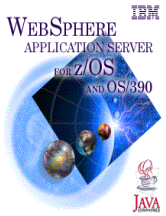


IBM Washington Systems Center

WebSphere Application Server V4.0.1 for z/OS and OS/390

Configuring Applications for Java Message Service (JMS) and setting up JMS Resources on z/OS and OS/390

We welcome your comments and corrections. Our goal is for this document to be 100% accurate. Your comments should be addressed directly to the author of this document.

Version Date: June 13, 2002

Chao-lin Liu
IBM Washington Systems Center
800 N Frederick Ave
Gaithersburg, MD 20879

Notes ID:	Chao-lin Liu/Bethesda/IBM
Internet:	cliu@us.ibm.com
T/L	372-2153
External	301-240-2153

Acknowledgments

The author gratefully acknowledge the contributions made by the following individuals in reviewing and ensuring the accuracy of this paper:

- John Hutchinson IBM USA
- Mike Cox IBM USA

Configuring Applications for Java Message Service (JMS) and setting up JMS Resources on z/OS and OS/390

1.1 Java Message Service (JMS) Introduction

Overview

Enterprise messaging products (also called MOM - Message Oriented Middleware) are an essential component for integrating intra-company operations. They permit separate business components to be combined into a reliable, yet flexible, system. JMS provides a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprises messaging product.

Java Message Service (JMS) provides a framework for developing and supporting Java software components that communicate by creating, sending, receiving and reading messages. This method of communication, known as messaging, allows components to interact asynchronously and reliably, knowing only message formats and destinations without concerns about their communication partners

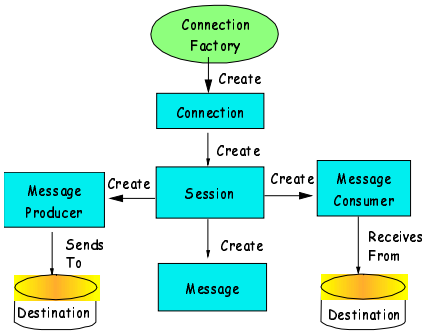
JMS Objectives

- Defines a common set of enterprise messaging concepts and facilities needed to implement sophisticated enterprise applications.
- Minimize the set of concepts a Java programmer must learn to use enterprise messaging products.
- Maximize the portability of messaging applications.

JMS Administered Objects:

- ConnectionFactory - This is the object a client uses to create a connection with a provider.
- Destination - This is the object a client uses to specify the destination of messages it is sending and the source of messages it receives.

Overview of JMS object relationships



Two Messaging Models (also called Styles or Domains):

1. The point-to-point (PTP) model: One message producer creates and sends messages to a queue, from which one message consumer retrieves the messages.
2. The publish/subscribe (PUB/SUB) model: One message producer creates and sends messages to a topic, from which many message consumers retrieve the messages.

For more information, see the JMS API specifications at http://java.sun.com/j2ee/docs/ or http://java.sun.com/products/jms/docs.html .

1.2 JMS implementation in WebSphere is built on WebSphere MQ

JMS is built on the assured messaging capability of WebSphere MQ, formerly known as MQSeries. A typical message flow uses WebSphere MQ messages for input and output. WebSphere MQ Supports a number of APIs - Message Queuing Interface (MQI), Application Messaging Interface (AMI), and Java Messaging Services (JMS); as well as a number of communication models; including PTP and PUB/SUB.

The Java Message Support Pac MA88 implements JMS to enable JMS programs accessing WebSphere MQ systems. MA88 was first available for the distributed environment. MA88 extended its support to z/OS July of 2001. WebSphere V4.0.1 for z/OS allows a JMS client participating in the global transaction via Java Transaction API (JTA) to use RRS.

The JMS API can also be used by WebSphere applications to indirectly access legacy applications. Examples should be CICS and IMS.

1.3 JMS in WebSphere Application Server V4.0.1 for z/OS

JMS Administered objects are modeled as J2EE resource references

- EJB 1.1 declares ConnectionFactory and Destination as resources.
- During deployment, these resources are resolved through the SM EUI.
- java:/comp/env lookups are used by the bean to locate administered objects.

Two types of connection factories are supported in WebSphere V4.0.1 for z/OS

1. RRS Enabled capable of 2 Phase Commit.

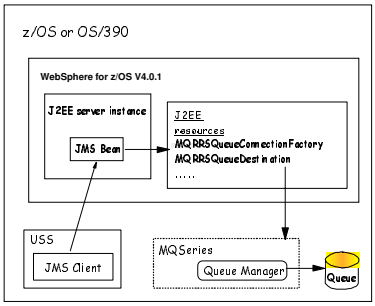
    - MQRRSQueueConnectionFactory
    - MQRRSTopicConnectionFactory

2 MQ Specific capable of 0 and 1 Phase Commit semantics only. Can be used for messaging out of a global transaction scope.

