Java CAPS 6
Using JCA, Note 2

Batch Inbound, through Batch Local File to JMS

Michael Czapski, July 2008

Table of Content

1 Introduction....1
2 Create Connection Pool and JNDI Reference....2
3 Create Project Group JCABatchProjects_PG....4
4 Create EJB Module....4
5 Create JCA Message-Driven Bean....5
6 Exercising the JCA MDB....14
7 Summary....16
8 Appendix A....**Error! Bookmark not defined.**

1 Introduction

Java CAPS 6 has the 5.x compatibility infrastructure which allows one to import 5.x projects right into Java CAPS 6, build, deploy and run without changes. One can also develop repository-based projects in Java CAPS 6 – that's the 5.x-style projects. This is the old way of developing Java CAPS solutions – still good and valid.

If one were to decide to not use the old way there is the JBI infrastructure, which allows development of solutions that use BPEL Service Engine, XSLT Service Engine, IEP Service Engine, Java EE Service Engine, etc., and a variety of Binding Components. The implication is that business logic is implemented in BPEL 2.0, which is used to orchestrate other services and resources, including interaction with external systems through Binding Components. This is the new way of developing Java CAPS solutions – 100% compatible with the Open Source OpenESB project since it uses the OpenESB project-developed container and components.

Someone might ask "so what happened to eGate?". "eGate" meaning Java Collaboration Definition-like logic components, eWays and the JMS messaging backbone.

While the facility seems underadvertised/downplayed, Java CAPS 6 provides a number of 5.1 eWay-based JCA Adapters and a moderately easy means of developing JCA Message-Driven Beans that can use these adapters to implement JCD-like logic components and, effectively, eGate-like solutions that do not use BPEL or the JBI infrastructure.

This Note discusses and illustrates the implementation of a JCD-like integration solution that retrieves a file from the local file system and writes its content to a JMS destination. This requirement I have seen and heard of being implemented in 5.x many times by many customers.

The JCA Message-Driven Bean, the piece of JCD-like Java logic, will be triggered by a Batch Inbound Adapter (what one would have called the Batch Inbound eWay in 5.1), will read the content of the file using the Batch Local File Adapter (eWay) and will write the payload as a string to a JMS destination. The batch Inbound Adapter will be configured to use a regular expression to match the name of the file. Once it finds the file it will rename the file by prepending the GUID to the name and will pass the new name, the original name and the directory path to the Java code. This is exactly what the 5.1 Batch Inbound does. The JCA MDB will use the new name, the original name and the directory path to dynamically configure the Batch Local File Adapter to retrieve the file content and rename the file (post transfer) to the original name with some string appended to indicate that the file was processed. This, too, is exactly what one would do in a 5.1 JCD in the same circumstance. Once the payload is available the JCA MDB will use the JMS OTD to send it, as a TextMessage, to a JMS Queue. Again, this is something that a 5.x JCD would do.

In effect, this Note describes and illustrates the process of re-creating a 5.x Java Collaboration Definition using Java CAPS 6, but instead of using the repository-based approach it is using JCA MDBs and JCA Adapters.

2 Create Connection Pool and JNDI Reference

Before one can use the Batch Inbound and Batch Local File JCA Adapters one must create and configure connection pools, one for each distinct directory+file combination and a corresponding JNDI reference. If the Batch Local File Adapter is to be dynamically configured the connection pool used for the Batch Inbound can be re-used since directory and file property values will be set at runtime.

Let's create the connection pool for the Batch adapter. This will be a generic pool used by both the Batch Inbound and the Batch Local File because the Batch Inbound Adapter's configuration is specified at the time the JCA MDB is created and the Batch Local File Adapter we will be using will be configured dynamically from the Java code.

Start the Application Server Admin Console and navigate to Resources> Connectors> Connector Connection Pools. Click the New … button and configure properties for Step 1 of 2 - Name: BatchInbound_generic, Resource Adapter: sun-batch-adapter, Connection Definition: make sure to choose the BatchLocalApplicationConnectionFactory. Figure 2-1 illustrates this configuration.

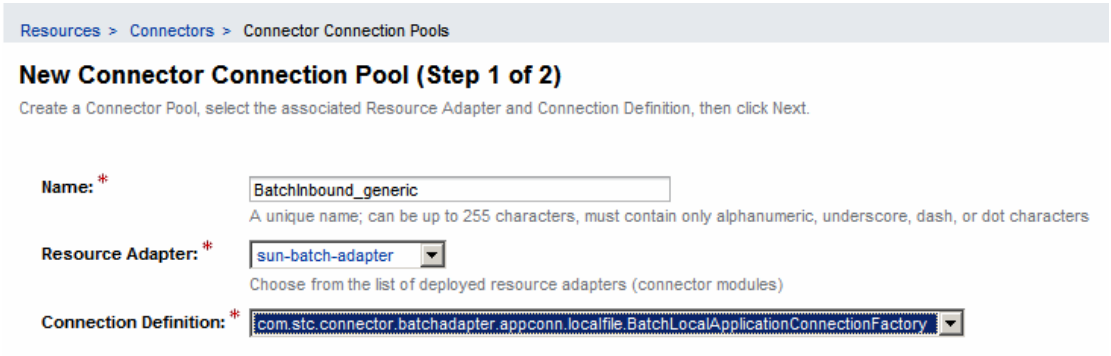

Figure 2-1 Configure pool name and adapter for which it is intended

Click Next and configure properties for Step 2 of 2 – leave all properties as they are. Figure 2-2 illustrates some of the settings for Step 2.

