Java CAPS 6
Using JCA, Note 1

JCA-based JMS Consumer

Michael Czapski, July 2008

Introduction

In addition to doing it the hard way, which is something a developer could always do and frequently did, Java CAPS 6 gives a developer 4 ways in which to receive messages from a JMS Destination. Two of these will be familiar to the Java CAPS 5.x developer and involve a JMS-triggered JCD and a JMS-triggered eInsight BP. Of the two additional ways in Java CAPS 6, one involves the use of the JBI infrastructure through the JMS Binding Component (JMS BC) and one involves the use of the just introduced evolution of the 5.x eWays variously referred to as Global RARs or Java 1.5 EE JCA adapters.

This discussion revolves around the JCA adapter use and points out, appearances notwithstanding, how easy it will be for a JCD developer to develop JCD-equivalent 'collaborations' using JCA adapters in Java CAPS 6.

Note that OpenESB does not provide this functionality and that this functionality has nothing whatever to do with the JBI.

Credits go to Frank Kieviet and others who proposed and oversaw the evolution of the eWays into JCAs and Dao Tien Tran whose JCA tutorial helped me get started.

Getting down to cases, this document goes through the steps involved in developing, deploying and exercising a JCA MessageDriven Bean (MDB) that is triggered by arrival of a message to a JMS Queue, reads that message and dumps its content to the server.log. There is no business excuse for this – just the mechanics configuring and using the bits.

Create the Queue and Advertise it

Before the receiver can receive messages JMS Queue needs to be created and the JNDI reference to it made available so that the MDB can use it.

Let's create Application Server resources required to find the JMS destination to be used in the project.

Make sure your Application Server is started. Use a web browser to connect to the Application Server Console. Expand the node tree to Resources -> Connectors -> Admin Object Resources. As shown in Figure 1.

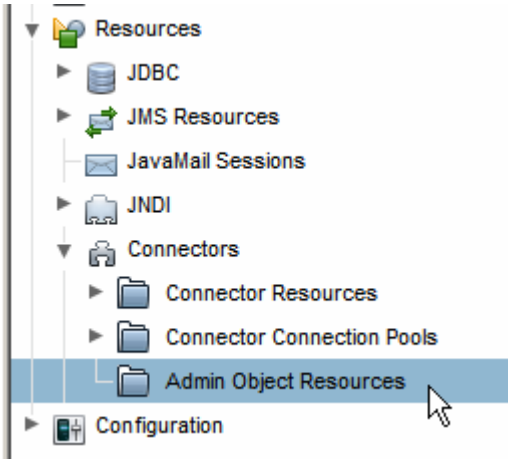


Figure 0-1 Admin Object Resources node

In the window pane that shows up at the right, click “New …” and create a new resource. Give it a JNDI name of jcaps/jms/qReceivers, Resource Type of javax.jms.Queue and select Resource Adapter sun-jms-adapter. Figure 1 illustrates the page used in Step 1 of the process.

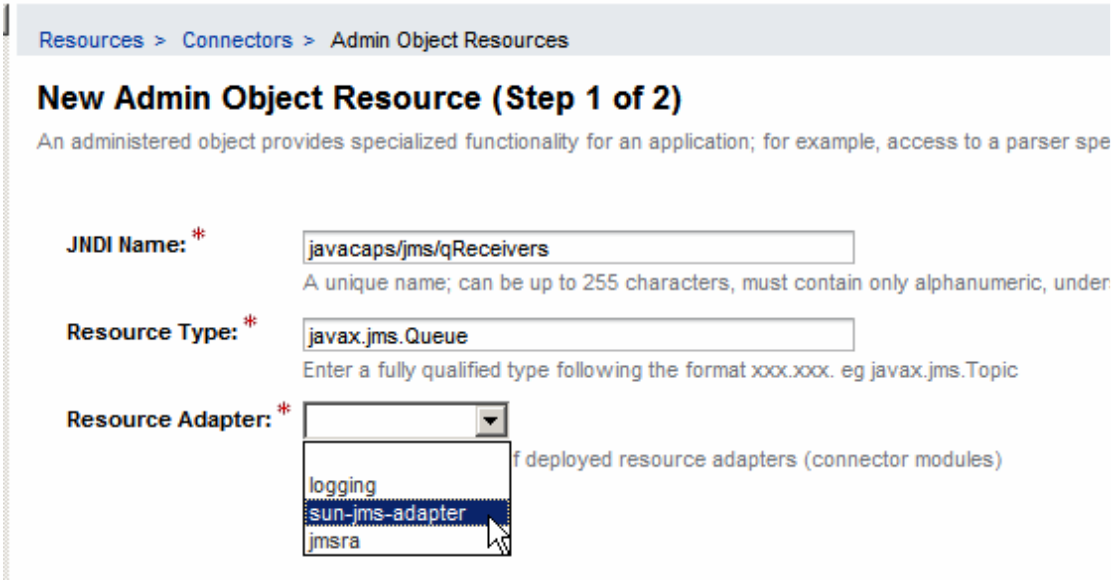


Figure 0-2 Configuring the JNDI name and related properties for the JMS Queue object

Click Next then, in Step 2 of 2, fill in queue name as the value of the Name property, as illustrated in Figure 3.

