

WebLogic Server 11gR1 Java Messaging Service (JMS) Labs

Introduction

The following hands-on labs are intended to provide an introduction to some of the main features of WebLogic JMS in WLS 11gR1. The labs are intended to give you practice in configuring and using WebLogic JMS, through the Admin Console and online WLST scripts, with a mixture of webapp and standalone client applications.

There are 7 labs in all. These cover:

- JMS Basics – Configuring Queues and Topics (Lab 1)
- JMS Basics – Working with Queues and Topics (Lab 2)
- Working with JMS Queues – Active Message Expiration (Lab 3)
- Configuring Store-and-Forward (SAF) between WLS Domains (Lab 4)
- Configuring Unit-of-Order (UOO) (Lab 5)
- Configuring Unit-of-Work (UOW) (Lab 6)
- Configuring Messaging Bridge between WLS Domains (Lab 7)

You will also find a number of labs covering JMS-related topics in the WebLogic Server 11gR1 Cluster Labs: in particular, these provide examples of how to use Distributed JMS Destinations and JMS Auto Service Migration in WLS 11gR1.

The lab materials are included in a zipped archive file. You can unzip this and store it anywhere on your local machine, this folder will be referred below as %LAB_HOME%. In it, you will find a number of folders:

- Apps – deployable client/test applications
- Clients – standalone JMS consumer/producer classes
- Scripts – WLST scripts for configuring labs 1, 2, 3 and 6
- Templates – Template jar files for creating WLS domains for the labs
- Shortcuts – A number of useful MS-Windows shortcuts for starting servers etc.

Apart from these materials, the only software you will to install is WebLogic Server 11gR1. You can install this wherever you like on your local machine (but note that the shortcuts provided assume a WLS installation directory of c:\wls103 – you may need to modify these for your local environment).

JMS Lab – Setup

You will need to create two simple WebLogic Server domains to work with for this lab. Both domains have one admin server and one managed server:

Domain1 (dizzyworld):

Admin Server (listen port: 7001)

one Managed Server (name: mainServer, listen port: 7003).

Template: WLS103JMSLabDomain1.jar

Domain2 (dizzyworld2):

Admin Server (listen port: 5001)

one Managed Server (name: remoteServer, listen port: 5003).

Template: WLS103JMSLabDomain2.jar

To create the [dizzyworld](#) domain, use the WebLogic Server Configuration Wizard ([Start -> All Programs -> Oracle WebLogic -> Tools -> Configuration Wizard](#)).

Follow these steps:

- Create a new WebLogic domain
- Base this domain on an existing template, browse to the [%LAB_HOME%\Templates](#) directory and select [WLS103JMSLabDomain1.jar](#)
- Keep default domain name and location
- Accept default Admin login/password (“weblogic/weblogic”)
- Accept default Development mode and SUN SDK
- Accept default environment and services settings
- Create

Repeat the procedure to create the [dizzyworld2](#) domain. The only difference is the template name – base this domain on [WLS103JMSLabDomain2.jar](#) template.

Modify the file paths for the shortcuts provided inside [%LAB_HOME%\Shortcuts](#) , these shortcuts will be used through all the labs.

Lab 1 – Creating and Managing JMS Resources

For this lab, you will use the [dizzyworld](#) domain you created in the Lab Setup.

Start the admin server and managed server: shortcuts have been provided in the JMSLab/Shortcuts folder ([dizzyworld Admin Server.Ink](#) and [dizzyworld mainServer.Ink](#)). Start the admin console (<http://localhost:7001/console>) and login as weblogic/weblogic. The following steps describe how to create JMS system resources:

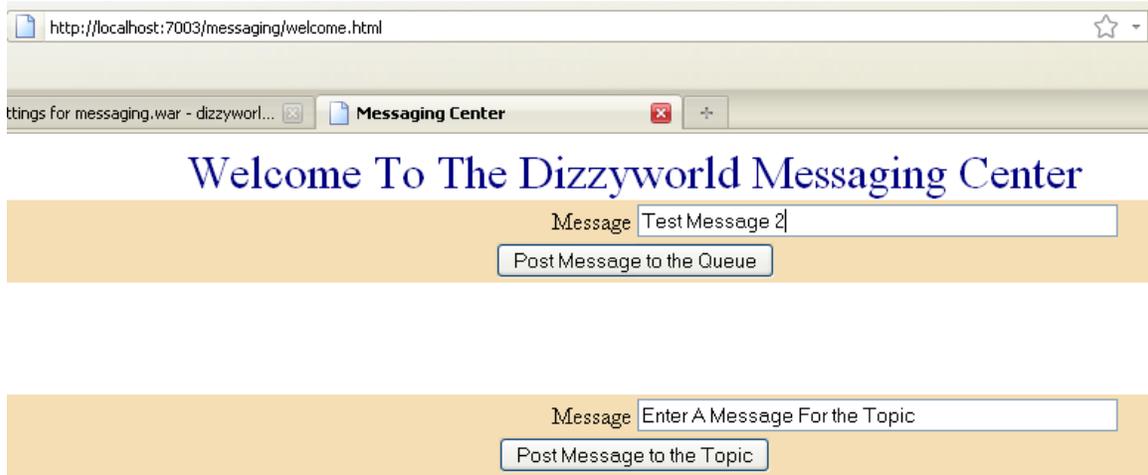
1. Create a new JMS Server
 - In the [Domain Structure](#) window select [Messaging->Services->JMS Servers](#) , click [New](#)
 - Set [msgJMSServer](#) as a Name, keep persistent store to [none](#), click [Next](#)
 - Select [mainServer](#) as a target and click [Finish](#)
2. Create a new JMS Module
 - In the [Domain Structure](#) window select [Messaging->Services->JMS Modules](#), click [New](#)
 - Set [msgJMSSystemResource](#) as Name, click [Next](#)
 - Set [mainServer](#) as a target, click [Next](#)
 - Click [Finish](#)
3. Create a new JMS SubDeployment for the JMS Module
 - In the [Domain Structure](#) window select [Messaging->Services->JMS Modules](#), click [msgJMSSystemResource](#)
 - Go to the [SubDeployments](#) tab, click [New](#)
 - Set [msgSubDeployment](#) as a Name, click [Next](#)
 - Target it to the [msgJMSServer](#), click [Finish](#)
4. Create a Queue
 - In the [Domain Structure](#) window select [Messaging->Services->JMS Modules](#), click [msgJMSSystemResource](#)
 - Click [New](#) button, select [Queue](#), click [Next](#)
 - Set queue name: [msgQueue](#), JNDI name: [PracticeQueue](#), click [Next](#)
 - Select [msgSubDeployment](#) subdeployment and click [Finish](#)
5. Create a Topic
 - In the [Domain Structure](#) window select [Messaging->Services->JMS Modules](#), click [msgJMSSystemResource](#)
 - Click [New](#) button, select [Topic](#), click [Next](#)
 - Set topic name: [msgTopic](#), JNDI name: [PracticeTopic](#), click [Next](#)

