueConnectionFactory qconFactory;  private QueueConnection qcon;  private QueueSession qsession;  private QueueReceiver qreceiver;  private Queue queue;  private boolean quit = false;  /\*\*  \* Message listener interface.  \* @param msg message  \*/  public void onMessage(Message msg)  {  try {  String msgText;  if (msg instanceof TextMessage) {  msgText = ((TextMessage)msg).getText();  } else {  msgText = msg.toString();  }  System.out.println("Message Received: "+ msgText );  if (msgText.equalsIgnoreCase("quit")) {  synchronized(this) {  quit = true;  this.notifyAll(); // Notify main thread to quit  }  }  } catch (JMSException jmse) {  System.err.println("An exception occurred: "+jmse.getMessage());  }  }  /\*\*  \* Creates all the necessary objects for receiving  \* messages from a JMS queue.  \*  \* @param ctx JNDI initial context  \* @param queueName name of queue  \* @exception NamingException if operation cannot be performed  \* @exception JMSException if JMS fails to initialize due to internal error  \*/  public void init(Context ctx, String queueName)  throws NamingException, JMSException  {  qconFactory = (QueueConnectionFactory) ctx.lookup(JMS\_FACTORY);  qcon = qconFactory.createQueueConnection();  qsession = qcon.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);  queue = (Queue) ctx.lookup(queueName);  qreceiver = qsession.createReceiver(queue);  qreceiver.setMessageListener(this);  qcon.start();  }  /\*\*  \* Closes JMS objects.  \* @exception JMSException if JMS fails to close objects due to internal error  \*/  public void close()throws JMSException  {  qreceiver.close();  qsession.close();  qcon.close();  }  /\*\*  \* main() method.  \*  \* @param args WebLogic Server URL  \* @exception Exception if execution fails  \*/  public static void main(String[] args) throws Exception {  if (args.length != 1) {  System.out.println("Usage: java examples.jms.queue.QueueReceive WebLogicURL");  return;  }  InitialContext ic = getInitialContext(args[0]);  QueueReceive qr = new QueueReceive();  qr.init(ic, QUEUE);  System.out.println(  "JMS Ready To Receive Messages (To quit, send a \"quit\" message).");  // Wait until a "quit" message has been received.  synchronized(qr) {  while (! qr.quit) {  try {  qr.wait();  } catch (InterruptedException ie) {}  }  }  qr.close();  }  private static InitialContext getInitialContext(String url)  throws NamingException  {  Hashtable env = new Hashtable();  env.put(Context.INITIAL\_CONTEXT\_FACTORY, JNDI\_FACTORY);  env.put(Context.PROVIDER\_URL, url);  return new InitialContext(env);  }  } |

## 2. How to Use This Class

### 2.1 From the file system on Linux

This section describes how to use the class from the file system of a WebLogic Server installation.

Log in to a machine with a WebLogic Server installation and create a directory to contain the source and code matching the package name, e.g. span>$HOME/examples/jms/queue. Copy the above QueueReceive.java file to this directory.

* Set the CLASSPATH and environment to match the WebLogic server environment.   
  Go to $MIDDLEWARE\_HOME/user\_projects/domains/base\_domain/bin  and execute  
    
  . ./setDomainEnv.sh
* Collect the following information required to run the script:
* The JNDI name of the JMS queue to use   
  In the WebLogic server console > Services > Messaging > JMS Modules > Module name, (e.g. TestJMSModule) > JMS queue name, (e.g. TestJMSQueue)   
  select the queue and note its JNDI name,   
  e.g. jms/TestJMSQueue
* The JNDI name of the connection factory to use to connect to the queue   
  Follow the same path as above to get the connection factory for the above queue, e.g. TestConnectionFactory and its JNDI name   
  e.g. jms/TestConnectionFactory
* The URL and port of the WebLogic server running the above queueCheck the JMS server for the above queue and the managed server it is targeted to, for example soa\_server1. Now find the port this managed server is listening on, by looking at its entry under Environment > Servers in the WLS console,   
  e.g. 8001   
  The URL for the server to be passed to the QueueReceive program will therefore be t3://host.domain:8001   
  e.g. t3://jbevans-lx.de.oracle.com:8001
* Edit Queue Receive .java and enter the above queue name and connection factory respectively under

|  |
| --- |
| ...  public final static String JMS\_FACTORY="jms/TestConnectionFactory";  ...  public final static String QUEUE="jms/TestJMSQueue";  ... |

* Compile Queue Receive .java using  
    
  javac Queue Receive .java
* Go to the source’s top-level directory and execute it using  
    
  java examples.jms.queue.Queue Receive   t3://jbevans-lx.de.oracle.com:8001
* This will print a message that it is ready to receive messages or to send a “quit” message to end.
* The program will read all messages in the queue and print them to the standard output until it receives a message with the payload “quit”.

### 2.2 From JDeveloper

The steps from JDeveloper are the same as those used for the previous program QueueSend.java, which is used to send a message to the queue. So we won't repeat them here. Please see the previous blog post at   
[JMS Step 2 - Using the QueueSend.java Sample Program to Send a Message to a JMS Queue](https://blogs.oracle.com/roller-ui/authoring/preview/soaproactive/?previewEntry=jms_step_2_using_the)   
and apply the same steps in that example to the QueueReceive.java program.

This concludes the example. In the following post we will create a BPEL process which writes a message based on an XML schema to the queue.