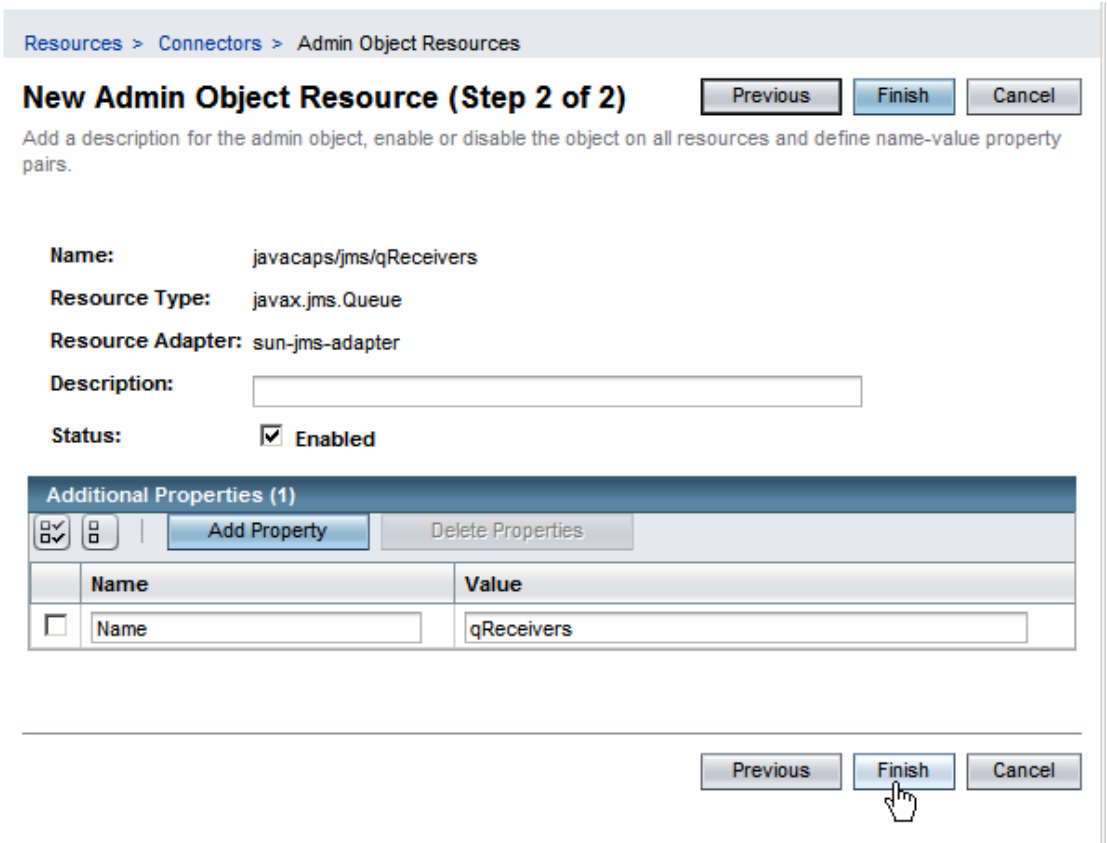


Figure 0-3 Naming the JMS Queue

Once done, click Finish.

So far we created a JMS Queue object called qReceivers and a JDNI reference to it. If all is well the server.log will show INFO-level feedback similar to what is shown in Figure 4.

```
[#|2008-07-11T10:05:38.187+1000|INFO|sun-appserver9.1|javax.enterprise.system.tools.admin|_ThreadID=23;_ThreadName=httpWorke
rThread-54848-3;ResourceDeployEvent -- reference-added aor/javacaps/jms/qReceivers;|ADM1041:Sent the event to instance:[Resc
urceDeployEvent -- reference-added aor/javacaps/jms/qReceivers]|#]

[#|2008-07-11T10:05:38.281+1000|INFO|sun-appserver9.1|javax.enterprise.system.core|_ThreadID=23;_ThreadName=httpWorkerThread
-54848-3;aor:javacaps/jms/qReceivers;|CORE5004: Resource Deployed: [aor:javacaps/jms/qReceivers].|#]
```

Figure 0-4 JNDI reference added

Create a JCA MDB (JCD-like thing)

Optionally, create a new Project Group to collect all related projects together. With a project containing one module this may seem like overkill but for projects containing dozens of modules and, for solutions involving dozens of projects, getting into the habit of organizing projects into groups may be a good idea.

It may come as an unpleasant surprise to Java CAPS 5.x developers that NetBeans does not have a notion of hierarchical project organization. “You win some you lose some”.

Create the project group, JMSConsumer_PG. Key steps are illustrated in Figures 5 and 6.

