Java CAPS 6, JCA, Note 4
Streaming FTP Inbound

Michael Czapski, July 2008

Table of Content

1 Introduction

As at now there does not seem to be a way to stream FTP payloads to the local file system, or stream local files to FTP servers in the JBI environment. This kind of functionality may or may not appear in the OpenESB project, perhaps in conjunctions with the FTP BC or the File BC or both. In the meantime, a Java CAPS 6 developer can use the JCA Adapters-based solution to stream payloads of arbitrary size between FTP servers and local file systems in either direction.

This Note walks through the implementation of a Batch FTP JCA solution, which is triggered by a JMS Message, and performs a streaming FTP transfer of an arbitrarily large payload between a remote FTP server and the local file system. This Note re-implements the inbound part of the "Java CAPS 5.1 and Java CAPS 6 - Streaming Large FTP Transfers" Note available at http://blogs.sun.com/javacapsfieldtech/entry/streaming_large_ftp_transfers_with.

It should be easy enough to re-implement the outbound part of that Note using the material in this Note.

2 Create a Project Group JCABatchProjects_PG

The project group JCABatchProjects_PG may already exist if you implemented Note 3 or Note 4 in this series. If not, create a new Project Group as illustrated in Figures 2-1 and 2-2.

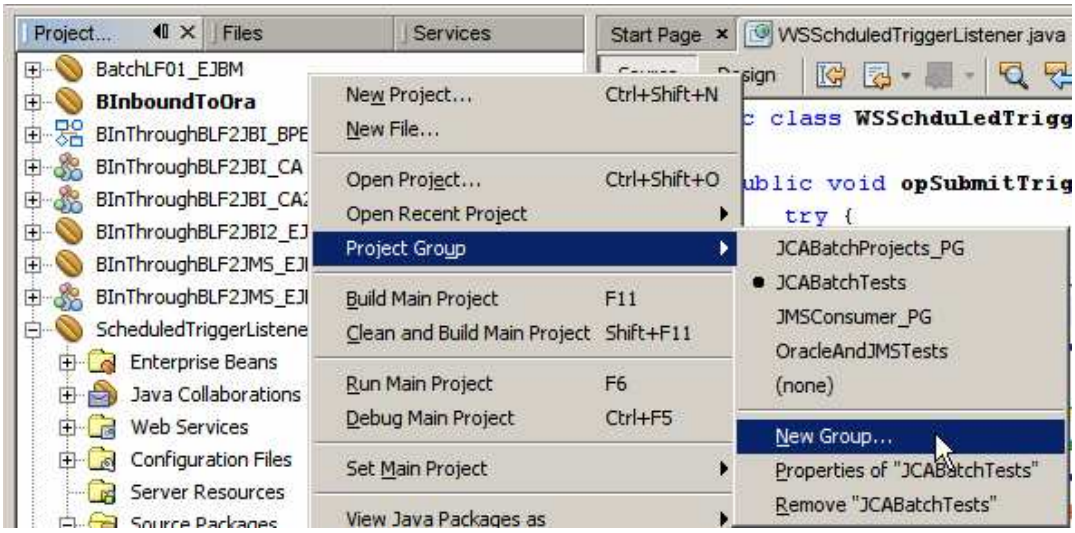

Figure 2-1 Trigger a New Project Group wizard

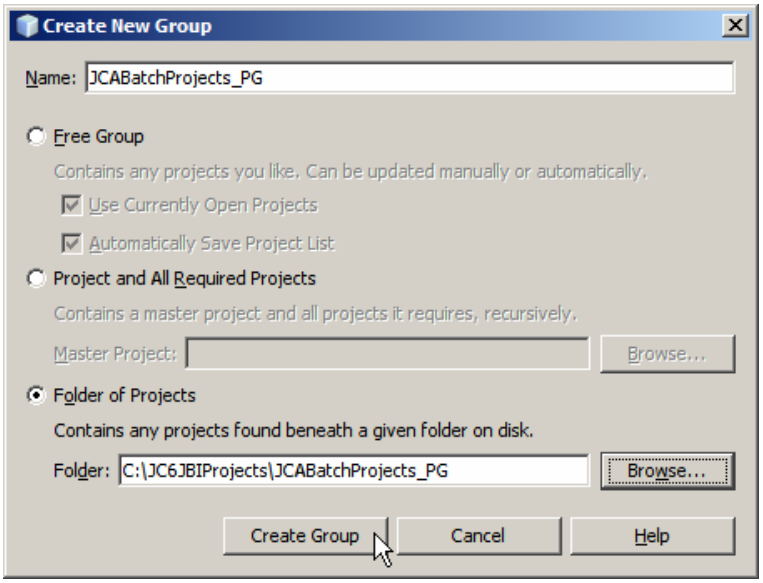

Figure 2-2 Name the Project Group and choose the directory to contain project artifacts

If the Project Group artefacts already exist in the nominated directory they will be loaded and will appear in the projects list. This is the case here as shown in Figure 2-3.

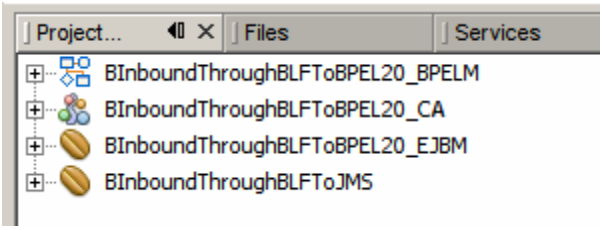

Figure 2-3 Existing projects in the Project Group

3 Scheduling the Solution

It is expected that FTP transfers will be scheduled to occur every now and then, perhaps once a day or more frequently. The scheduling solution that could be used is discussed in “Java CAPS 6, Scheduler for JCA and JBI Projects, Note”, at