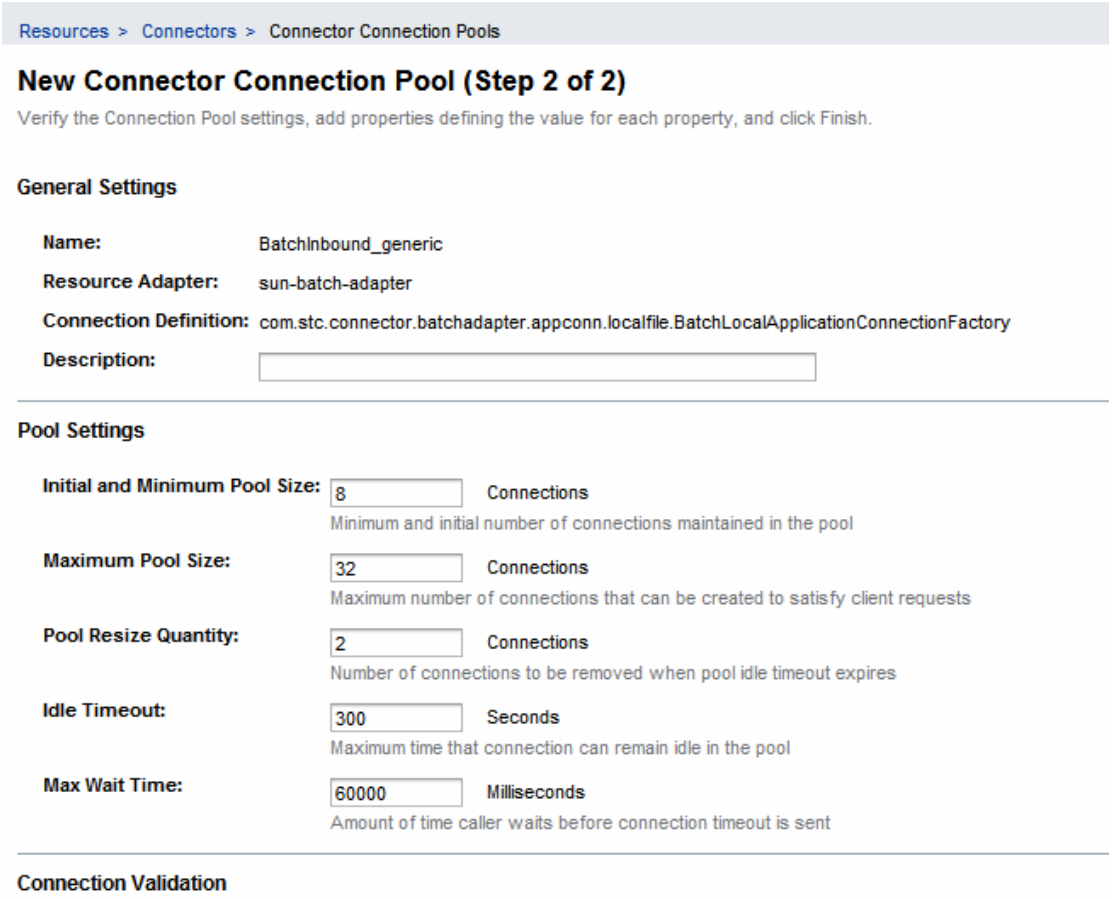

Figure 2-2 Settings for Step 2

Click Finish.

The proceeding steps created a connection pool for the Batch Local Adapter. This pool is good for generic Batch Inbound as well as a generic, dynamically configured Batch Local File. As the pool was created an entry with the same name was added to CAPS> Connector Connection Pools. As previously stated, the Batch Inbound is configured at JCA MDB creation time and the Batch Local File will be configured dynamically so there is no need to do anything to the CAPS> Connector Connection Pool entry. Had we used a static configuration CAPS> Connector Connection Pool entry would be the place to configure static property values like directory, file name, pre- and post-transfer, etc..

The Bach Local File Adapter configuration wizard, used later, will require a JNDI reference to the connection pool we just created. We must create this JNDI reference.

Let’s navigate to Resources> Connectors> Connector Resources and choose New … Let’s name this reference jndiBatchInbound_generic and associate it with the BatchInbound_generic pool we created earlier. Figure 2-3 illustrates this.

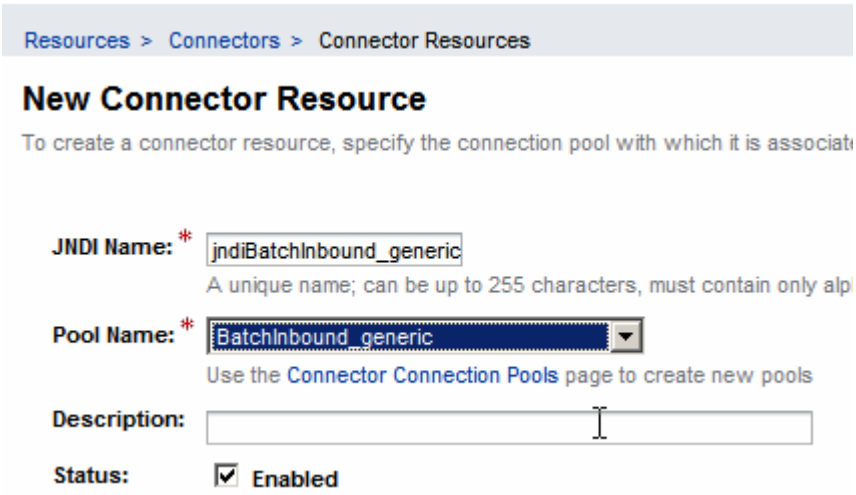

Figure 2-3 Create a JNDI reference to the connection pool

3 Create Project Group JCABatchProjects_PG

As on previous occasions, let's create a new project group to contain projects we will be building in this Note. The group will be called "JCABatchProjects_PG".

Let's right-click anywhere in the Projects tab and choose Project Groups -> New Group … to start the wizard. Figure 3-1 shows the dialog box where project group name and file system directory path are specified.

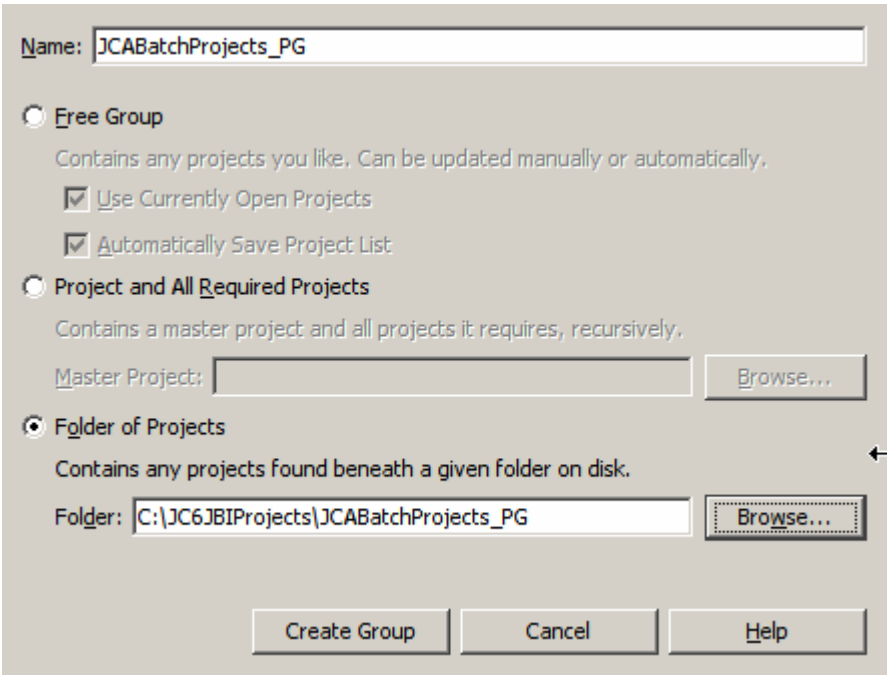

Figure 3-1 Creating a new Project Group

4 Create EJB Module

Create an new Enterprise -> EJB Module project, BInboundThroughBLFToJMS_EJBM, making sure to pick the correct folder for project files. Figure 4-1 illustrates the major step.