Figure 5 Start the process of project group creation

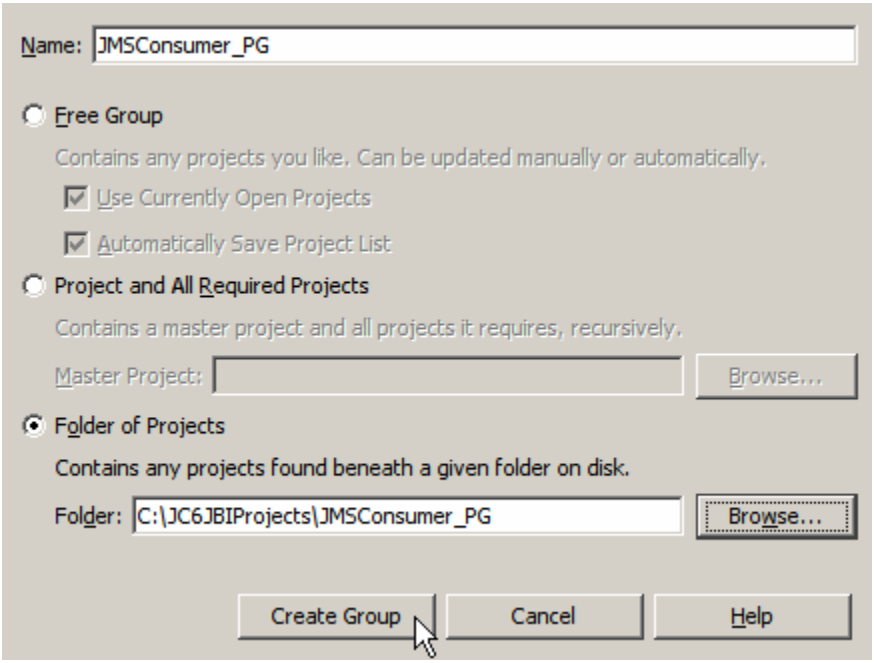


Figure 6 name the project group and nominate the folder to contain its projects

The JCA MDB be an EJB so let's create an Enterprise EJB Module, JMSConsumer_EJBM. Figures 7 and 8 illustrate the major steps.