http://blogs.sun.com/javacapsfieldtech/entry/java_caps_6_scheduler_for. The scheduling solution discussed there can be used to trigger the solution discussed in this Note.

It is assumed in this Note that the scheduling of the FTP transfer will occur as a result of a trigger message arriving to the JMS Queue, qSchedulerTrigger, when appropriate.

Scheduling will not be discussed further in this Note.

4 Configuring Connector and JNDI Resources

We will be using a Batch FTP and a Batch Local File JCA Adapters. Unlike in 5.1, where these things were configured through Connectivity Maps, we need to create Connector Pool resources, JNDI references and adapter configurations using the Application Server Administration Console.

So as to be able to correctly name the resources let's spend a little time on the relationship different resources and how they relate to individual connections between the integration solution and the external systems.

For a Batch Adapter we need one of Resources -> Connectors -> Connector Connections Pools pool (which defined pool size, connector default configuration and other properties which are specific to the connection pool) and one of Resources -> Connectors -> Connector Resources resources (which associates the JNDI Name with the specific Connector Connection Pool). Figure 4-1 calls out the nodes in the hierarchy.

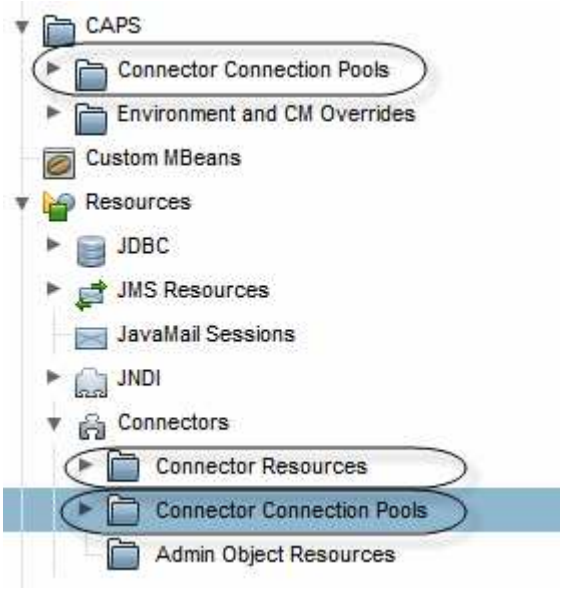

Figure 4-1 Resource hierarchy

Notice, from the statement above, that there is a one-to-one correspondence between the pool and the JNDI reference to it, and that so far we have nowhere to configure the connector properties, like directory or file name. Notice also that we need one of the CAPS -> Connector Connection Pools pools but we will not actually create it. It will be created automatically when we create the Resources -> Connectors -> Connector Connections Pools pool.