    - MQQueueConnectionFactory
    - MQTopicConnectionFactory

In general, RRS enabled factories are used to support the J2EE programming model.

Below is a Point-to-Point example of a JMS configuration for WebSphere V4.0.1 for z/OS.



1.4 Software Prerequisites

The following software products are required to deploy and run JMS EJBs:

1. WebSphere Application Server V4.0.1 for z/OS and OS/390. As of this writing, the current available PTF level is 401069.

2 MQSeries V5.2 for z/OS and OS/390. JMS enabling PTF (APAR PQ52271 - PTF UQ59032, superseded by UQ65906), and Timer Services PTF (APAR PQ52891 - PTF UQ59338), are minimum requirements. (**Current PTF maintenance is strongly recommended to avoid unnecessary problems.)**

3 The Java Message Service Support Pac MA88 for z/OS and OS/390. This software can be downloaded from the website: http://www.ibm.com/software/ts/mqseries/txppacs/ma88.html

4 Software Development Kit, SDK, V1.3.1 for z/OS and OS/390. Current available JVM build level is hm131s - 20020207 (service refresh 13).

5 Application Assembly Tool, AAT, for z/OS and OS/390. Current available version is 27. AAT can be downloaded from the website: http://www.ibm.com/software/webservers/appserv/download_v4z.html

6 Systems Management EUI for z/OS. This can be downloaded from WebSphere for z/OS. The default directory is: /usr/lpp/WebSphere/bin/bboninst.exe

At the time of this writing, WebSphere should be able to use MQSI for z/OS V2.1 to provide its Pub/Sub engine. The development team has efforts underway to verify that this is indeed the case. For the time being, we will focus on point-to-point (PTP) requirements, and discuss minimum setup and configurations for two of the above software components, MQSeries and MA88.

1.5 MQSeries V5.2 Setup and Local Queue.

WebSphere Application Server V4.0.1 for z/OS is built on WebSphere MQ (or MQSeries) to exchange messages. Refer to the following manuals for detailed information about WebSphere MQ:

- MQSeries for OS/390 V5.2 System Setup Guide (SC34-5651)
- MQSeries for OS/390 V5.2 Concept and Planning Guide (GC34-5650)

JMS point-to-point (PTP) configurations need a local MQ Queue Manager. The channel initiator is not required for PTP communication with a local queue already defined. **MQ maintenance should be kept up-to-date. Especially the afore-mentioned PTFs.**

Listed below is a sample local queue definition for JMS point-to-point.

```
DEFINE QLOCAL (xxxxxx.LOCAL.QUEUE) REPLACE   +
     DESCR(' Queue for JMS PTP messages ')      +
     PUT (ENABLED)          +
     GET (ENABLED)          +
     TRIGTYPE( NONE )       +
     MSGDLVSQ (FIFO)        +
     MAXDEPTH (1000)        +
     MAXMSGL (2000)         +
     USAGE (NORMAL)
```

NOTE: You can replace the xxxxxx.LOCAL.QUEUE with a name of your choice. For example, your local queue name can be JMSWSD.LOCAL.QUEUE or any naming convention that suits your needs.

1.6 Installing Java Message Service Support Pac, MA88, on z/OS.

MQSeries classes for Java and MQSeries classes for the Java message Services Support Pac (MA88) have recently been made available on z/OS to support the development of Java based MQSeries applications. Special support has been added to the MQSeries implementation of JMS to support the RRS based transaction environment used by WebSphere on z/OS.

WebSphere Application Server V4.0.1 and MA88 Support Pac for z/OS, make it possible to write J2EE compliant EJBs and Web Applications that exploit the JMS APIs.

The file, ma88_zos.tar.Z, contains the software. It can be downloaded from the website:
http://www.ibm.com/software/ts/mqseries/txppacs/ma88.html

To install the ma88 support pac, perform the following:

1. Create a directory called ma88 in the /usr/lpp directive with Permissions 755.

2 Create a HFS data set called hlq.MA88.HFS using the File_systems in ishell. (primary 6 cyls and secondary 1 cyl)

3 Mount the HFS data set hlq.MA88.HFS at the mount point /usr/lpp/ma88, and file system type: HFS using the File_systems in ishell.

4 Within omvs, change the current directory to **/usr/lpp/ma88.**

5 Within omvs, use the following command to uncompress and explode the MA88 :
tar -xozf / ~ download directory/ma88_zos.tar.Z
This creates and populates a directive named mqm in the current directory, /usr/lpp/ma88.