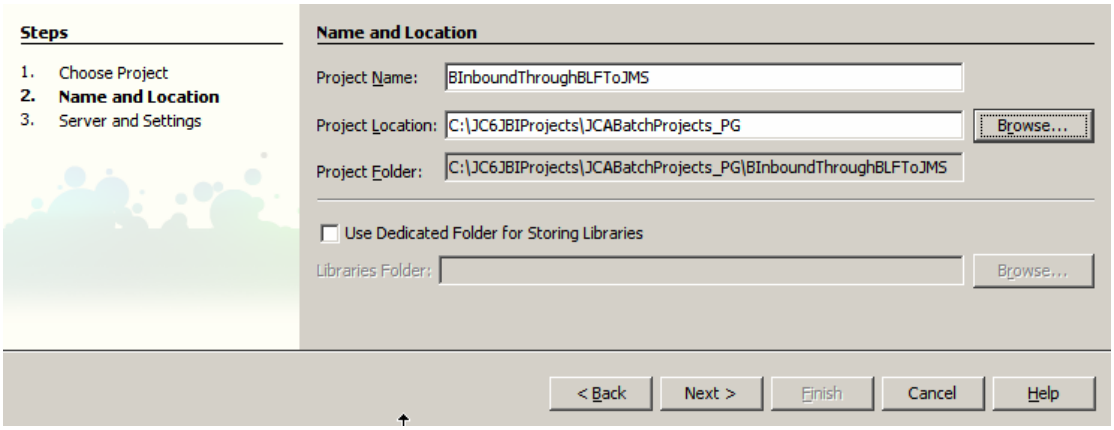

Figure 4-1Naming the project and setting project file location

5 Create JCA Message-Driven Bean

Create a new JCA Message Driven Bean, jcaBInboundThroughBLFToJMS. Figures 5-1 and 5-2 illustrate the initial steps.

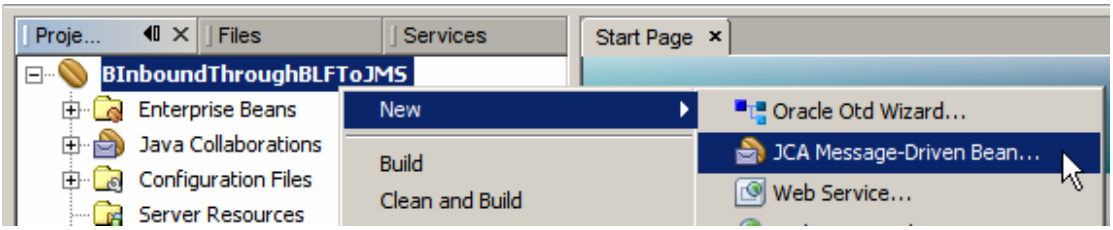

Figure 5-1 Choose JCA Message-Driven Bean

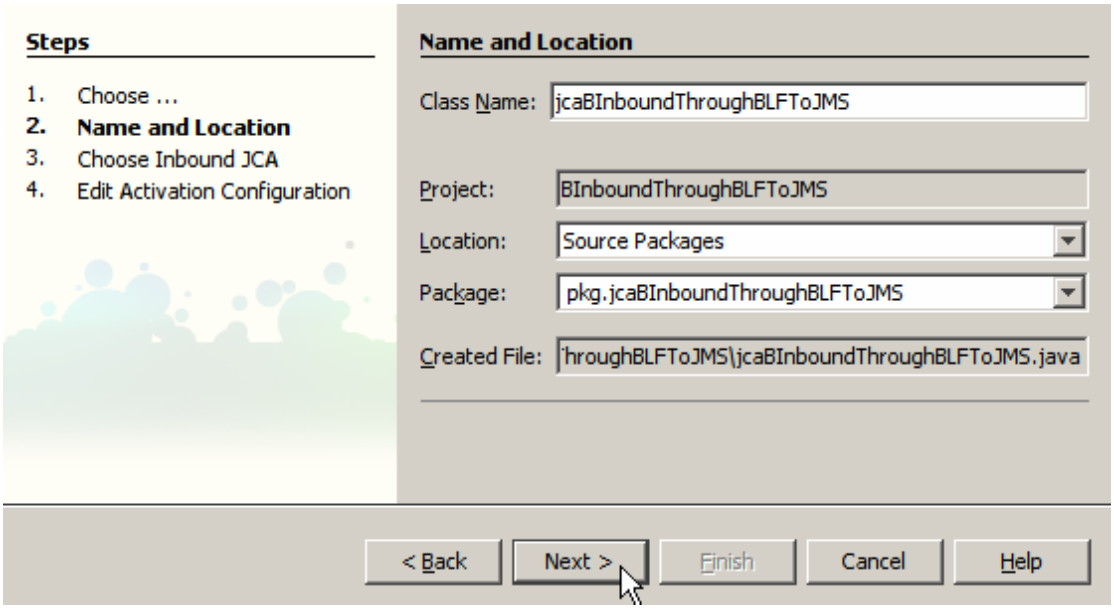

Figure 5-2 Name the bean and the package

Select the Batch JCA Adapter and configure its properties. Click on the ellipsis button at the right of the Configuration field. Figures 5-3 and 5-4 illustrate the steps.

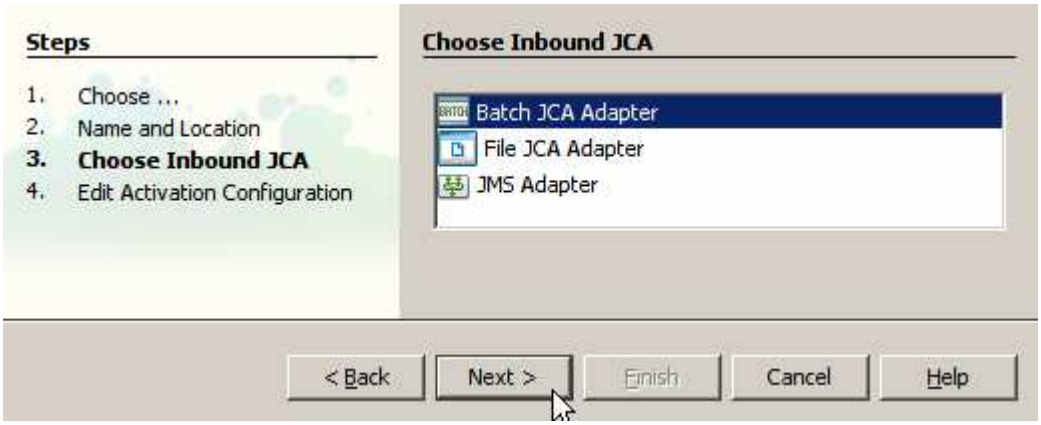

Figure 5-3 Choose Batch JCA Adapter.