Before providing more explanations let's create a Resources -> Connectors -> Connector Connections Pools pool for the Batch FTP Adapter, call it cp-batch-ftp-localhost-mczapski, see Figure 4-2 through 4-7 for major steps involved in creating and configuring the pool.

Figure 4-2 Start the New Connector Connection Pool configuration page set

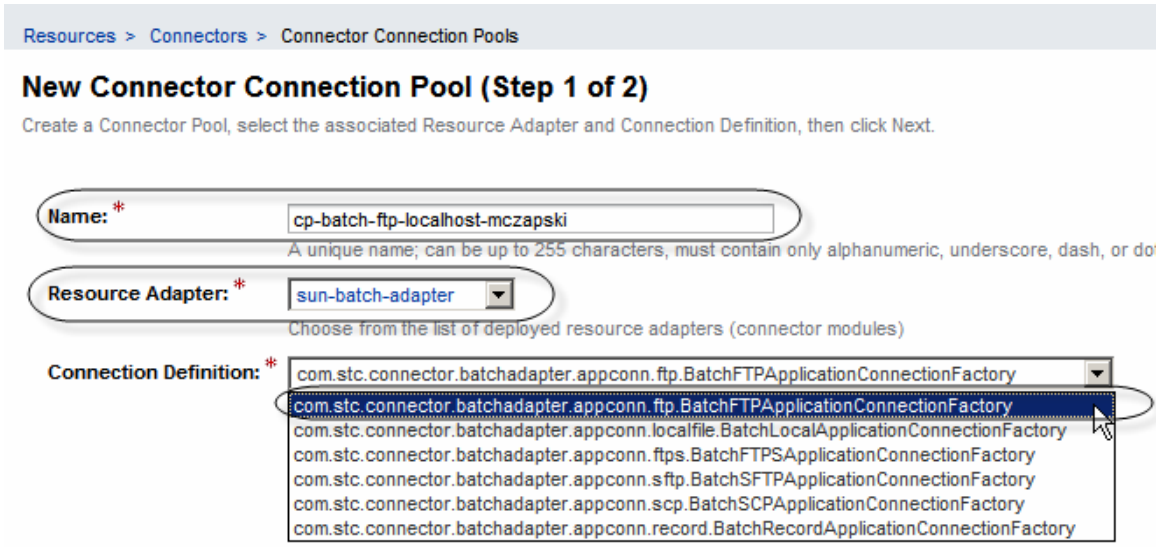

Figure 4-3 Name the pool, choose the adapter and choose the connection factory

Notice, in Figure 4-3, that there are a number of connection factories to choose from. They correspond to the gross functionality variant of the adapter – FTP, FTPS, SFTP, SCP, Local File and Record. All of the functionality is provided by the one Batch JCA Adapter. By choosing the connection factory we are creating a pool for that kind of connection. If we need a Batch FTP and a Batch Local File connections we will have to have one pool for each type, and as we will se later, one pool for each distinct connection to the external system/resource.

The second step in pool creation, Figure 4-4, allows us to configure pool size and timings, and other connection-related and pool-related properties.