1.7 Assembling the sample JMS application for use on WebSphere

In our example, an .ear file has been provided. But in real life situations, the application must be developed in a tool such as WSAD, and then assembled into an .ear file using the WebSphere V4.0.1 for z/OS Application Assembly Tool (AAT).

This section describes how to take the sample JMS PTP EAR file and deploy it into WebSphere Application Server for z/OS.

There are two sample beans provided to demonstrate the use of the JMS point-to-point and JMS pub/sub APIs in EJBs. As mentioned before, the message broker is not available on z/OS at this time. We'll focus on point-to-point sample deployment. (The majority of this section is taken from the README.txt with some clarity modifications)

All JMS related sample files can be found in the WebSphere directory. Usually it is under the /usr/lpp/WebSphere/samples/jms directive. Below is a summary of files relevant to our deployment processes:

- README.txt A description of how to deploy and configure the JMS sample beans.
- JMSClient.sh Shell script to execute the client for the JMS sample beans.
- JMSSample.jar Deployed ejb-jar file that contains all compiled code for this sample. This jar should be placed on the client's classpath.
- ws390jms.properties
  Properties used to configure the client's behavior at run time.
- JMSSample.ear Deployable application.
- JMSSample-noPubSub.ear
  Deployable application that will not exercise the JMS publish/subscribe APIs. The only differences between this application archive and the JMSSample.ear is that the deployment descriptor for the JMSPubSubLogger bean does not have any resource references defined and the usePubSubLogging environment entry in the JMSSample bean is set to false. There are no differences in the implementation.
- JMSSample-src.jar
  Source code (in ASCII) for the sample beans and client. This jar file also contains the ejb-jar deployment descriptor.

If you wish to skip the AAT configuration, you may use the JMSSample-noPubSub.ear for installing the J2EE application to WebSphere using the SM EUI and proceed with the step Deploying JMS Point-to-Point Sample to WebSphere Application Server for z/OS.

- Import the EAR file - Import the JMSSample.ear file into the Application Assembly Tool (Version 4.00.027 or later) by right-clicking on the Applications folder and selecting Import. A file location dialog should pop up asking for the location of the EAR file that is to be imported. Once the path name of the file is correct, press the OK button to import the file. A new application called "JMS Sample Application" should now be available in the AAT.

- Locate the EJBs - Expand the "JMS Sample Application" node and the "Ejb Jars" folder. A "JMS Sample" node should now be visible. Expanding the "JMS Sample" node should reveal a folder called "Session Beans." Expanding the "Session Beans" folder will reveal two stateless session beans - JMSSample and JMSPubSubLogger.

- Setup environment entry - Disable the Pub/Sub calls made by the JMS Sample bean as follows. Select the JMSSample bean in the left pane. In the right pane, please select the "Environment" tab. On the environment tab you will find two properties. Please verify that the usePubSubLogging environment entry is set to "False" for this bean. If it is set to true, the sample will attempt to use the Pub/Sub APIs.

- Remove Pub/Sub related resource references - After ensuring that the JMS Sample will not attempt to use the Pub/Sub APIs, you will need to remove the Pub/Sub resource-refs that have been declared for the JMSPubSubLogger. To do this, select the "JMSPubSubLogger" entry in the left pane of the AAT. Next, select the "Resource" tab in the right pane of the AAT. Three resource-refs should be visible. In order to delete these resource references, data needs to be put into the "modify" state by selecting "Modify" from the "Selected" menu. Once in the "modify" state, delete the three resource references declared for the JMSPubSubLogger. This step is required if you do not want to define the administered objects on the SM EUI at deploy time. Once the resource references have been deleted, save the updates by selecting "Save" from the "Selected" menu.

- Deploy the EAR in the AAT - Once the beans have been configured for your environment, you will need to deploy the beans in the AAT and export the EAR file. To do this, right click on the "JMS Sample Application" folder and select the "Deploy" option. This will generate the implementations for the Home and Remote interfaces of the bean, as well as the stubs and ties required for RMI/IIOP.

- Export the EAR - Now that the EAR has been deployed, it is time to export it for the deploy phase in the SM EUI. To do this, right click on the "JMS Sample Application" folder and select the "Export" option. A dialog box will pop up asking for the file name and location to export to. Suggestion - export the EAR to a new file so that the IBM supplied EAR file is not overwritten.