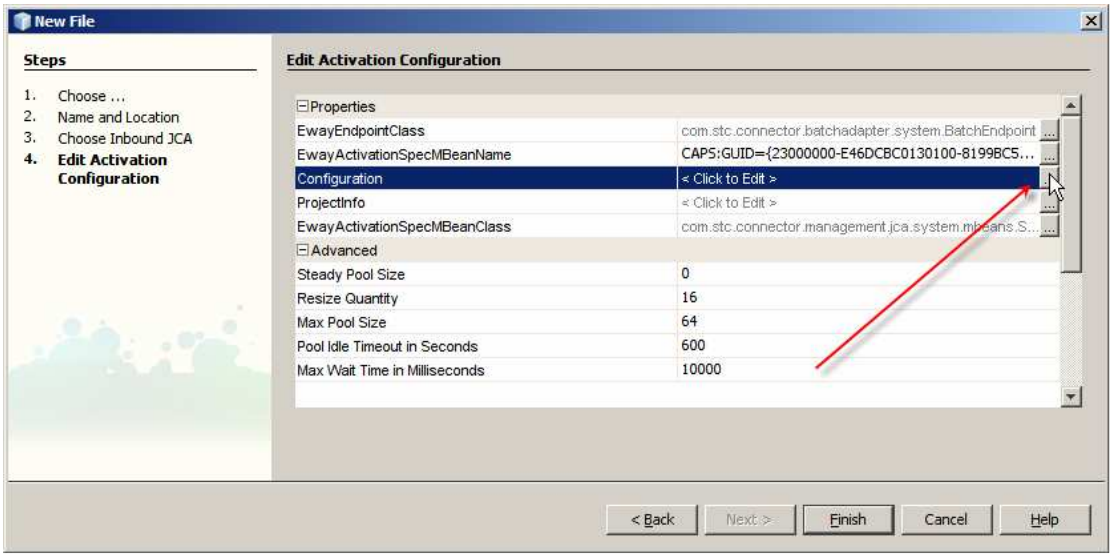

Figure 5-4 Trigger configuration editor

Let's configure the Batch Inbound properties as shown in Figure 5-5, click Close and click Finish.

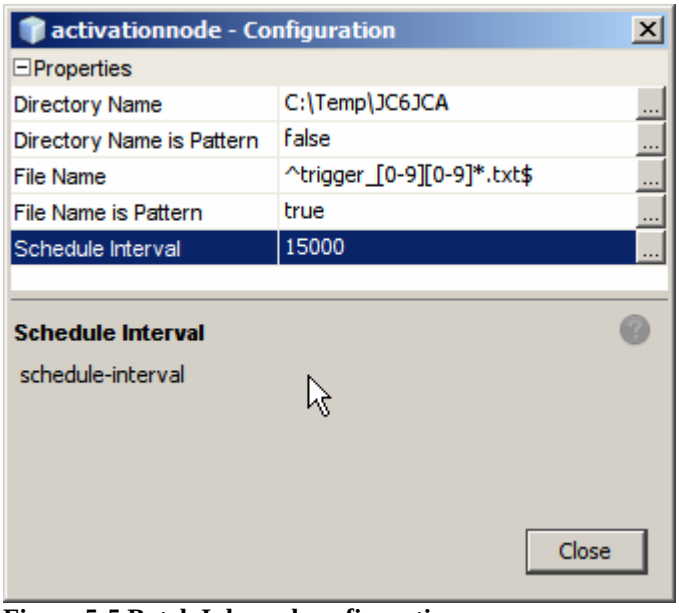

Figure 5-5 Batch Inbound configuration

The Java code shown in Figure 5-6 will appear.

```
package pkg.jcaBInboundThroughBLFToJMS;


import javax.ejb.MessageDriven;
import com.stc.connector.batchadapter.appconn.BatchAppconnMessage;
import com.stc.connector.batchadapter.system.BatchListener;

/**
 *
 * @author mczapski
 */
@MessageDriven(name="pkg.jcaBInboundThroughBLFToJMS.jcaBInboundThroughBLFToJMS")
public class jcaBInboundThroughBLFToJMS implements BatchListener {

    public jcaBInboundThroughBLFToJMS() {
    }

    public void onBatchFileList(BatchAppconnMessage data) throws Exception {
        // implement listener interface here

    }
}
```

Figure 5-6 JCA MDB template source

The Batch Inbound configuration can be modified through the right-click menu off the Java Collaborations node under the EJB Module project tree.

The skeleton MDB needs business logic to do something useful. The next step in the process is addition of the Batch Local File JCA Adapter invocation.

Let’s drag the Batch JCA icon from the palette to the source code window inside the receive method, as illustrated in Figure 5-7, choose the Batch Local File OTD, as illustrated in Figure 5-8, invent and enter a method name and choose the JNDI reference to the connection pool created at the beginning of the process in section 2, as illustrated in Figure 5-9. Note that when choosing a JNDI reference it mat take some time for the JNDI ‘tree’ to appear in the dialog box. There is no indication that work is going on in the background. The dialogue box appears like that shown in Figure 5-10. After a while it will change to look similar to that shown in Figure 5-11, which indicates that the resource tree is ready to be expanded and pool reference can be chosen. Until the usability fix is available just be patient.