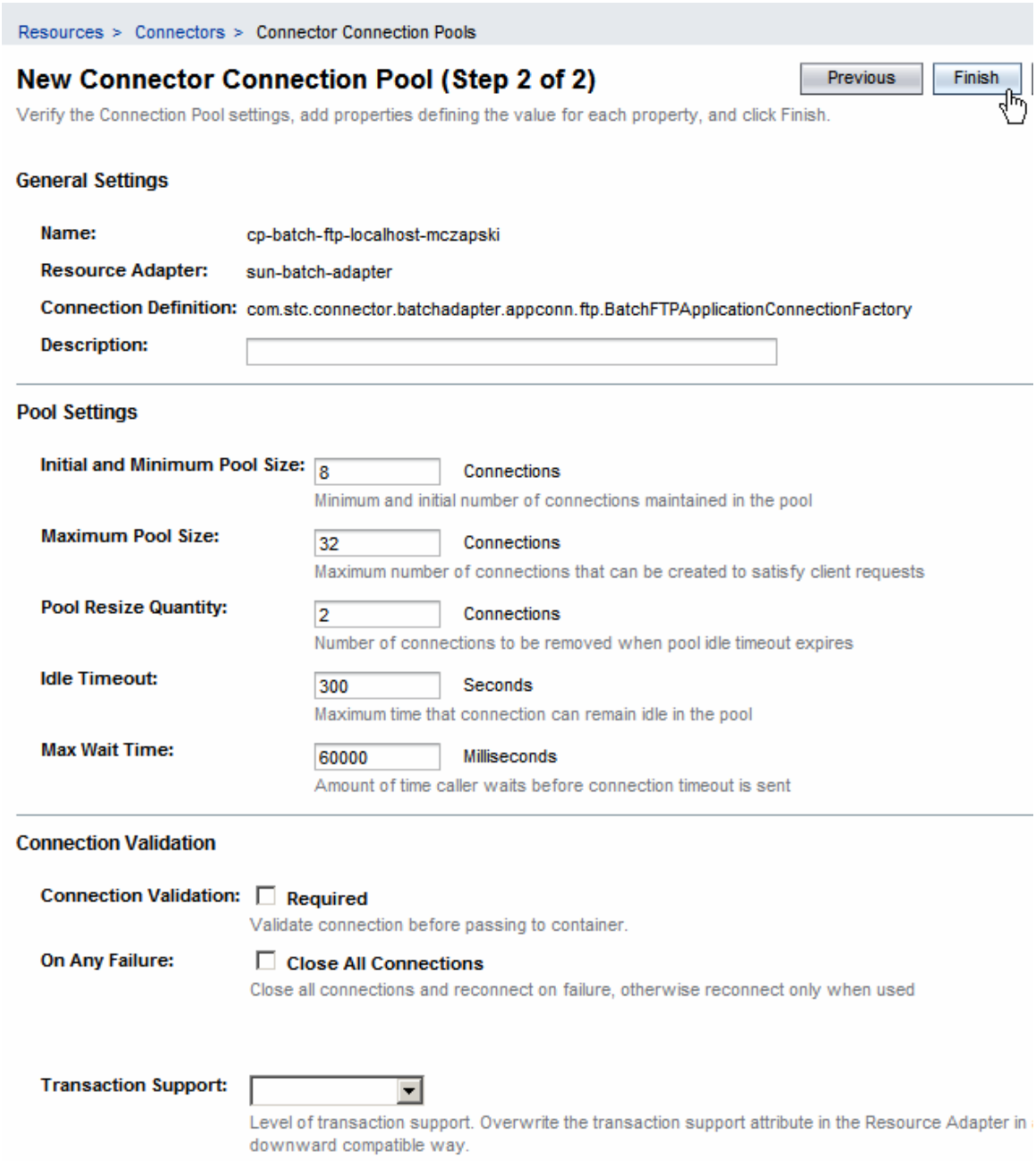

Figure 4-4 Pool-related and Connection-related properties of the connection pool

As we completed creation of the connection pool under the resources node tree a corresponding pool, with the same name, was created under the CAPS node tree. See Figure 4-5.