- Select `msgSubDeployment` subdeployment and click `Finish`

You could also create all these resources using WLST script, look at `%LAB_HOME%\Scripts\Lab1JMSResources.py` for more details.

From the admin console's Deployments page, deploy the test application `messaging.war` from the `%LAB_HOME%\Apps` folder to the `mainServer`. If you prefer, you can open a command shell (run `setDomainEnv.cmd` from the dizzyworld domain bin directory to set your environment) and type:
`java weblogic.Deployer -username weblogic -password weblogic -targets mainServer -deploy <path>/messaging.war`

Open the test application in a browser (<http://localhost:7003/messaging>). Try sending a few messages to the queue.



Go to the Administration Console, Navigate to `Services->Messaging -> JMS Modules -> msgJMSSystemResource -> msgQueue` and click on the `Monitoring` tab. Check the box and click on `Show Messages` to view the messages on the queue.

Lab2 – Working with Queues and Topics

To see the difference between queues and topics, compare what happens when you run QueueSend/QueueReceive and TopicSend/TopicReceive (you will find these JMS client classes in the `%LAB_HOME%/Clients/Lab2` folder. Open a command shell, set your environment (you can use "WebLogic Shell.Ink" shortcut), cd to `%LAB_HOME%/Clients/Lab2` and type:

```
> java QueueSend t3://localhost:7003
```

Send a few messages:

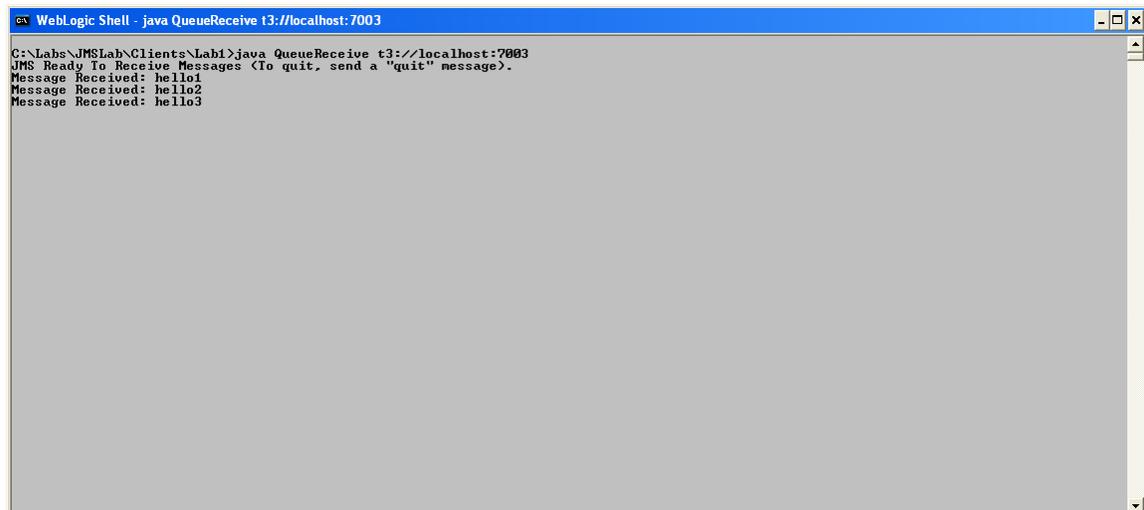


```
WebLogic Shell - java QueueSend t3://localhost:7003
C:\Labs\JMSSLab\Clients\Lab1>java QueueSend t3://localhost:7003
Enter message ("quit" to quit):
hello1
JMS Message Sent: hello1
Enter message ("quit" to quit):
hello2
JMS Message Sent: hello2
Enter message ("quit" to quit):
hello3
JMS Message Sent: hello3
Enter message ("quit" to quit):
```

Now open a second window and type:

```
> java QueueReceive t3://localhost:7003
```

You will see the messages:



```
WebLogic Shell - java QueueReceive t3://localhost:7003
C:\Labs\JMSSLab\Clients\Lab1>java QueueReceive t3://localhost:7003
JMS Ready To Receive Messages (To quit, send a "quit" message).
Message Received: hello1
Message Received: hello2
Message Received: hello3
```

You can continue sending messages (finish with a 'quit' message) and you will see them delivered to the QueueReceive client application.