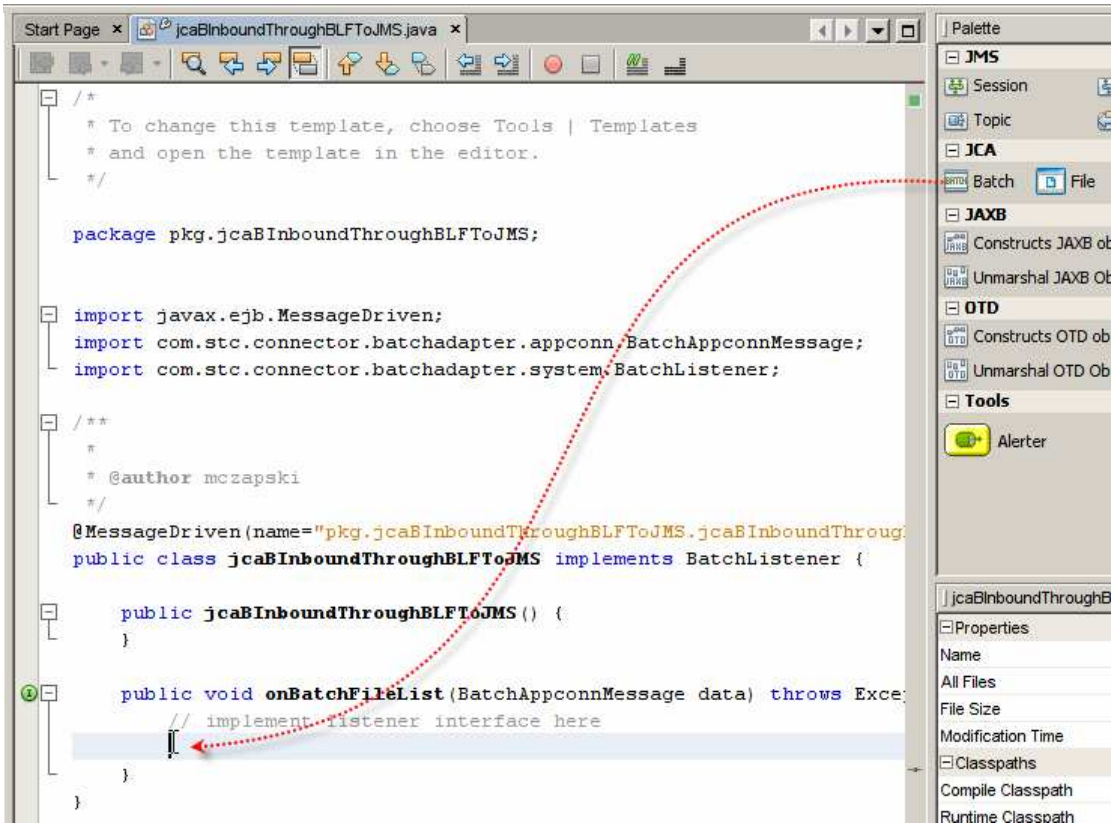

Figure 5-7 Adding Batch JCA to the MDB

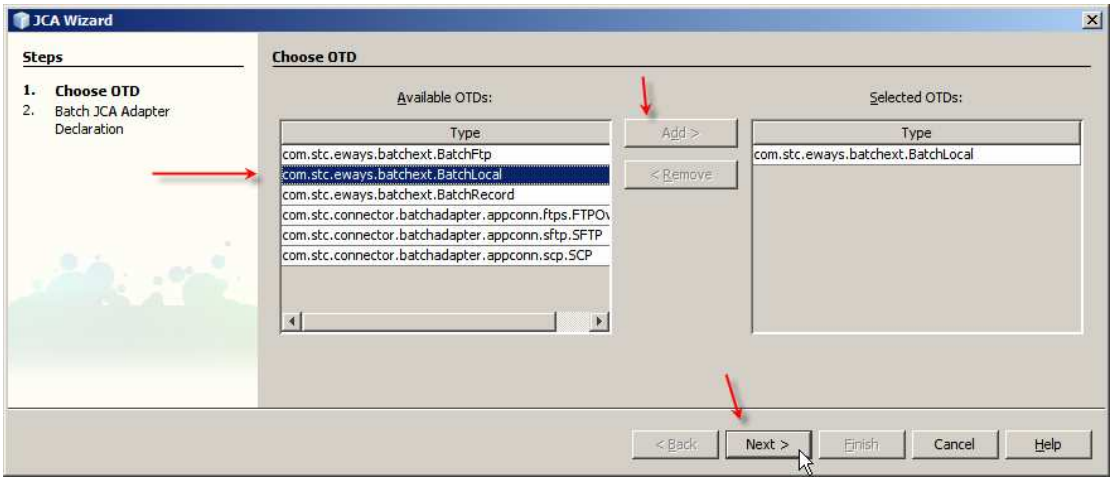

Figure 5-8 Choosing the Batch Local File OTD

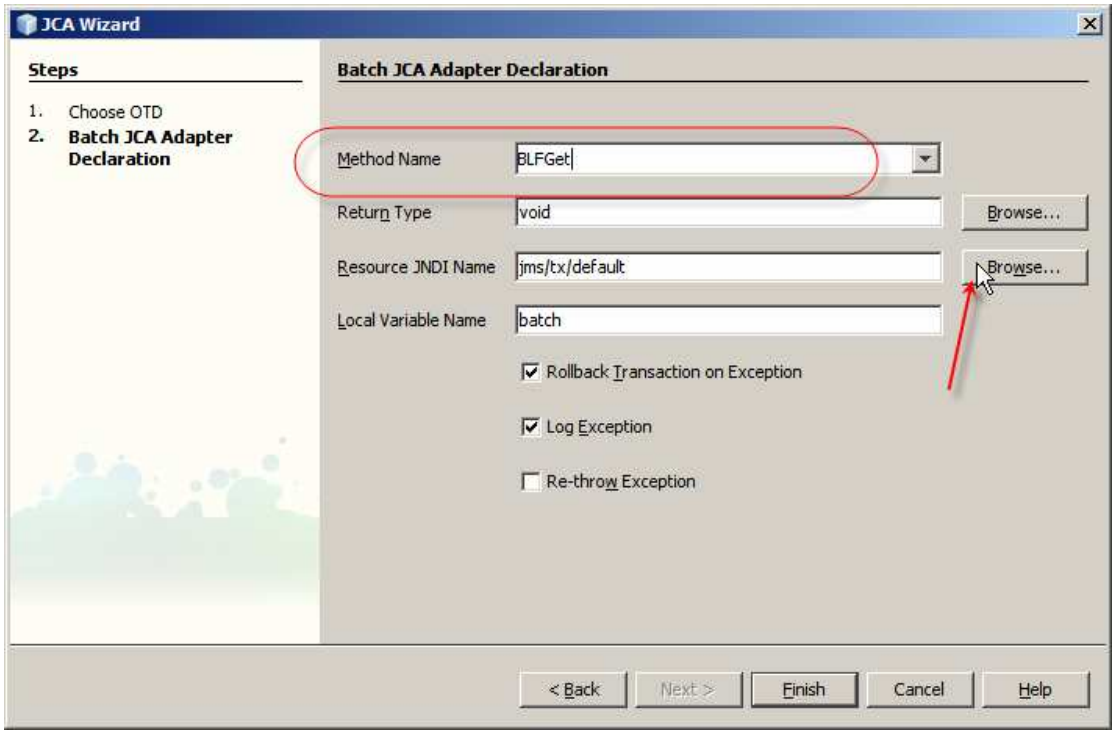

Figure 5-9 Invent the method name and start the process of choosing JNDI Reference