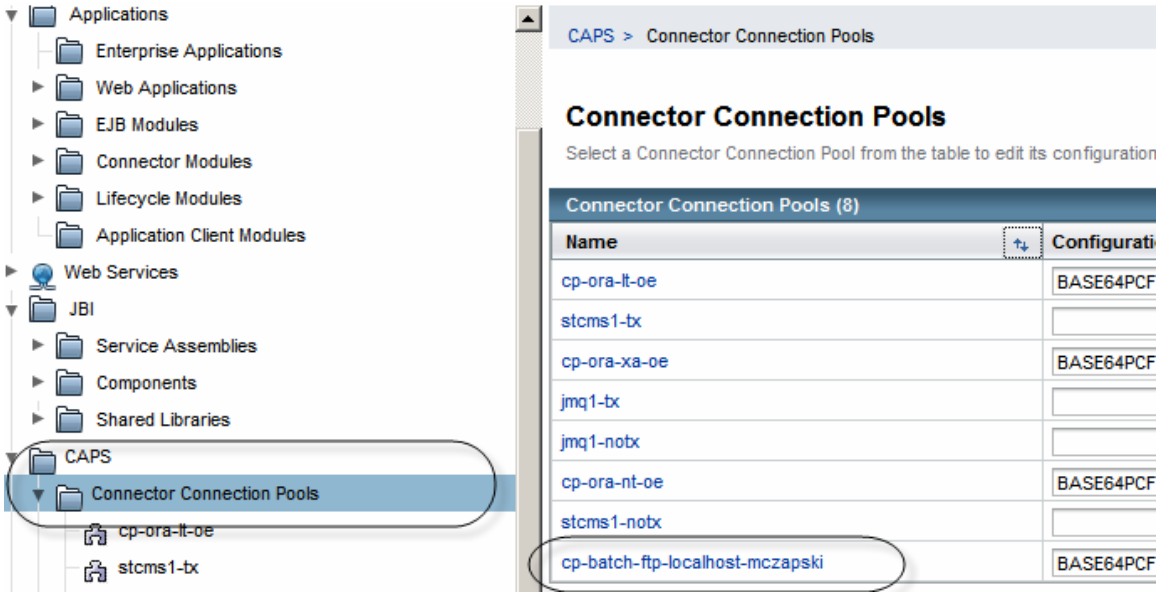

Figure 4-5 Corresponding pool under the CAPS node tree

Let's look at the properties of this pool, Figure 4-6.