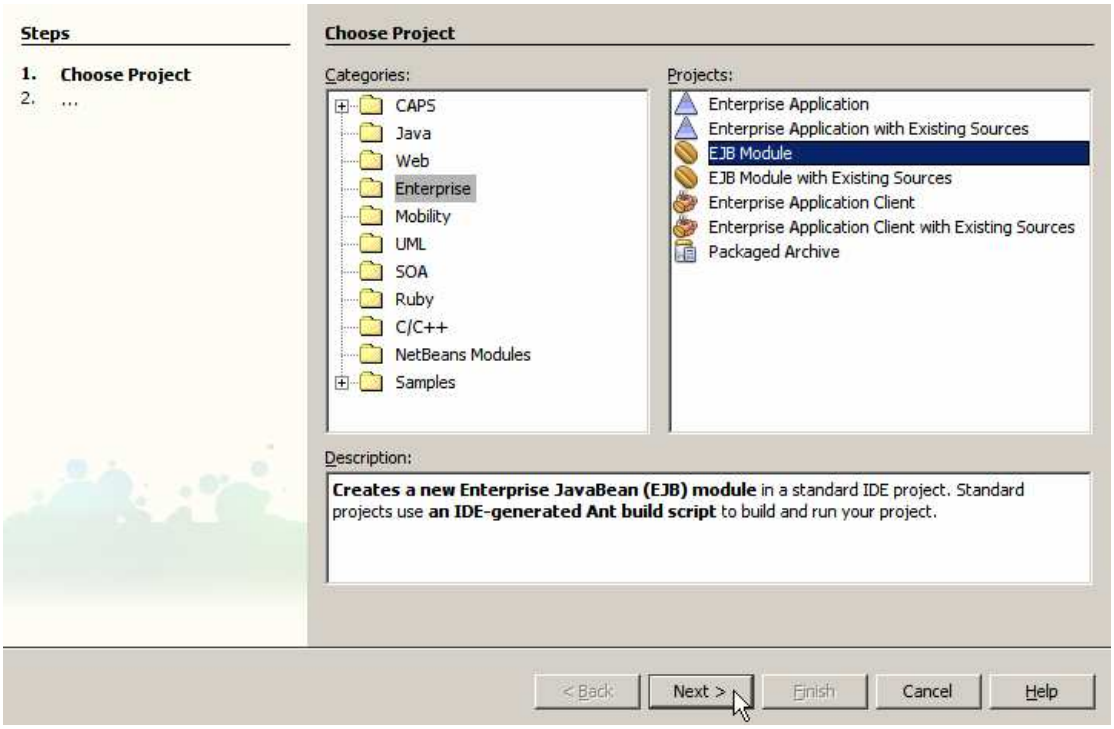


Figure 7 Start EJB Module creation wizard

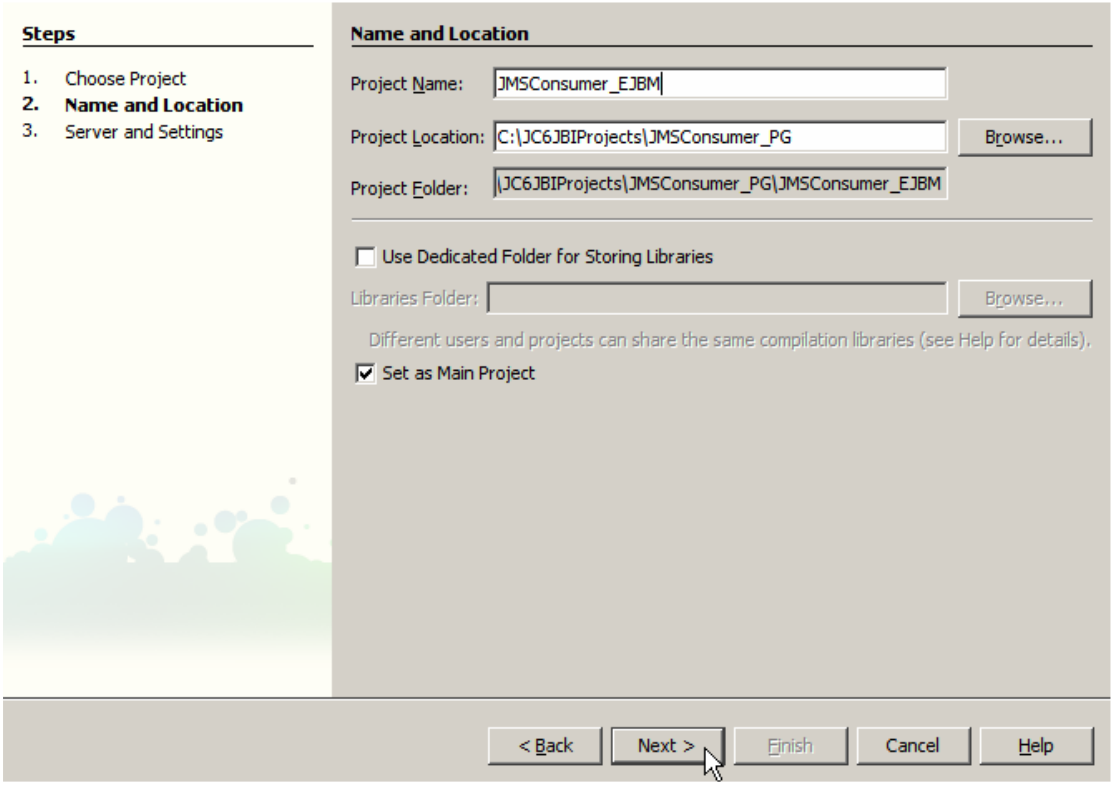


Figure 8 Name the EJB Module project and identify project location

In the EJB Module project let's create a new Enterprise -> JCA MessageDrivenBean, called jcaJMSConsumer. Figures 9 and 10 point out the File Type in the Enterprise category, which to use, and call out the naming of the class and the package.