Now try exactly the same steps, but this time use the TopicSend/TopicRecieve client classes (in the %LAB_HOME%/Clients/Lab2 folder). Open a command shell, set your environment (you can use "WebLogic Shell.lnk" shortcut), cd to %LAB_HOME%/Clients/Lab2 and type:
 > java TopicSend t3://localhost:7003

Publish a few messages to the topic, so:

```

WebLogic Shell - java TopicSend t3://localhost:7003
C:\Labs\JMSLab\Clients\Lab1>java TopicSend t3://localhost:7003
Enter message <"quit" to quit>:
hello1
JMS Message Sent: hello1
Enter message <"quit" to quit>:
hello2
JMS Message Sent: hello2
Enter message <"quit" to quit>:
hello3
JMS Message Sent: hello3
Enter message <"quit" to quit>:
  
```

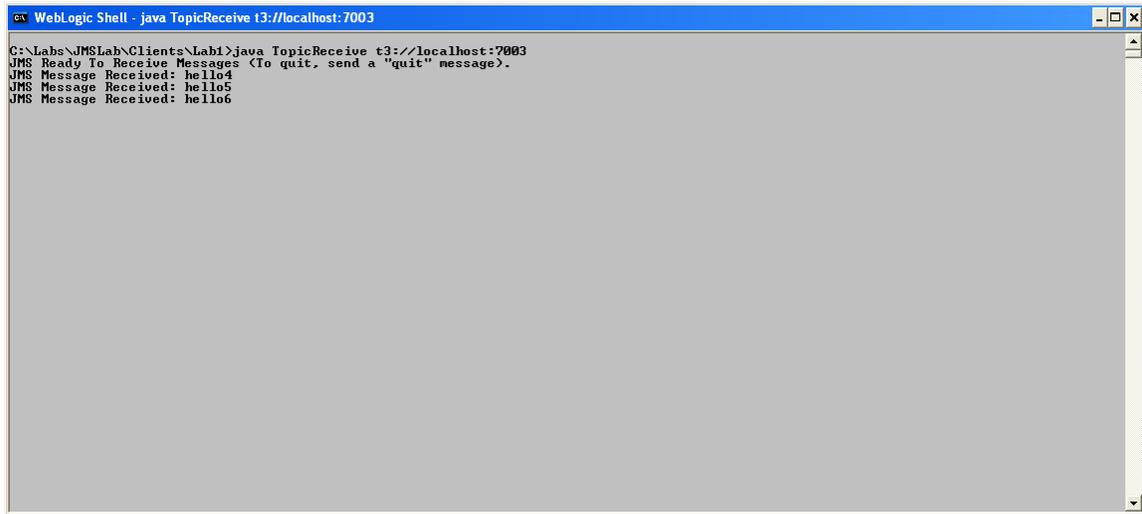
Now open a second window and type:
 > java TopicReceive t3://localhost:7003

This time, you will see that the topic consumer is ready to receive messages but the messages sent to the topic before the consumer subscribed are not received:

```

WebLogic Shell - java TopicReceive t3://localhost:7003
C:\Labs\JMSLab\Clients\Lab1>java TopicReceive t3://localhost:7003
JMS Ready To Receive Messages (To quit, send a "quit" message).
  
```

Now try publishing a few more messages to the topic and now you will see them arrive at the consumer (TopicReceive):



```
WebLogic Shell - java TopicReceive t3://localhost:7003
C:\Labs\JMSLab\Clients\Lab1>java TopicReceive t3://localhost:7003
JMS Ready To Receive Messages (To quit, send a "quit" message).
JMS Message Received: hello4
JMS Message Received: hello5
JMS Message Received: hello6
```

Open a third command shell, set your environment and start another instance of the consumer ([TopicReceive](#)):

```
> java TopicReceive t3://localhost:7003
```

Send a few more messages from the publisher ([TopicSend](#)) and you will see that the messages are received by both consumers. This is one of the differences between the Publish-Subscribe (topic) and Point-to-Point (queue) messaging models. To see this, send a 'quit' message to close the producer and consumer clients, then try the same thing but this time using `QueueSend` for the producer and two instances of `QueueReceive` as the consumer application. This time, you should see that each message goes only to one consumer (by default, they will be distributed round-robin).

Pause/Resume Message Production/Consumption

Start up a Topic producer and consumer(s) as above, then navigate to the admin console page for the `msgTopic` JMS Topic ([dizzyWorld](#) -> [Services](#) -> [Messaging](#) -> [JMS Modules](#) -> [msgJMSSystemResource](#) -> [msgTopic](#)) and click on the **Control** tab. Check the box to select `msgTopic` and then try pausing message production/consumption and watch what happens with your message producer and consumer clients.

Settings for msgTopic

Configuration Monitoring **Control** Security Subdeployment Notes

A JMS destination identifies a queue (Point-To-Point) or a topic (Pub/Sub) that is targeted to a JMS server. This page summarizes the active JMS destinations that have been created for this JMS module.

[Customize this table](#)

Destinations

Production Consumption Insertion Showing 1 to 1 of 1

<input checked="" type="checkbox"/>	Name	ProductionPaused	ConsumptionPaused	InsertionPaused
<input checked="" type="checkbox"/>	msgJMSSyst	false	false	false

Production Consumption Insertion Showing 1 to 1 of 1

Resume production/consumption for the [msgTopic](#) Topic

Using Persistent Message Stores and Durable Subscribers

Create a new persistent file store in which to persist JMS messages. First, create a new folder ('stores') in the domain directory to hold the new message store (%MIDDLEWARE_HOME%\user_projects\dizzyworld\stores).