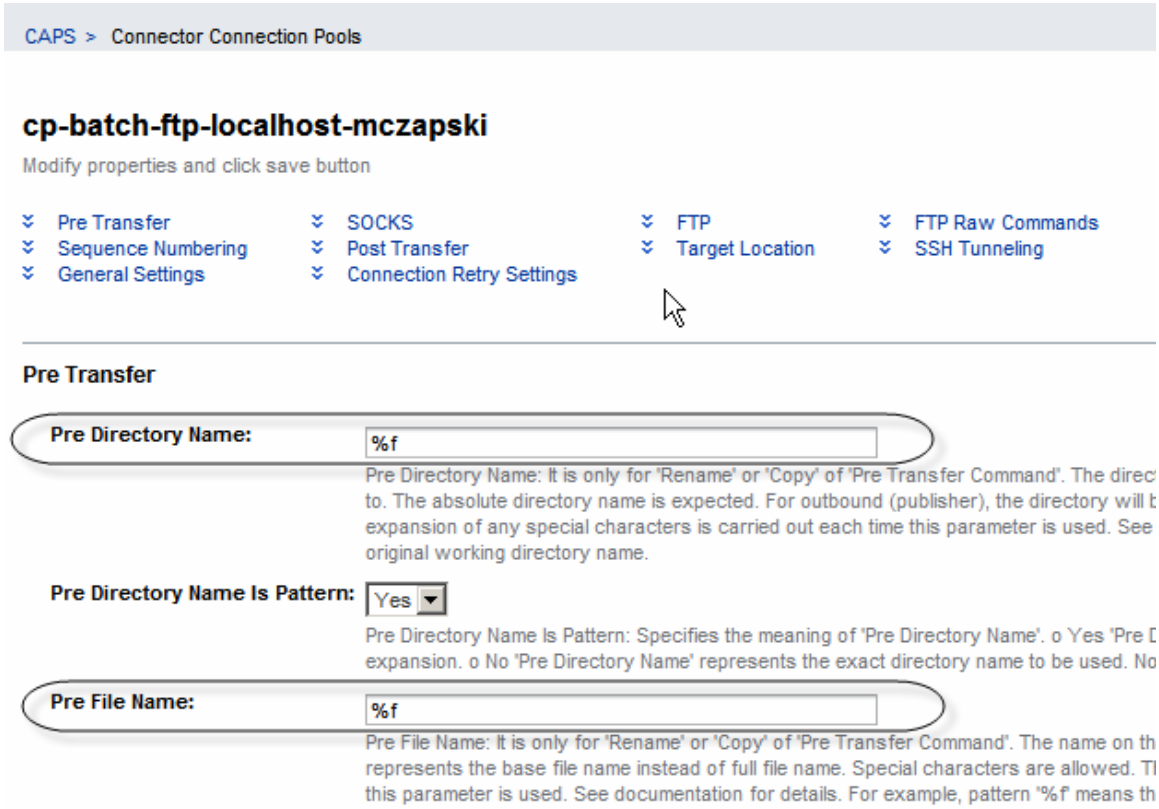

Figure 4-6 Connection pool properties for connection pool under CAPS node tree

Notice that the sole purpose of this pool entry is to allow configuration of the external system – things like directory name and file name for a Batch Adapter, or database URL and credentials for a database Adapter. This pool corresponds in functionality to the combination of Connectivity Map connector properties and External System Container properties in 5.x.

Observe that although the connection pool under the Resources node tree only specified pool characteristics, and the adapter and the connection factory to use, it did not specify the actual external resource which to use. It is the associated pool under the CAPS node tree that provides these properties. The conclusion from this should be that, if the connection is

statically configured to use a specific file and a specific directory then it will have a pool associated with it. There will be a different pool for a different connection of the same adapter type and the same connection factory if the directory or file name is to be statically configured to be different. On the other hand, if logic components dynamically populate properties found in the pool under the CAPS node tree then the same pool can be reused for different directory and file combinations – a generic pool, if you like.

I intend to use the pool called cp-batch-ftp-localhost-mczapski to connect to the FTP Server on the localhost and use credentials associated with user mczapski. By naming the connection pool the way I have I remind myself of what the pool is for – FTP, localhost, mczapski. To be strictly correct I should have included the directory name and the file name in the name of the pool.

If the pool I wanted to create was to be generic (as in configured dynamically by a JCA MDB) I would have called it something like cp-batch-ftp-generic.

This is my convention. You are most welcome to invent your own if you don't like mine ☺

Let's configure the external system properties (Pool under the CAPS node tree) as shown in the table. Your values will vary, particularly for credentials and suchlike. Adapt as appropriate.

Section	**Property**	**Value**
FTP	Host Name	localhost
FTP	Username	mczapski
FTP	Password	********
Post Transfer	Post Transfer Command	Rename
Post Transfer	Post File Name	%f.done
Post Transfer	Post File Name is Pattern	Yes
Target Location	Target Directory name	//
Target Location	Target File Name	jms-1_1-fr-spec.pdf

Figure 4-7 illustrates some of the properties.