1.8 Installing the Application using the System Management End User Interface (SM EUI)

Defining the Required J2EE Resources to Enable JMS Support

You are now ready to set up JMS connection information to the J2EE server. Perform the following:

1. From the System Management End User Interface (SM EUI), create a new conversation. This conversation will be used to add the JMS support.

2 Expand the list of J2EE servers and locate the J2EE server that you wish to use with JMS resources. Right click on the J2EE server and select **Modify.** Make the following changes in the environment variable list:

- Add the following classes from section 1.6 - I*nstalling* Java Message Service Support Pac, MA88, on z/OS, to the **CLASSPATH** variable:

  /~ma88 install directory~/mqm/java/lib/com.ibm.mq.jar and
  /~ma88 install directory~/mqm/java/lib/com.ibm.mqjms.jar

  NOTE: Separate each entry with a **colon (:)**

- Add the MQSeries native libraries for JMS to the **LIBPATH**:

  /~ma88 install directory~/mqm/java/lib

- Press **Selected -> Save** to save your changes.

3 To define a RRS enabled JMS ConnectionFactory as a new J2EE resource perform the following:

- Locate the J2EE Resources folder under your sysplex, right click it and select **Add.**

- In the properties window enter the following:

  a. Set the J2EE resource ConnectionFactory **name**. We used Sample_QCF
  b. Set the J2EE resource ConnectionFactory **type** to **MQRRSQueueConnectionFactory**.
  c. Set the J2EE resource Queue **name**. We used Sample_Queue.
  d. Set the J2EE resource Queue **type** to **MQQueue**.

- Press **Selected -> Save** to save your changes**.** Look for the following message in the status bar to confirm that the operation was completed successfully.
  BBON0515I J2EE Resource name was added.

- You now can create J2EE resource instances which define the connection information. For each of the resource instances we define. Please provide your own names and descriptions. Unless explicitly stated below, the defaults provided by the panels should be accepted.

  a. Expand the J2EE resources you have just created. This displays a J2EE resource instance folder. Right click it and select **Add**.

  b In the properties window enter the following:

    i. Set the ClientID for Sample_QCF instance. i.e. ClientID = JMSSample.
    The JMSSample ClientID is used for all connections created by the Sample_QCF Connection-Factory. This is for concurrency and authentication purposes.

    ii Set the Queue Manager Name for Sample_QCF instance. This is the name of the local queue manager to which this sample application will connect.

    iii Set the MQ Queue Name for Sample_Queue instance - This is the name of the MQ Queue to be used in this sample. It is recommended that a queue is defined explicitly for this sample as this sample will be placing messages to and getting messages from this queue. (see Section 1.5 MQSeries V5.2 Setup and Local Queue for the sample provided.)

    iv Set the Queue Manager Name for Sample_Queue instance- This is the name of the queue manager that hosts the queue. Generally, this will be the same queue manager that was specified in the Sample_QCF instance.

  c Press **Selected -> Save** to save your changes**.** Look for the following message in the status bar to confirm the operation was successfully completed.
  BBON0515I J2EE resource instance (name) was added.

The J2EE server region is now configured to use JMS. Do not commit the conversation yet, because you still need to deploy an Application to make use of this support. This is described in the next section.

Deploying JMS Point-to-Point Sample to WebSphere Application Server for z/OS.

The final task is to take the EAR file, and deploy it to a J2EE server in WebSphere. Complete the following steps:

- Return to the conversation you were previously editing in the SM EUI. Right click on the J2EE server where you install the JMS support and select **Install J2EE Application**.

- In the EAR filename field, enter the name of the EAR file you exported from the AAT, then press OK.
- A resources and references window will pop up. Completely expand the tree of beans in the left pane. Once they are expanded, select the JMSSample bean.

    i. On the EJB tab, specify the physical JNDI name where you would like the bean's home to be regis -tered. The suggested JNDI path is "/samples" and the suggested JNDI name is "JMSSample."
    ii. On the Reference tab, make sure that the EJB reference points to the JMSPubSubLogger and that a green check mark appears in the tab.
    iii. On the J2EE Resource tab, resolve the jms/sampleQ reference to the Sample_Queue resource and the jms/sampleQCF reference to the Sample_QCF resource. A green check mark should appear in the tab.

- Verify that a green check mark is visible over each of the beans shown in the left pane of the resources and references window. If the check marks are not visible, please verify that you have resolved all references. If the check marks are visible, click the OK button on the window. The SM EUI will connect to the server and begin the installation of the bean.

- Commit and activate the conversation.