In the Admin Console navigate to the [dizzyworld -> Services -> Persistent Stores](#) page and create a file-based persistent store, with the following properties:

Name: [myFileStore](#)

Target: [mainServer](#)

Directory: <MIDDLEWARE_HOME>\user_projects\domains\dizzyworld\stores

- Configure msgJMSServer to use myFileStore as its persistent file store. Navigate to [dizzyworld -> Services -> Messaging -> JMS Servers -> msgJMSServer](#) in the admin console,
- go to the [Configuration -> General](#) tab and select [myFileStore](#) as the Persistent Store from the drop-down list, click [Save](#)

Use persistent messaging for [msgTopic](#) and create durable subscribers: this will allow us to publish messages to [msgTopic](#) and know that they will be received by subscribers, even if the subscribers are not connected when the messages are published (unlike in our earlier example). To achieve this, first we need to make sure that messages we post to [msgTopic](#) are persisted by [myJMSServer](#) to the persistent file store. We can do this by specifying `DeliveryMode.PERSISTENT` as the delivery mode to use in our publisher client. In the `%LAB_HOME%/Clients/Lab2` folder you will find a version of the TopicSend class, called [PersisentTopicSend](#), which has been written to do this:

```
public void send(String message) throws JMSEException {
    msg.setText(message);
    publisher.publish(msg, DeliveryMode.PERSISTENT, 4, 0 );
}
```

Next, we need to create durable subscribers to the JMS topic and to do this, we need to change our client code to call `JMSSession.createDurableSubscriber()`

and we also use a unique Client ID to identify both the client and the durable subscription. In the `%LAB_HOME%/Clients/Lab2` folder, you will find a version of the `TopicReceive` class, called `DurableSubscriber`, which has been written in this way:

```
public void init(Context ctx, String topicName, String clientID)
    throws NamingException, JMSException
{
    tconFactory = (TopicConnectionFactory)
        PortableRemoteObject.narrow(ctx.lookup(JMS_FACTORY),
            TopicConnectionFactory.class);
    tcon = tconFactory.createTopicConnection();
    tcon.setClientID(clientID);
    tsession = tcon.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
    topic = (Topic)
        PortableRemoteObject.narrow(ctx.lookup(topicName),
            Topic.class);
    tsubscriber = tsession.createDurableSubscriber(topic, clientID);
    tsubscriber.setMessageListener(this);
    tcon.start();
}
```

To see persistent JMS topics and durable subscribers in action, run a couple of instances of the `DurableSubscriber` class in separate command shells, making sure that each one has a unique subscriber ID, so:

```
> java DurableSubscriber t3://localhost:7003 Client1
```

```
> java DurableSubscriber t3://localhost:7003 Client2
```

Open another command shell and run the `PersistentTopicSend` class:

```
> java PersistentTopicSend t3://localhost:7003
```

Publish a few messages to the topic and, as before, you will see the messages being received by the subscribers. However, now send a 'quit' message to close the two `DurableSubscriber` instances, then rerun the `PersistentTopicSend` class and publish some more messages. With non-durable topics, the new messages would never be delivered to subscribers which do not have an active connection to the JMS Server; however, if you now restart one of the `DurableSubscriber` instances (using the same syntax as before, with the same subscriber ID), you should see that the earlier messages all received as soon as the subscriber client connects to the JMS Server, so:

```

WebLogic Shell - java DurableSubscriber t3://localhost:7003 Client1
C:\Labs\JMSLab\Clients\Lab1>java DurableSubscriber t3://localhost:7003 Client1
JMS Ready To Receive Messages (To quit, send a "quit" message).
JMS Message Received: hello1
JMS Message Received: hello2
JMS Message Received: hello3
JMS Message Received: hello4
JMS Message Received: hello5
JMS Message Received: hello6
  
```

You can view and manage persistent messages that have been published to a durable topic, but not yet consumed by a particular subscriber, using the Admin Console. Navigate to the [dizzyworld -> Services -> Messaging -> JMS Modules -> msgJMSSystemResource -> msgTopic](#) page, open the [Monitoring -> Durable Subscribers](#) tab and you should see the two durable subscribers:

Settings for msgTopic

Configuration **Monitoring** Control Security Subdeployment Notes

Statistics **Durable Subscribers**

Use this page to monitor and manage durable subscribers that are running on a JMS topic. From this page, you can view detailed runtime statistics for each durable subscriber, as well as create new durable subscribers or delete existing ones.

Click on a subscriber to view its configuration details. To view a subscriber's messages, select the check box next to its name, and then click the **Show Messages** button.

[Customize this table](#)

Durable Subscribers (Filtered - More Columns Exist)

New Delete Show Messages Showing 1 to 2 of 2 Previous Next

<input type="checkbox"/>	Client ID ↕	Client ID Policy	Subscription Name	Active	Last Message Received Time	Messages Current Count	Messages Pending Count	Bytes Current Count	Bytes Pending Count
<input type="checkbox"/>	Client1	Restricted	Client1	true	Mon Feb 07 15:26:26 MSK 2011	0	0	0	0
<input type="checkbox"/>	Client2	Restricted	Client2	false	Mon Feb 07 15:21:49 MSK 2011	3	0	20	0

New Delete Show Messages Showing 1 to 2 of 2 Previous Next

Notice that one of these subscribers is marked as (Active = false). Check the box for that Subscription and the Show Messages button will be enabled. Click on this and you will see the unconsumed messages for that durable subscriber:

Summary of JMS Messages

This page summarizes the available messages for a stand-alone queue, a distributed queue, or a topic durable subscriber. Use this page to view message details, create new messages, delete selected messages, move messages to another destination, export message contents in XML format to another file, import XML formatted message contents from another file, or drain all the messages from a destination.

Click on a message to view its contents.