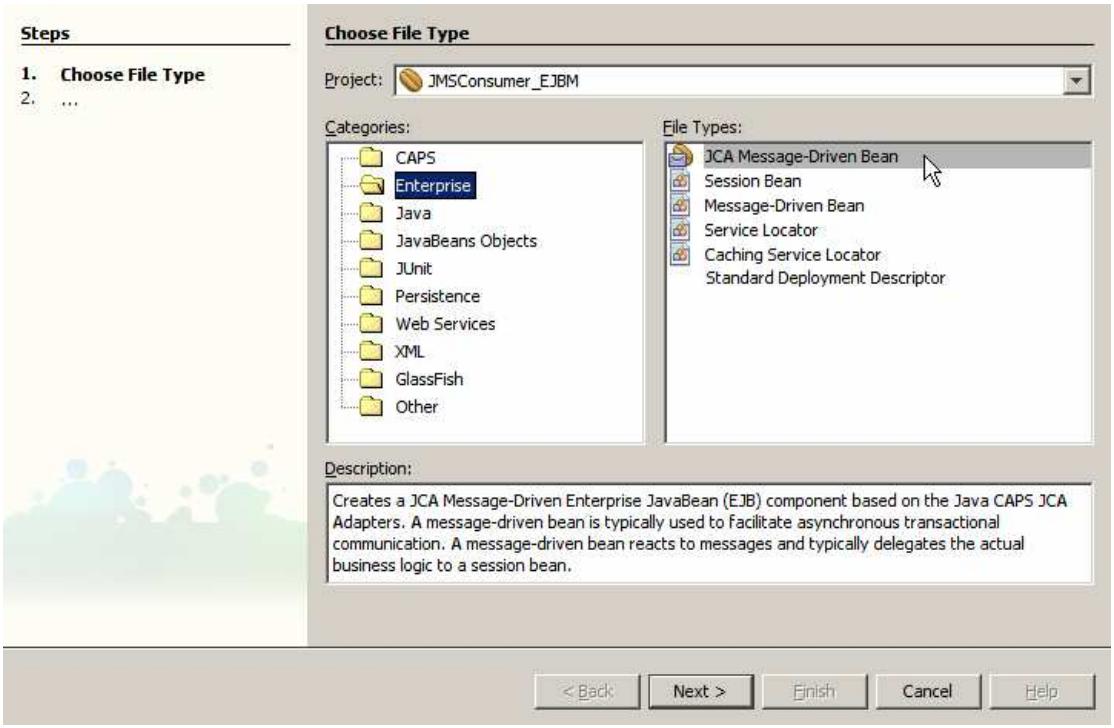


Figure 9 JCA Message-Driven Bean file type

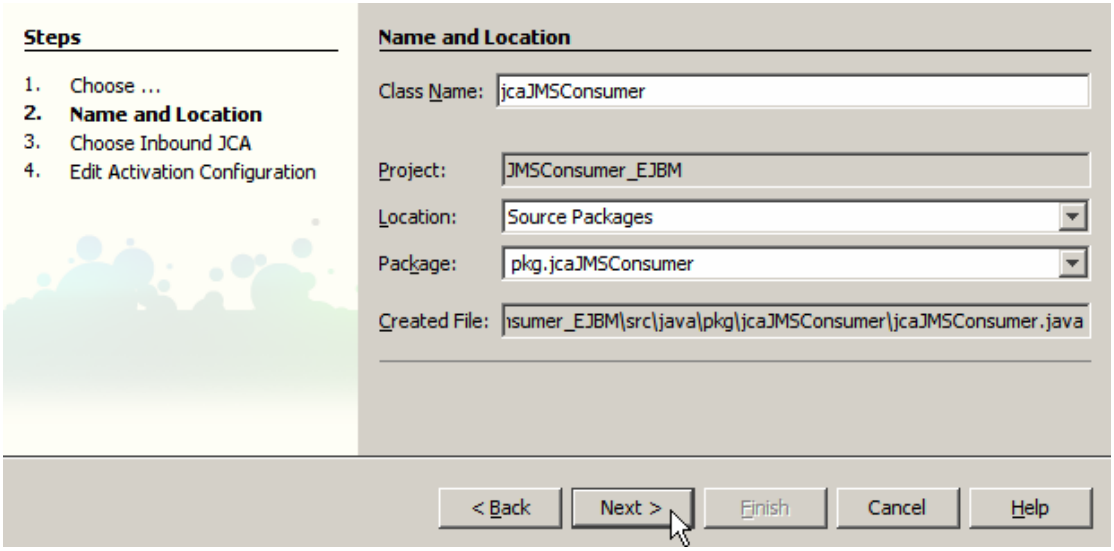


Figure 10 Name the class and the package

In order for the bean to be a JCA MDB one must pick the JCA adapter to use. Selection of adapters on offer will vary with the adapters that were installed. The list shown in Figure 11 reflects the adapters installed as part of the standard Java CAPS 6 General Availability release with no extras. Pick the JMS Adapter and com.stc.connectors.jms.Message, as illustrated in Figure 11. By picking the com.stc.connectors.jms.Message instead of the javax.jms.Message, which is another choice, one makes this MDB's method signature the same as what the JMS-triggered JCD's would be and provides us with the JMS OTD to use, much as a JCD would.

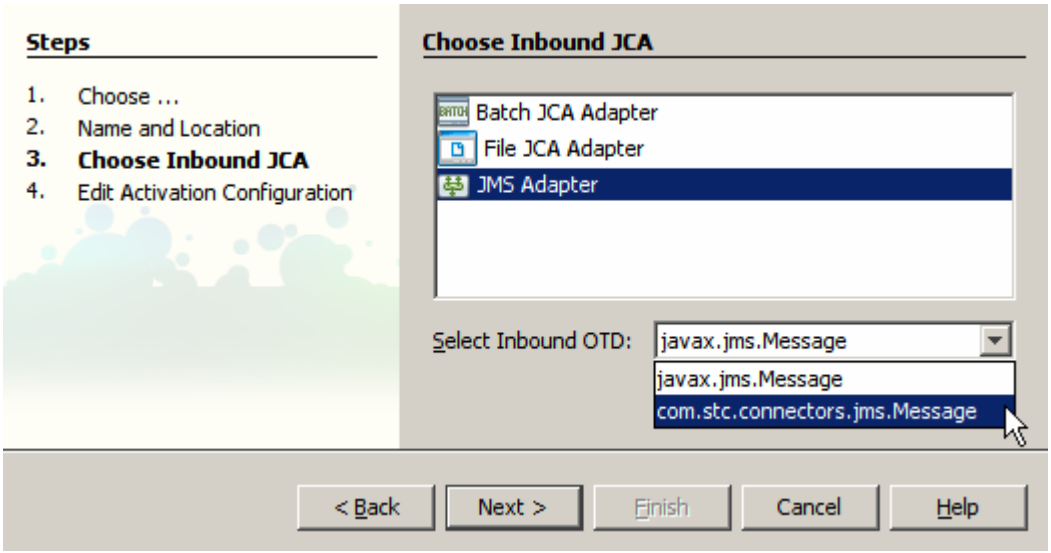


Figure 11 Select JMS Adapter and com.stc.connectors.jms.Message

Click Next.

In the Edit Activation Configuration dialogue box, General Tab, browse the Connection URL resource tree and choose jms/notx/default, as illustrated in Figure 12. This selection determine the JMS server implementation and transactional behavior that will be used at runtime. Resource names and their implied properties are self-evident, I think.

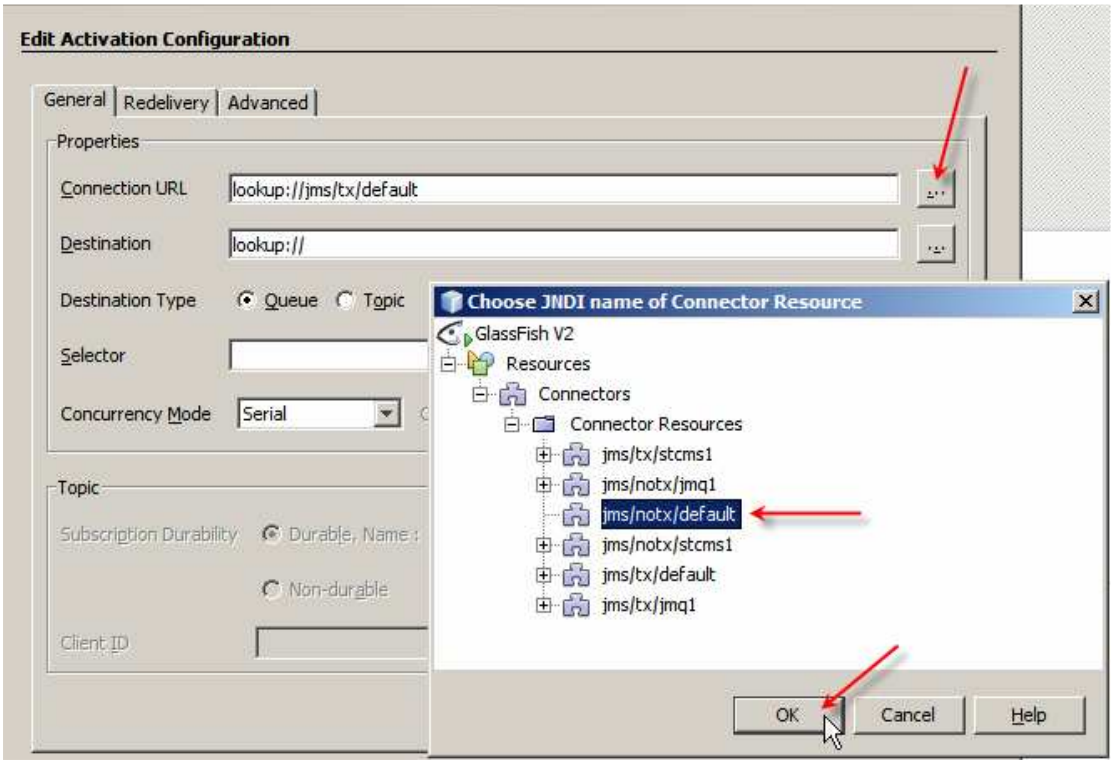


Figure 12 Selecting JMS implementation and transactionality

Browse the Destination resource tree and choose the javacaps/jms/qReceivers object created earlier, as shown in Figure 13. Here we are making the connection between

the MDB and the JMS Destination resource we configured earlier through the Application Server Administration Console.