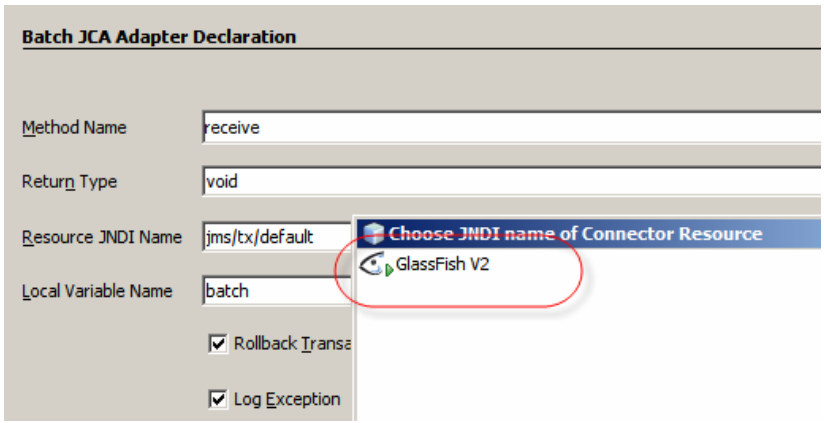

Figure 5-10 Waiting for the JNDI resource list to be assembled

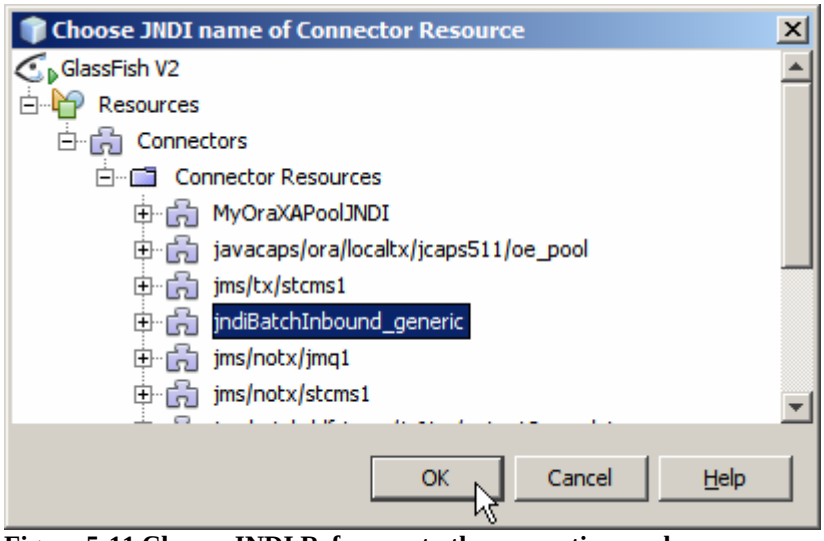

Figure 5-11 Choose JNDI Reference to the connection pool

Slabs of boilerplate code are added to the Java source, mostly in the wrong places for what we need to do. Figure 5-12 shows some of that code.

Figure 5-12 Boilerplate code added by the Batch Local File wizard

Take note of the onBatchFileList method – this is the 'onMessage' method that is invoked when a message is delivered to the MDB. The BatchAppconnMessage parameter named "data" provides access to the Batch Inbound fields, much as the "input" message in a Batch Inbound-triggered JCD would. Figure 5-13 illustrates this.

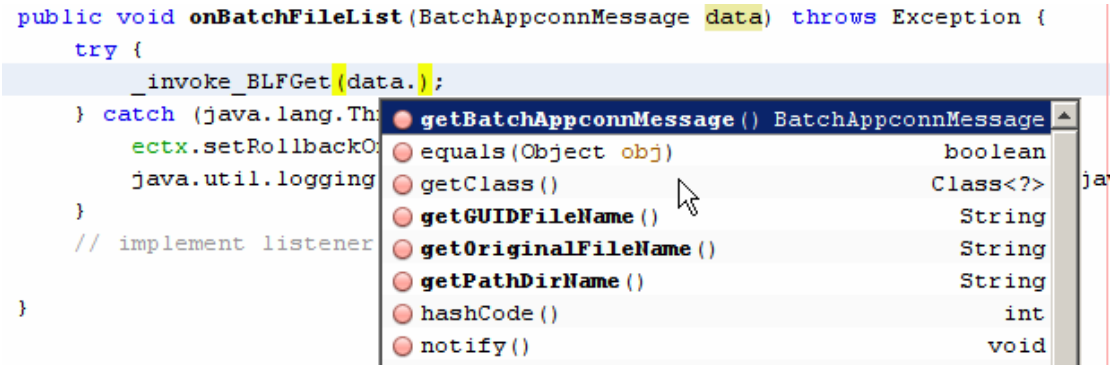

Figure 5-13 Batch Inbound message fields

Within the onBatchFileList method notice the _invoke_BLFGet(data) mthod invocation. Recall that BLFGet is the method name we provided to the wizard when

adding the Batch Local File to the code. This method, see Figure 5-12, gives us a "connected" batchOTD and invokes the BLFGet method with the Batch Inbound message, data, and the batchOTD as parameters. Our "creative code" will go into that method.

To make it easier to relate what we are doing to a JCD in 5.1 let's rename the parameters to the BLFGet to "input" and "G_BatchLocalFile", where "input" is the name that Java CAPS 5.x gives and "G_BatchLocalFile" is the name I use as a convention.

The method signature now looks like that shown in Figure 5-14.

```
    private void BLFGet
            (BatchAppconnMessage input
            ,com.stc.eways.batchext.BatchLocal G_BatchLocalFile)
                throws java.lang.Exception {

    }
```

Figure 5-14 BLFGet method signature after parameter name changes

The action will be in the BLFGet method, which receives the Batch Inbound "input" message and the Batch Local File "G_BatchLocalFile".

As we would have done in the 5.1 JCD we can use the batch Inbound message fields to get access to the original file name, the GUID file name and the directory path of the file that Batch Inbound found. Figure 5-15 shows the code involved.

```
    private void BLFGet
            (BatchAppconnMessage input
            ,com.stc.eways.batchext.BatchLocal G_BatchLocalFile)
                throws java.lang.Exception {

        String sGUIDFileName = input.getGUIDFileName();
        String sOrigFileName = input.getOriginalFileName();
        String sPathDirName = input.getPathDirName();


    }
```

Figure 5-15 Getting data from the Batch Inbound message

Configuring the Batch Local File so that it can read the correct file, from the correct directory, and rename it after it is read, is accomplished the same way it would have been done in a 5.x JCD. Figure 5-15 illustrates this. We are using the GUID file name as the name of the file to read and the directory name prvided by the Batch Inbound as the directory where the file resides. The original file name, with the literal ".~in" appended is the used as the name to which to rename the input file once it is read. Finally, once the configuration is populated, we execute the get() method of the Batch Local File OTD to bet the payload.

```
String sGUIDFileName = input.getGUIDFileName();
String sOrigFileName = input.getOriginalFileName();
String sPathDirName = input.getPathDirName();

G_BatchLocalFile.getConfiguration().setTargetDirectoryName(sPathDirName);
G_BatchLocalFile.getConfiguration().setTargetDirectoryNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setTargetFileName(sGUIDFileName);
G_BatchLocalFile.getConfiguration().setTargetFileNameIsPattern(false);

G_BatchLocalFile.getConfiguration().setPostDirectoryName(sPathDirName);
G_BatchLocalFile.getConfiguration().setPostDirectoryNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setPostFileName(sOrigFileName + ".~in");
G_BatchLocalFile.getConfiguration().setPostFileNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setPostTransferCommand("Rename");

G_BatchLocalFile.getClient().get();
```

Figure 5-16 Dynamically configuring the Batch Local File and getting the payload

As stated at the beginning, the intention was to get the content of the file and write it, as a TextMessage, to a JMS Queue.