Message Selector:

[Customize this table](#)

JMS Messages (Filtered - More Columns Exist)

<input type="checkbox"/>	ID ↕	CorrId	Time Stamp	State String	JMS Delivery Mode	Message Size
<input type="checkbox"/>	ID:<433511.1297081559500.0>		Mon Feb 07 15:25:59 MSK 2011	visible	Persistent	7
<input type="checkbox"/>	ID:<433511.1297081562562.0>		Mon Feb 07 15:26:02 MSK 2011	visible	Persistent	7
<input type="checkbox"/>	ID:<433511.1297081566531.0>		Mon Feb 07 15:26:06 MSK 2011	visible	Persistent	6

Showing 1 to 3 of 3 Previous | Next

You can click on any of the messages to view them, or use the GUI to delete, move, import or export messages for this topic/subscriber.

Lab 3 – Configuring Active Message Expiration

For this lab, you will use the [dizzyworld](#) domain you created in the Lab Setup; start both [AdminServer](#) and [mainServer](#). The following steps describe how to configure Active Message Expiration for a JMS Queue/Connection Factory:

1. Create connection factory.
 - Navigate to [JMS Modules -> msgJMSSystemResource](#) and click on the [New](#) button to create JMS Resources.
 - Create a new ConnectionFactory (Name: [msgConnectionFactory](#), JNDI Name: [msgConnectionFactory](#), leave the other settings untouched), [click Next](#)
 - Click “[Advanced Targeting](#)” button to add it to the [msgSubDeployment](#) on [msgJMSServer](#), [click Finish](#).
 - When you have created it, click on [msgConnectionFactory](#), go to the “[Default Delivery](#)” sub tab and set the [Default Time-to-Live](#) = 10000 (10secs), [click Save](#).
2. Create a queue.
 - Navigate to [JMS Modules -> msgJMSSystemResource](#) and click on the [New](#) button to create JMS Resources.
 - Create a Queue (Name: [expireQueue](#), JNDI: [expireQueue](#)) and add it to the [msgSubDeployment](#) on [msgJMSServer](#).
3. Navigate to [JMS Modules -> msgJMSSystemResource -> msgQueue](#) and click on the “[Delivery Failure](#)” tab. Set [Expiration Policy](#) to [Redirect](#) and [Error Destination](#) to [expireQueue](#).
4. Navigate to [JMS Servers -> msgJMSServer -> Configuration -> General](#) and set the [Expiration Scan Interval](#) to 20 seconds.

You could also create all these resources using WLST script, look at [%LAB_HOME%\Scripts\Lab3MsgExpiration.py](#) for more details.

From the admin console’s Deployments page, deploy the test application [shoppingcart.war](#) from the [%LAB_HOME\Apps](#) folder to the [mainServer](#). If you prefer, you can open a command shell (run [setDomainEnv.cmd](#) from the [dizzyworld](#) domain bin directory to set your environment) and type:
java weblogic.Deployer -username weblogic -password weblogic -targets mainServer -deploy <path>/shoppingcart.war

Run the test client from a browser (<http://localhost:7003/shoppingcart>), click [Go shopping -> Add to Shopping Cart](#) for a couple of items (will add messages to the queue) and wait for about 10 seconds for the messages to expire and move from [msgQueue](#) to [expireQueue](#). In the admin console, navigate to [JMS Modules ->](#)

[msgJMSSystemResources](#) -> [expireQueue](#) and click on the Monitoring tab.
Check the box and click on "Show Messages" to view the expired messages.

Lab 4 – Store-and-Forward between WebLogic Server Domains

For this lab, you will configure a Store-and-Forward (SAF) Agent to forward JMS messages to a remote queue in another WebLogic Server domain. You will use [dizzyworld](#) as the source domain and [dizzyworld2](#) domain as the target domain. You should start the Admin Servers and managed servers for these domains (you can use shortcuts from [%LAB_HOME%\Shortcuts](#) to start all 4 servers)

Configuring the SAF Source Domain (dizzyworld)

To save time, you may prefer to use the WLST script provided to configure the SAF source domain. The script can be found in the [%LAB_HOME%/Scripts](#) folder: it is called [createSAFsource.py](#). Open a shell window, set your environment by running `setDomainEnv.cmd`, go to [%LAB_HOME%/Scripts](#) and type:

```
>java weblogic.WLST createSAFsource.py
```

For your reference, the following gives a step-by-step description of how to accomplish this using the admin console:

1. Create a new JMS Server called [sourceJMSServer](#) and target it to [mainServer](#).
2. Create a new JMS Module called [sourceJMSModule](#) and target it to [mainServer](#).
3. Create a new Store-and-Forward Agent Agent (Name: [sourceSAFAgent](#), Persistent Store: none, Agent Type: Sending-Only) and target it to [mainServer](#) managed server.
4. Navigate to JMS Modules -> [sourceJMSModule](#), and create a new JMS SubDeployment (Name: [sourceSubDeployment](#), JNDI: [sourceSubDeployment](#)) and target it to [sourceJMSServer](#). This subdeployment will be used for the connection factory ([sourceSAFConnectionFactory](#)) that clients will use to connect to the [sourceJMSServer](#).
5. Create a second JMS SubDeployment (Name: [sourceSAFSubDeployment](#), JNDI: [sourceSAFSubDeployment](#)) and target to [sourceSAFAgent](#). This subdeployment will be used for the SAF Imported Destinations resources.
6. Navigate to JMS Modules -> [sourceJMSModule](#), and create a new Connection Factory (Name: [sourceConnectionFactory](#) , JNDI: [sourceConnectionFactory](#)), and use “Advanced Targeting” to add it to [sourceSubDeployment](#) and target to [sourceJMSServer](#).
7. Navigate to JMS Modules -> [sourceJMSModule](#), and create a new Remote SAF Context (Name: [sourceRemoteSAFContext](#), URL: `t3://localhost:5003, User/Pwd: weblogic/weblogic`)

8. Navigate to JMS Modules -> sourceJMSModule, and create a new SAF Error Handling Resource (Name: [sourceSAFErrorHandler](#), Policy: Log)
9. Navigate to JMS Modules -> sourceJMSModule, and create a new Imported SAF Destinations (Name: [sourceSAFImportedDestinations](#), Remote SAF Context: sourceRemoteSAFContext, SAF Error Handling: sourceSAFErrorHandler) and use Advanced Targeting to add it to the sourceSAFSubDeployment and target to sourceSAFAgent.
10. Navigate to JMS Modules->sourceJMSModule->sourceSAFImportedDestinations and create a new remote Queue (Name: [remoteQueue](#), Remote JNDI: remote Queue).