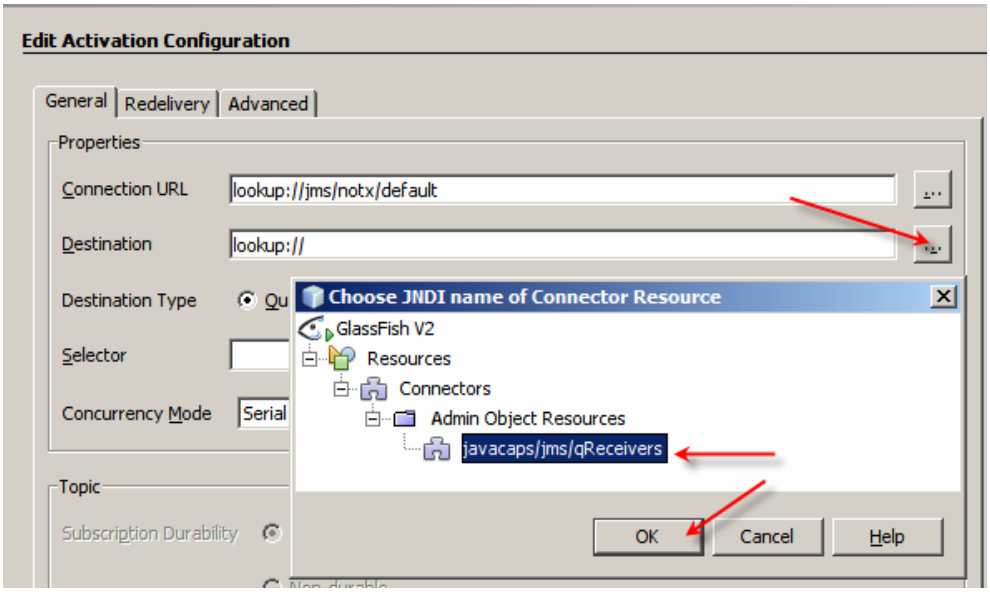


Figure 13 JMS Destination object reference

Click Finish to accept all other default values and complete the wizard.

A skeleton Java class will been created, as shown in Figure 14. A Java developer is likely to know what to do next. A Java CAPS 5.1 integration developer will very likely be interested in the receive method. This is where his/her code would have gone into in a 5.1 JCD.

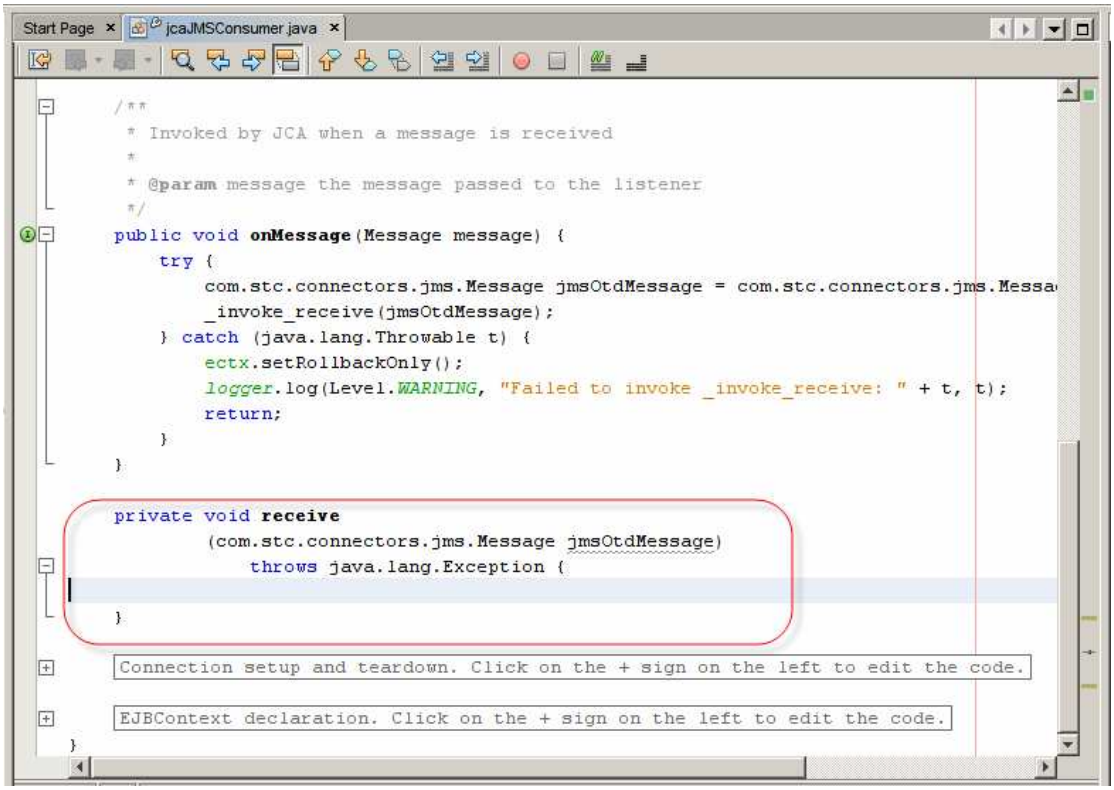


Figure 14 The 'receive' method is of specific interest

The jmsOtdMessage variable is equivalent to the input variable one would have had in a JCD triggered by a JMS message.