Let's drag the JMS OTD from the JMS part of the palette to the source windows, following the G_BatchLocalFile.getClient().get(); statement, as illustrated in Figure 5-17.

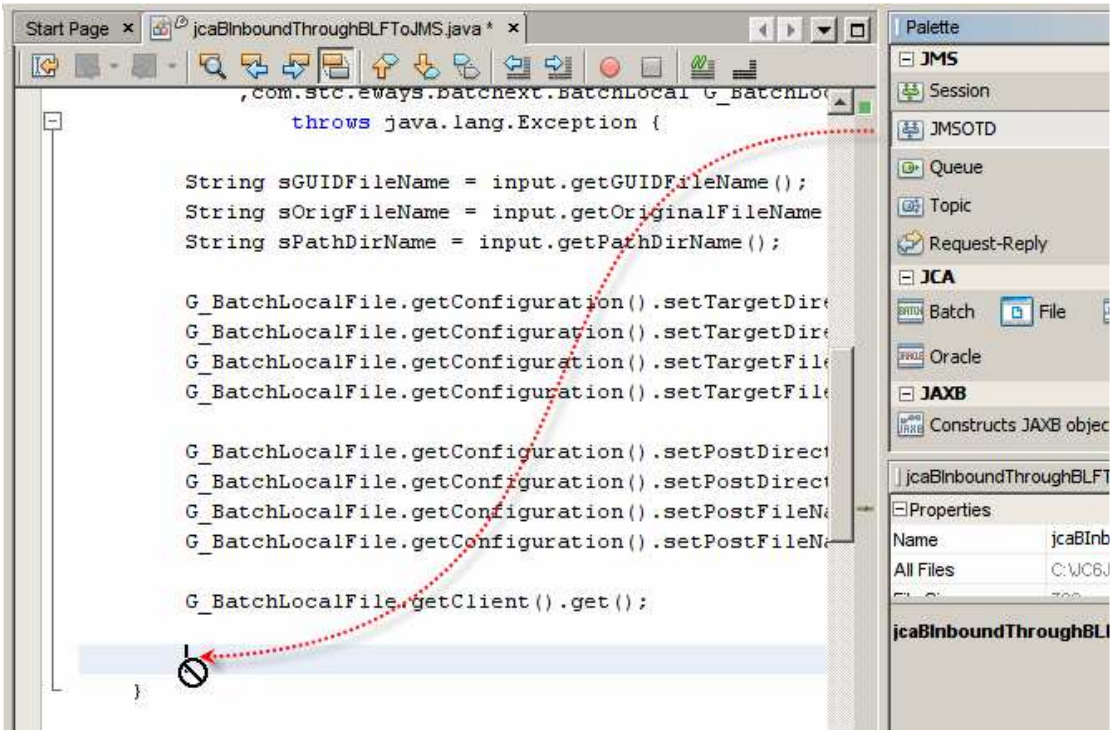

Figure 5-17 Adding JMS OTD

The JMS OTD Wizard, which pops up, allows us to configure the JMS OTD to use the correct JMS Destination type and the correct JMS Destination name. Figure 5-18 illustrates this.

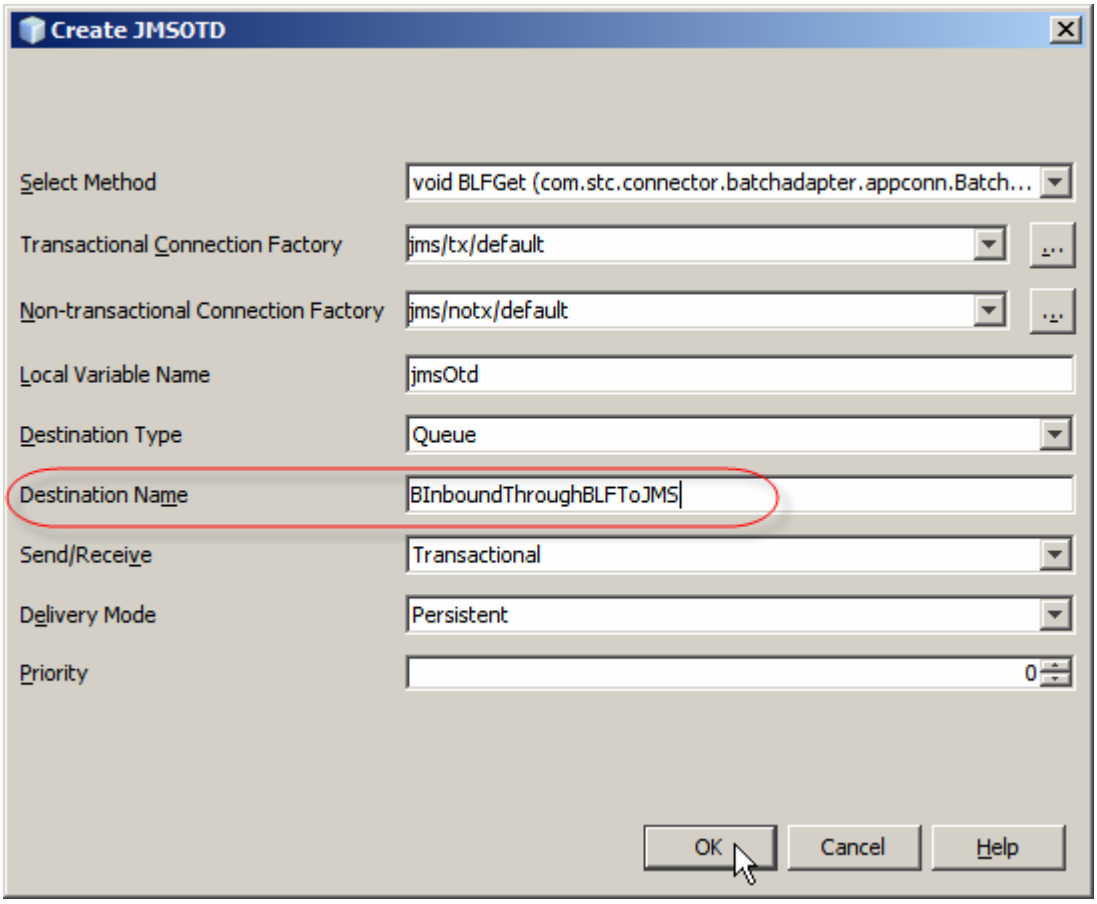

Figure 5-18 Configuring the JMS Destination

Note that transactionality and persistence can be configured through this wizard as well.