Configuring the SAF Target Domain (dizzyworld2)

To save time, you may prefer to use the WLST script provided to configure the SAF remote domain. The script can be found in the %LAB_HOME%/Scripts folder: it is called [createSAFremote.py](#). Open a shell window, set your environment by running setDomainEnv.cmd and type:

```
>java weblogic.WLST createSAFremote.py
```

For your reference, the following gives a step-by-step description of how to accomplish this using the admin console:

1. Logon to the Admin Server for dizzyworld2 (<http://localhost:5001/console>) and create the remote JMS resources needed to test out the SAF configuration as follows:
2. Create a new JMS Server called remoteJMSServer and target it to remoteServer.
3. Create a new JMS Module called remoteJMSModule and target it to remoteServer.
4. Navigate to JMS Modules -> remoteJMSModule, and create a new JMS SubDeployment (Name: remoteSubDeployment, JNDI: remoteSubDeployment) and target it to remoteJMSServer
5. Navigate to JMS Modules -> remoteJMSModule, and create a new Connection Factory (Name: remoteConnectionFactory , JNDI: remoteConnectionFactory), and use “Advanced Targeting” to add it to remoteSubDeployment and target to remoteJMSServer.
6. Navigate to JMS Modules -> remoteJMSModule, and create a new Queue (Name: remoteQueue , JNDI: remoteQueue), and use “Advanced Targeting” to add it to remoteSubDeployment and target to remoteJMSServer.

To test the SAF configuration, use the JMS test programs [QueueSend.java](#) and [QueueReceive.java](#). These can be found in the %LAB_HOME%\Clients\SAF

folder. Use the shortcut to open two shell windows (and set the environment by running `setDomainEnv.cmd`), then in the first type:

```
> javac QueueSend.java  
> java QueueSend t3://localhost:7003
```

This will allow you to enter messages (type 'quit' to end the program) – these are forwarded by the SAF Agent on `mainServer` to `remoteQueue` on `remoteServer`. In the second window, type:

```
> javac QueueReceive.java  
> java QueueReceive t3://localhost:5003
```

You should see the messages send to the `dizzyWorld` domain's queue are received by the client listening `dizzyWorld2` domain's queue.

Lab 5 – WebLogic JMS Unit-of-Order (UOO)

For this lab, you will use the [dizzyworld](#) domain you created in the Lab Setup; start both AdminServer and mainServer.

Open 3 command shells (set your environment with `setDomainEnv` – use the shortcuts provided). You will use these to run three JMS client applications, one message producer and two consumers. Cd to `%LAB_HOME%/Clients/UOO` folder, you will find the client `.java` and `.class` files. In the first window, type:

```
>java QueueSend t3://localhost:7003
```

This will allow you to post messages to the `msgQueue` (JNDI: `PracticeQueue`) queue on mainServer's `msgJMSServer` JMS Server.

In the other two windows, type

```
>java QueueReceiveAuto t3://localhost:7003
```

You will see the messages being delivered to the two JMS clients: The client program has `AUTO_ACKNOWLEDGE` set for the JMS Session, so the message delivery is completed as soon as the `onMessage()` call completes. As a result, messages are simply delivered round-robin to the two client programs and if you send a sequence of messages (“hello 1”, hello 2”, “hello 3” ...) you will see all the even-numbered messages appear in one window and all the odd-numbered messages appear in the second window.

Stop the client programs and take a look at `QueueReceiveAck.java`. This version of the message consumer has `CLIENT_ACKNOWLEDGE` set, and you will be prompted to acknowledge receipt of the message – until you enter “y”, the client will not call `msg.acknowledge()` to acknowledge the message and get the next message on the queue. Without UOO order set, the messages will be delivered round-robin to the two clients, and the result is that messages can be consumed out of order.

Run the client programs `QueueSend` and `QueueReceiveAck` in 3 windows:

```
Window1: >java QueueSend t3://localhost:7003
```

```
Window2: >java QueueReceiveAck t3://localhost:7003
```

```
Window3: >java QueueReceiveAck t3://localhost:7003
```

Try sending a numbered sequence of messages (such as “hello1”, “hello 2”, “hello3” ...) and acknowledge the messages in one client but not the other. You will find that you can easily manage to simulate a scenario where the messages

are actually received out of sequence. In many use cases, this is not acceptable: the solution is **unit-of-order functionality**:

Stop the client programs and re-configure the [msgConnectionFactory](#) object on [mainServer](#) to use UOO by navigating in the admin console to [JMS Modules -> msgJMSSystemResource -> msgConnectionFactory](#) and selecting the “Default Delivery” tab. Set [Default Unit of Order for Producers](#) to “User-generated”, with [User-generated Unit-of-Order Name](#) = “uoo-test”. You will need to restart [msgJMSSystemResource](#): go to [View Changes and Restarts](#), click [Restart Checklist](#), check the item and click [Restart](#) button.

Re-run the [QueueSend](#) and [QueueReceiveAck](#) clients as before. Now you will see that while one client is waiting to acknowledge a message, no messages can be consumed by the other client – the messages are always consumed in the same order as they were sent. Only when there are no unconsumed messages, will the JMS Server deliver messages to the other client, and so on.