To prove it let's get the text message from the JMS queue and write it's content to the server.log using the logger method, familiar from the JCD side of things. This is exactly what a JCD developer would have done in this circumstance. Figure 15 shows the creative code that accomplishes this.

```
private void receive
        (com.stc.connectors.jms.Message jmsOtdMessage)
            throws java.lang.Exception {
    logger.info("Received [" + jmsOtdMessage.getTextMessage() + "]");
}
```

Figure 15 Getting the content of the JMS message and logging it to the server.log

Notice that unlike in a Java CAPS 5.x JCD the logger object does not have the debug method but, rather, it has fine, finer, finest methods. The logger has been brought into line with the java.util.logging API's severity levels and names.

Build and deploy the project.

In the server.log notice INFO-level messages relating to the qReceivers. Figure 16 illustrates this. Notice that because we specified the jms/notx/default Connector Resource the default JMS Message Server was used – STCMS in this case. If we wished to use a specific implementation, Java MQ for example, we could have specified jms/notx/jmq1 instead.

```
[#|2008-07-11T10:24:16.937+1000|INFO|sun-appserver9.1|javax.enterprise.system.tools.deployment|_Th
readID=24;_ThreadName=Thread-1270;|deployed with moduleid = JMSConsumer_EJBM|#]

[#|2008-07-11T10:24:18.281+1000|INFO|sun-appserver9.1|javax.enterprise.system.core.classloading|_T
hreadID=25;_ThreadName=httpWorkerThread-54848-2;JMSConsumer_EJBM;|LDR5010: All ejb(s) of [JMSConsu
mer_EJBM] loaded successfully!|#]

[#|2008-07-11T10:24:18.984+1000|INFO|sun-appserver9.1|com.stc.jmsjca.core.Activation|_ThreadID=26;
_ThreadName=JMSJCA connect;|JMSJCA-E015: [serial-QueueReceiver(lookup://javacaps/jms/qReceivers) @
[stcms://(domain1stcms1)]]: message delivery initiation was successful.|#]
```

Figure 16 Delivery initiation indication in server.log

Exercise the solution

Start the Enterprise Manager Web Console, navigate to your application server instance, choose STCMS and note the JMS Queue qReceivers. Figure 17 shows the Enterprise Manager screen. Non Java CAPS 5.x people may be unaware that this facility exists. 5.x people will be right at home ☺ 5.x people will also notice the Sun Java MQ JMS Server in the EM.

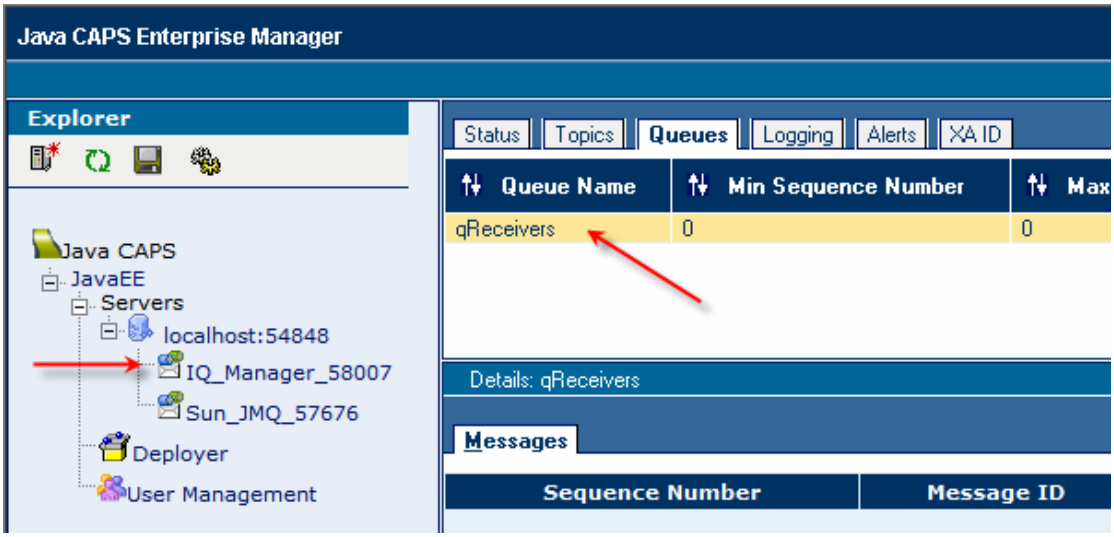


Figure 17 qReceivers in Enterprise Manager

Manually submit a message to that queue and observe message content logged in server.log, as illustrated in Figures 18-20.