The wizard adds more boilerplate code. Most notably it adds the reference to the JMS OTD, named jmsOtd, right at the place where we dragged the JMS OTD object. Figure 5-19 illustrates the code fragment, reformatted for readability.

```
G_BatchLocalFile.getClient().get();

com.stc.connectors.jms.JMS jmsOtd =
        com.stc.connectors.jms.JMS.createInstance
            (_jms_connfact
            , _notx_jms_connfact
            , com.stc.connectors.jms.JMS.DestinationType.Queue
            , "BInboundThroughBLFToJMS"
            , true
            , javax.jms.DeliveryMode.PERSISTENT
            , 0);
```

Figure 5-19 JMS OTD reference

Had I been doing this is a JCD in 5.x I would have named the OTD variable W_ToJMS. To keep with that convention, which would have helped me a great deal if I had to transcribe a 5.1 JCD to the 6 JCA MDB, I will rename the variable from jmsOtd to W_ToJMS.

Com.stc.connectors.jms.JMS is the same object that is used in a 5.1 JCD to publish/send to JMS.

Let's take the payload read by the Batch Local File, convert it to String and send it to JMS as a textMessage. Figure 5-20 illustrates the code.

```
com.stc.connectors.jms.JMS W_ToJMS =
        com.stc.connectors.jms.JMS.createInstance
            (_jms_connfact
            , _notx_jms_connfact
            , com.stc.connectors.jms.JMS.DestinationType.Queue
            , "BInboundThroughBLFToJMS"
            , true
            , javax.jms.DeliveryMode.PERSISTENT
            , 0);

W_ToJMS.sendText(new String(G_BatchLocalFile.getClient().getPayload()));
```

Figure 5-20 Sending file content to a JMS destination.

This is all that is required for a JCA MDB to be triggered by a Batch Inbound Adapter, use the Batch Local File Adapter to read the content of a file and to send it ti a JMS Destination as a TextMessage. The final code is almost identical to that one would have in a 5.x JCD to accomplish the same thing.

Let's build and deploy this MDB.

6 Exercising the JCA MDB

When the project is deployed the server.log, when appropriate logging level is enabled for the appropriate logging category, will show text similar to what is shown in Figure 6-1. Not the regular expression pattern, we specified for the Batch Inbound.

```
[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.syste
m.BatchInboundWork|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.connector.logging.JD
KLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|Checking for files...|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.util.
BatchInboundFileUtil$FileNameFilter|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.con
nector.logging.JDKLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|<init>: Create
d a RegEx with the string ^trigger_[0-9][0-9]*.txt$|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.util.
BatchInboundFileUtil$FileNameFilter|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.con
nector.logging.JDKLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|.accept: It is
 not a match batch_inbound_00.txt.~in|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.util.
BatchInboundFileUtil$FileNameFilter|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.con
nector.logging.JDKLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|.accept: It is
 not a match output2_14.dat|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.util.
BatchInboundFileUtil$FileNameFilter|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.con
nector.logging.JDKLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|.accept: It is
 not a match trigger_0.txt.~in|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.util.
BatchInboundFileUtil|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.connector.logging.
JDKLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|BATCH-DBG-D0405: getFirstFile
Name: Checking for file match in directory C:\Temp\JC6JCA|#]

[#|2008-07-16T18:33:41.765+1000|FINE|sun-appserver9.1|STC.eWay.batch.com.stc.connector.batchadapter.syste
m.BatchInboundWork|_ThreadID=16;_ThreadName=p: thread-pool-1; w: 3;ClassName=com.stc.connector.logging.JD
KLogger;MethodName=debug;_RequestID=b0381df2-fdfd-4143-94fe-d71ac0f83170;|BATCH-DBG-D0399: "Directory Nam
e", "File Name" or "File Name is Pattern" is found no match|#]
```

Figure 6-1 server.log trace from batch Inbound poll

Let's create a file, with the name of trigger_0.txt.~in, with some content. Once the file is ready, let's change its name to trigger_0.txt. Within at most 15 seconds, which is the polling interval we configured for the Batch Inbound, the file will be picked up, renamed by prefixing the GUID to it, read, renamed to trigger_0.txt.~in and a JMS message will be written to the JMS Queue BInboundThroughBLFToJMS.

Figure 6-2 shows the message in the queue using the Enterprise Manager Web interface.

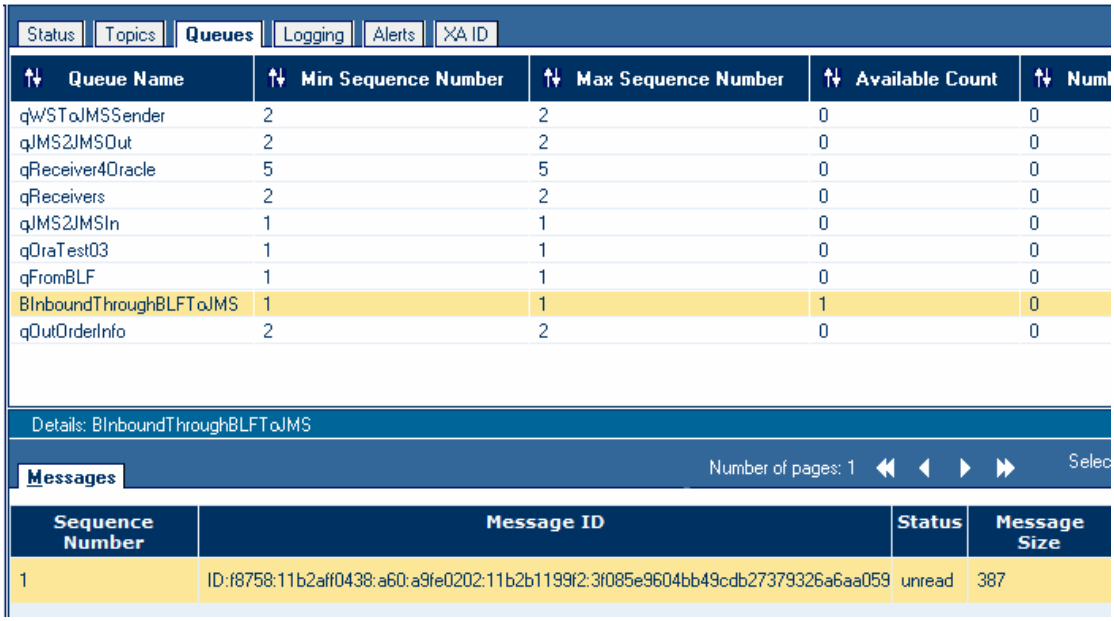

Figure 6-2 Message in the BInboundThroughBLFToJMS queue

7 Summary

This document walked the reader, step-by-step, through the process of creating and exercising a Java CAPS 6 JCA-based solution that used the Batch Inbound, Batch Local File and JMS JCA Adapters to implement an eGate-like Java logic. The actual creative code, added to the Java source once the JCA wizards were finished with it, was very much the same as what one would have added to the 5.x JCD to accomplish the same tasks. We even gave the instance variables names that would have been used in JCD to make the similarity even greater.