Lab 6 – WebLogic JMS Unit-of-Work (UOW)

For this lab, you will use the dizzyworld domain you created in the Lab Setup; start both AdminServer and mainServer.

In this lab, you will experiment with WebLogic Server's JMS Unit-of-Work (UOW) feature. This enables message producers to send multiple messages that need to be processed together, as part of a logical group. Unlike Unit-of-Order, where messages are received independently but in strict sequential order, Unit-of-Work addresses scenarios where a number of distinct JMS messages must be handled as a single unit by the consumer.

We will use the same JMS queue and connection factory that we used for the previous example. Note that Unit-of-Order and Unit-of-Work should not be used together, so we must reconfigure msgConnectionFactory not to use UOO. Navigate to [JMS Modules->msgJMSSystemResource->msgConnectionFactory](#) and select the Default Delivery tab. Set 'Default Unit-of-Order for Producer' to None. You will need to restart [msgJMSSystemResource](#) JMS Module to make sure that this change is effective.

To configure UOW on [msgQueue](#) (JNDI: [PracticeQueue](#)), open the Admin Console and navigate to [Services -> Messaging -> JMS Modules -> msgJMSSystemResource -> msgQueue](#), click on 'Advanced' and set [Unit-of-Work \(UOW\) Message Handling Policy](#) to Single Message Delivery.

To try out JMS Unit-of-Order, open two command shells (set the environment with setDomainEnv.cmd) and go to the [%LAB_HOME%/Clients/UOW](#) folder, where you will find sample JMS consumer and producer client classes: these are called [QueueSendUOW](#) and [QueueReceiveUOW](#). They allow you to send and receive messages grouped by UOW, from the command line:

```
>java QueueSendUOW t3://localhost:7003  
>java QueueReceiveUOW t3://localhost:7003
```

Enter a number of messages in the QueueSend window and notice that nothing appears in the QueueReceive window until you send a 'quit' message, which completes the transaction and sends the final message for this UOW. At this point, all the messages are delivered together to the consumer QueueReceive program. Take a look at the client classes and note how the [QueueSend](#) class sets the [JMS_BEA_UnitOfWork](#), [JMS_BEA_UnitOfWorkSequenceNumber](#) and [JMS_BEA_IsUnitOfWorkEnd](#) properties on the messages, how the UOW ID is derived from a UUID (Unique User ID) to ensure uniqueness and how all the messages are sent within the scope of a single transaction to avoid having incomplete UOW messages sent to the JMS Server. In the QueueReceive class, note how a single message of type ObjectMessage holds an Array of messages

that all share the same UOW. Extracts from the sample client classes are given below for reference:

JMS Producer (QueueSendUOW.java)

```

public void send(String message, String strUOW, int SeqNo, boolean isLast)
    throws JMSEException {
    msg.setText(message);
    msg.setStringProperty("JMS_BEA_UnitOfWork", strUOW);
    msg.setIntProperty("JMS_BEA_UnitOfWorkSequenceNumber", SeqNo);
    msg.setBooleanProperty("JMS_BEA_IsUnitOfWorkEnd", isLast);
    qsender.send(msg, DeliveryMode.PERSISTENT, 7, 0);
}

private static void readAndSend(QueueSendUOW qs)
    throws IOException, JMSEException
{
    BufferedReader msgStream = new BufferedReader(new InputStreamReader(System.in));
    String line=null;
    UUID UOW = null;
    boolean quitNow = false;
    int seqNumber = 1;

    UOW = UUID.randomUUID();
    while (! quitNow) {
        System.out.print("Enter message (\quit to quit): \n");
        line = msgStream.readLine();
        if (line != null && line.trim().length() != 0) {
            quitNow = line.equalsIgnoreCase("quit");
            if ( ! quitNow ) {
                qs.send( line, String.valueOf(UOW), seqNumber, false );
            } else {
                qs.send( line, String.valueOf(UOW), seqNumber, true );
            }
        }
        System.out.println("JMS Message Sent: " + line + "\n");
        seqNumber += 1;
    }
}
}
}

```

JMS UOW Consumer (QueueReceiveUOW.java)

```

public void onMessage(Message msg)
{
    try {
        if (msg instanceof ObjectMessage) {
            ArrayList msgList = (ArrayList)((ObjectMessage)msg).getObject();
            int numMsgs = msgList.size();
            System.out.println("Received [" + numMsgs + "] Messages");
            System.out.println("UOW id: " +
                (((TextMessage)msgList.get(0)).getStringProperty("JMS_BEA_UnitOfWork")));
            for ( int i = 0; i < numMsgs; i++ ) {
                System.out.println( "Message[" + i + "]" + ((TextMessage)msgList.get(i)).getText());
            }
        }
    }
}

```

```
    }  
    System.out.println();  
  }  
} catch (JMSEException jmse) {  
  System.err.println("An exception occurred: "+jmse.getMessage());  
}  
}
```

Lab 7 – Configuring a Messaging Bridge

For this lab, you will configure a Messaging Bridge between the [dizzyworld](#) and [dizzyworld2](#) domains. You will use [dizzyworld](#) as the source bridge destination and [dizzyworld2](#) as the target bridge destination. You should start the Admin Servers and managed servers for these domains, using the shortcuts in the [%LAB_HOME%/Shortcuts](#) folder.

In most cases where you need to exchange JMS messages between WebLogic Server domains, you will use the WebLogic JMS Store-and-Forward capabilities we worked with in Lab 3. However, when you are working with other application servers or JMS providers, or when exchanging messages with older WebLogic Server versions, you may need to use the WebLogic JMS Messaging Bridge.