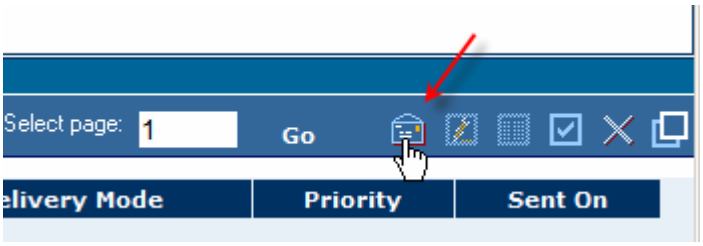


Figure 18 Open Send/Publish New Message dialog box

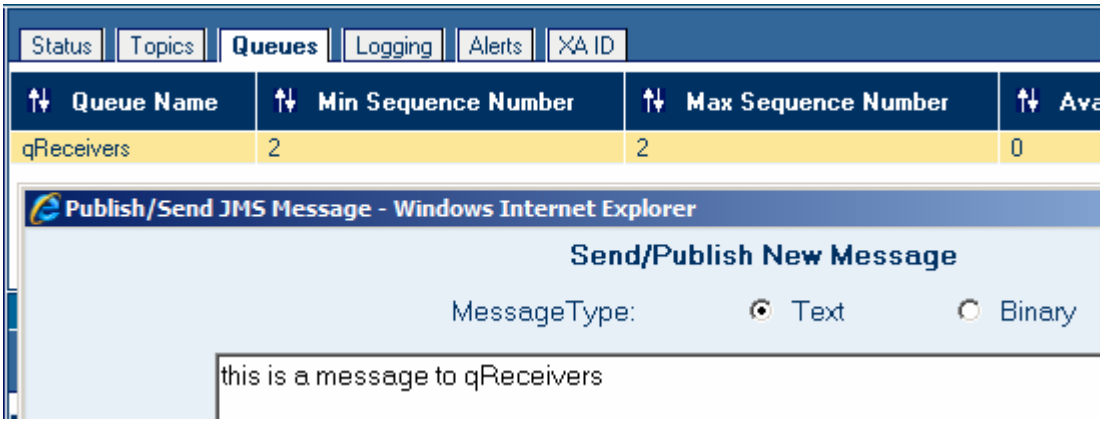


Figure 19 Submit a text message

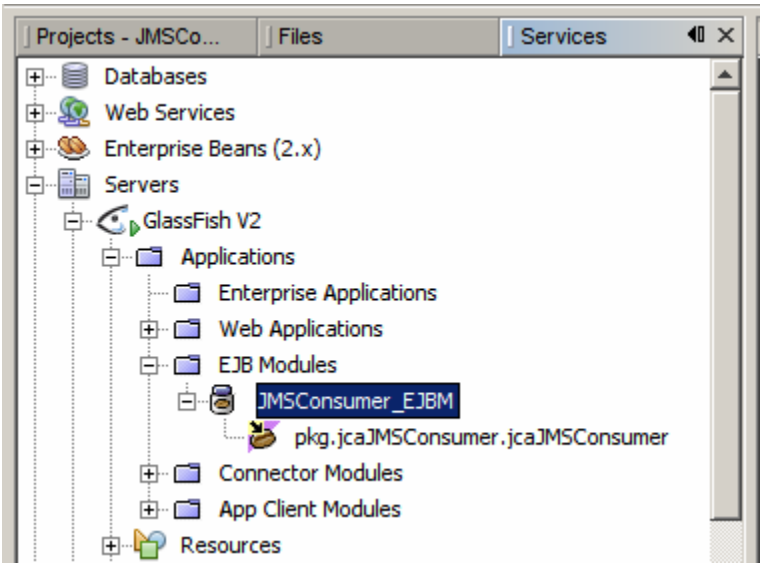
```
[#|2008-07-11T10:34:08.937+1000|INFO|sun-appserver9.1|javax.enterprise.system.core.clas
sloading|_ThreadID=25;_ThreadName=httpWorkerThread-54848-2;JMSConsumer_EJBM;|LDR5010: A
ll ejb(s) of [JMSConsumer_EJBM] loaded successfully!|#]

[#|2008-07-11T10:34:08.937+1000|INFO|sun-appserver9.1|com.stc.jmsjca.core.Activation|_T
hreadID=29;_ThreadName=JMSJCA connect;|JMSJCA-E015: [serial-QueueReceiver(lookup://java
caps/jms/qReceivers) @ [stcms://(domain1stcms1)]]: message delivery initiation was succ
essful.|#]

[#|2008-07-11T10:34:38.093+1000|INFO|sun-appserver9.1|pkg.jcaJMSConsumer.jcaJMSConsumer
|_ThreadID=30;_ThreadName=JMS Async S5;Context=JMSConsumer_EJBM-jcaJMSConsumer-Context;
|Received [this is a message to qReceivers]|#]
```

Figure 20 Observe JMS message being received and logged in server.log

Summary

What we did in this exercise was to create the simplest JCD 5.x-equivalent (sort of) Java Message Driven Bean that was triggered by a message, delivered to a JMS Queue. We also created, ahead of time, the Queue and the JNDI reference to that Queue. The interface to the "receive" method in the Java class was the same as would have been in the 5.x JCD (except for the name of the JMS OTD). The JMS OTD was available and could have been used exactly the same way as in 5.x JCD. The only difference, superficially, is that there is no graphical mapper. Everything needs to be coded by hand in Java source code mode.

Looking at the server through NetBeans IDE one will note that the EJB Module we created, deployed and exercised, is neither a Java CAPS repository-based object, nor the Java CAPS 6 JBI component. It is a Java EE Message Driven Bean.



I may or may not get around to writing on this topic more, for example discussing how to re-implement, using JCA MDBs, some of the EAI patterns I discuss and illustrate with 5.1 JCDs in the Java CAPS Basics book. Suffice it to say that these Java CAPS 5.x integration sites which do not wish to use BPEL can move out of the repository-based development and into JCA MDB-based development with much less pain then might at first be expected.