1.9 Configure the client

The client for the JMSSample bean needs to be configured before it can be used. First, the shell script that will invoke the java client needs to have the CLASSPATH and LIBPATH settings set to include the MQ Series and WebSphere/390 jar files and libraries. The deployed jar file that contains the server stubs and interfaces needs to be put on the client's CLASSPATH. There are two basic ways to do this. The first is to extract the JMSSample.jar from the JMSSample.ear via 'jar xvf JMSSample.ear JMSSample.jar'. The second is to point to the deployed jar in the WebSphere install directory.

Once the shell script is configured, the ws390jms.properties file (or any file specified by the -CFG argument in the shell script) needs to be updated to reflect the actual MQSeries setup used for the EJBs. This involves two steps.

- First, be sure that the client.subscribe.to.status.topic property is set to false. This is the default.
- Second, in order for the client to drive the server side EJB, the client will need to be able to locate the home of the JMSSample in JNDI. The home's location in JNDI is configured by the client.sample.bean.jndi.name property. If you accepted the suggested name when deploying the bean on the server, this property should be set to /samples/JMSSample.

For your reference, listed below is the ws390jms.properties file. No need to make any changes for our exercise.

```
# If true, the client will subscribe to status messages from the EJB
client.subscribe.to.status.topic=false

# JNDI name of the JMS Sample bean
client.sample.bean.jndi.name=/samples/JMSSample

# Local Queue Manager that can reach the message broker
client.tcf.qm.name=MVS

# Queue Manager on which the message broker is running
client.tcf.broker.name=BROKER

# Topic to which the client should subscribe
client.status.topic=WS390JMS/Sample/Status*

# Use JNDI to locate configured JMS administered objects
client.jms.administered.with.jndi=false

# JNDI name of the configured TopicConnectionFactory
client.tcf.jndi.name=/jms/sample/SampleTCF

# JNDI name of the configured Topic
client.status.topic.jndi.name=/jms/sample/StatusTopic
```

Copy and modify the sample JMSClient.sh script.

Listed below is the client shell script that we have used to run the sample JMS bean.

```
#------------------------------------------------------------
# Point to the directory where the JMS Sample is located
#------------------------------------------------------------

SAMPLEHOME=/usr/lpp/WebSphere401/samples/jms

#------------------------------------------------------------
# Point to the configuration file for this client
#------------------------------------------------------------

JMSPROP="$SAMPLEHOME/ws390jms.properties"

#------------------------------------------------------------
# Point to the directory where the WebSphere native libraries
# and jar files are located
#------------------------------------------------------------

WSLIBDIR="/usr/lpp/WebSphere401/lib"
WSJARDIR="/usr/lpp/WebSphere401/lib"

#------------------------------------------------------------
# Point to the directory containing the MQ JMS native libraries
# and the MQ JMS jar files
#------------------------------------------------------------

MQLIBDIR="/usr/lpp/ma88/mqm/java/lib"
MQJARDIR="/usr/lpp/ma88/mqm/java/lib"

#------------------------------------------------------------
# Setup the CLASSPATH and LIBPATH
#------------------------------------------------------------

CLASSPATH="$SAMPLEHOME/JMSSample.jar:$CLASSPATH"
CLASSPATH="$CLASSPATH:$WSJARDIR/ws390crt.jar"
CLASSPATH="$CLASSPATH:$MQJARDIR"
CLASSPATH="$CLASSPATH:$MQJARDIR/com.ibm.mq.jar"
CLASSPATH="$CLASSPATH:$MQJARDIR/com.ibm.mqjms.jar"

LIBPATH="$LIBPATH:$WSLIBDIR:$MQLIBDIR"

export CLASSPATH LIBPATH




#------------------------------------------------------------
```

```
# Execute the code
#-------------------------------------------------------------

java -Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
-Djava.naming.provider.url=iiop://wsc4:4900  com.ibm.ws390.samples.jms.client.JMSSampleClient
-CFG "$JMSPROP"
```

NOTE : the above java commands should be on ONE line.

1.10 Run the JMSSampleClient and verify the output

If the client has been executed successfully, the output should look similar to this:

```
Looking up home /samples/JMSSample...
Narrowing home...
Creating JMSSample bean...
Sending message "Very simple text message" to the app queue...
Sent message.
Receiving message from the app queue...
Received message "Thu Oct 04 15:22:44 GMT 2001: Very simple text message"
  from the app queue.
Calling bean to send 10 messages
Calling bean to receive 10 messages
10 messages received
Calling runRequestReply()
Calling runPutGetPrimitive()
Removing the bean
Cleaning up the JMS resources
```

If any stack traces are listed, an unexpected exception occurred. If the exception originated on the server side, please check the server side output for the source of the problem. If a JMSException is the source of the problem, please use the message number shown in the exception's message to determine the cause in the JMS documentation.