The WebLogic JMS Messaging Bridge will provide fully-transactional message delivery, provided that the target bridge destination is capable of supporting an XA-enabled JMS connection. Although the target queue and connection factory already exist in the target ([dizzyworld2](#)) domain, the connection factory needs to be made XA-enabled. You can do this using the admin console by connecting to the [dizzyworld2](#) admin server (<http://localhost:5001/console>) and navigating to [Services->Messaging->JMS Modules->remoteJMSModule->remoteConnectionFactory](#); select the 'Transactions' tab and check 'XA Connection Factory Enabled', then [Save](#).

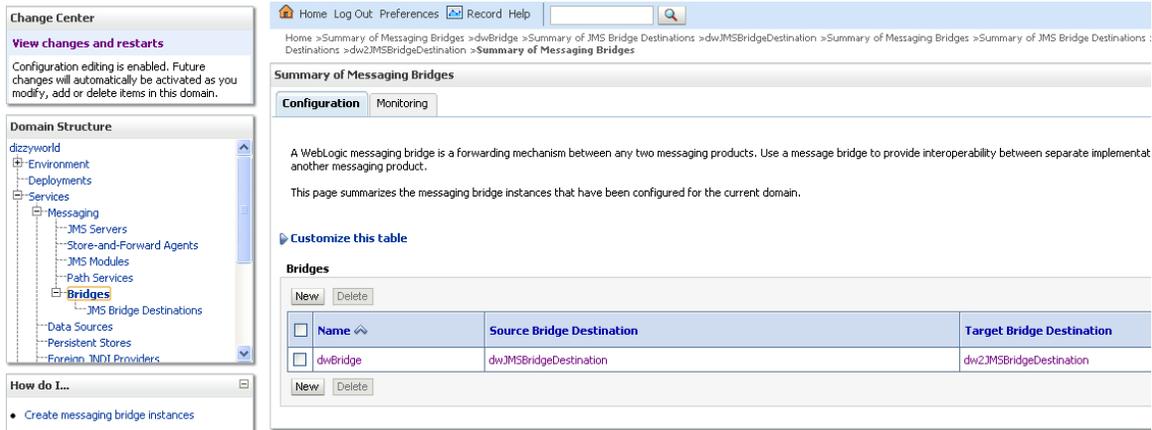
If you wish to use the WLST script provided to configure the Messaging Bridge. ([JMSLab/Scripts/createMessageBridge.py](#)), you will first need to deploy an XA-compliant Resource Adapter, which the WebLogic JMS Messaging Bridge will use to connect to the target system. To do so, open a shell window, set your environment by running [%MIDDLEWARE_HOME%\user_projects\domains\dizzyworld\bin setDomainEnv.cmd](#) and type:

```
> java weblogic.Deployer -username weblogic -password weblogic -targets  
AdminServer,mainServer -deploy  
%MIDDLEWARE\_HOME%\wlserver\_10.3/server/lib/jms-xa-adp
```

Cd to [%LAB_HOME%\Scripts](#) and run

```
> java weblogic.WLST createMessageBridge.py
```

Open Admin Console for [dizzyWorld](#) domain and examine the Messaging Bridge created by WLST script:



You can use the client classes provided in the [%LAB_HOME%/Clients/MessageBridge](#) folder to send and receive test messages. Try sending a few messages using `msgQueue` and `msgConnectionFactory` in the `msgJMSSystemResource` JMS Module and you should see them appear on the `dizzyworld2` remoteServer's `remoteQueue`. To try this out, open a couple of command shells, set your environment and type:

```
>java QueueSend t3://localhost:7003
>java QueueReceive t3://localhost:5003
```

For your reference, the following gives a step-by-step description of how to configure the WebLogic JMS Messaging Bridge using the admin console:

Open the Admin Console (<http://localhost:7001/console>) for the `dizzyworld` source domain. Note the Messaging Bridge configuration below is all done on the source (`dizzyworld`) domain:

1. Navigate to `msgJMSSystemResource->msgConnectionFactory` and select the Transactions tab. Check the XA Connection Factory Enabled box. Now select the Security tab and check the Attach JMSXUserId box. This is required as the Messaging Bridge uses XA transactions to send messages to the target system.
2. Navigate to `Services->Messaging->Bridges->JMS Bridge Destinations` and configure the source JMS Bridge Destination on the `dizzyworld` mainServer with the following properties:

```
Name: dwJMSBridgeDestination
Adapter JNDI Name: eis.jms.WLSConnectionFactoryJNDIXA
Connection URL: t3://localhost:7003
Connection Factory JNDI Name: msgConnectionFactory
Destination JNDI Name: PracticeQueue
```

[Click OK and then select the newly-created dwJMSBridgeDestination]

Destination Type: Queue

User Name: weblogic

Password: weblogic

3. Navigate to Services->Messaging->Bridges->JMS Bridge Destinations and configure the target JMS Bridge Destination on the dizzyworld2 remoteServer with the following properties

Name: dw2JMSBridgeDestination

Adapter JNDI Name: eis.jms.WLSConnectionFactoryJNDIXA

Connection URL: t3://localhost:5003

Connection Factory JNDI Name: remoteConnectionFactory

Destination JNDI Name: remoteQueue

[Click OK and then select the newly-created dwJMSBridgeDestination]

Destination Type: Queue

User Name: weblogic

Password: weblogic

4. Navigate to Services->Messaging->Bridges and configure a Messaging Bridge between the dizzyworld and dizzyworld2 domains:

Name: dwBridge

Select the "Started" checkbox.

Source Bridge Destination: dwJMSBridgeDestination

Target Bridge Destination: dw2JMSBridgeDestination

Quality of Service: Exactly-once

Target: mainServer

You will need to restart JMS Server for these configuration changes to take effect.