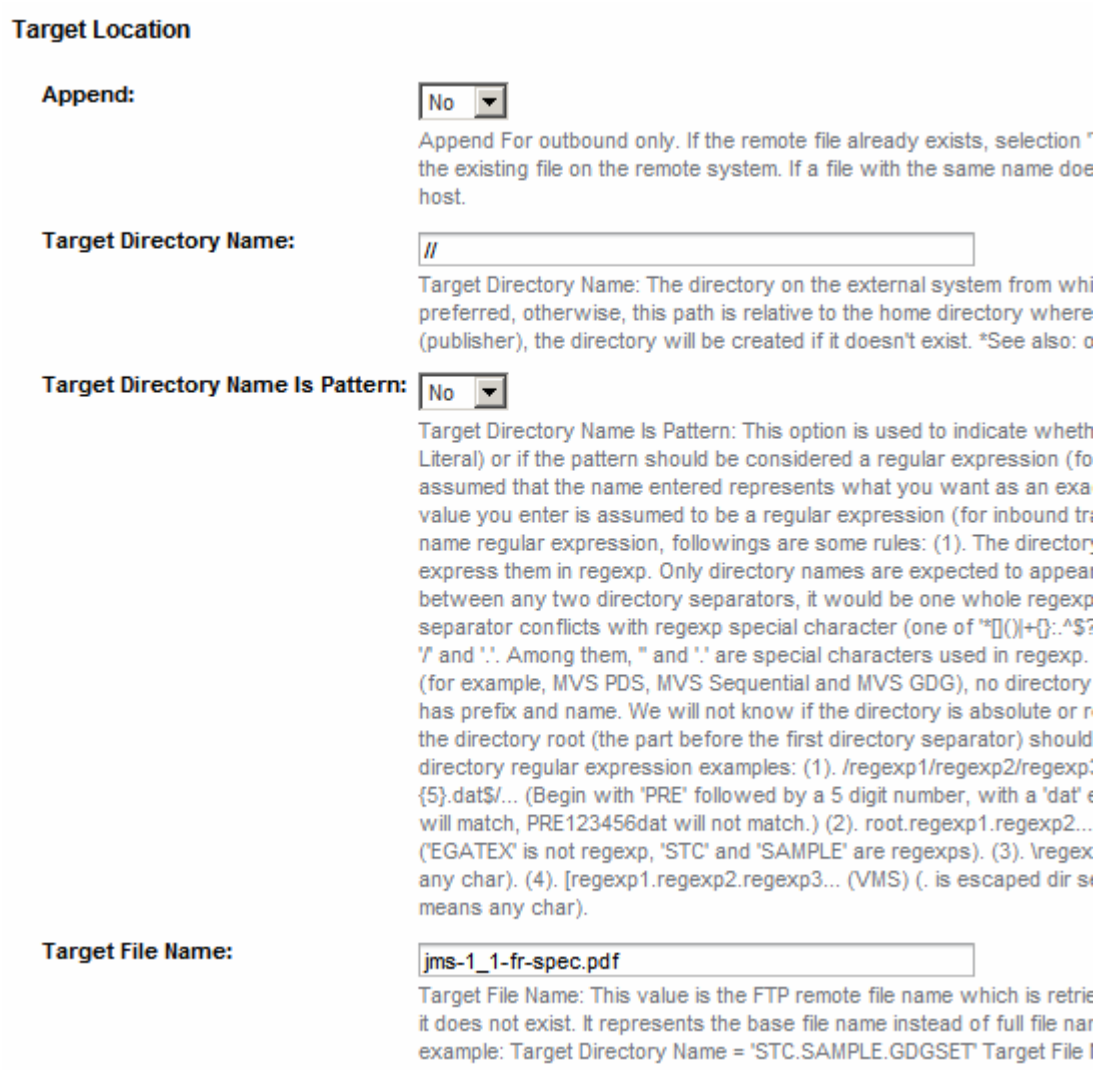

Figure 4-7 Selected connection properties in the pool under the CAPS node tree

Note

> Target Directory Name may be a trap for the young layers. If the file is at the root of the hierarchy one is tempted to specify "/" as the directory name. Use "//" instead, otherwise the file will not be found. Subdirectories, like "/aaaa" work as expected.

Now that we have the pools we must create the JNDI name that can be used to obtain a connection from the pool. I use the name jndi- cp-batch-ftp-localhost-mczapski. Figures 4-8 and 4-9 illustrate this.

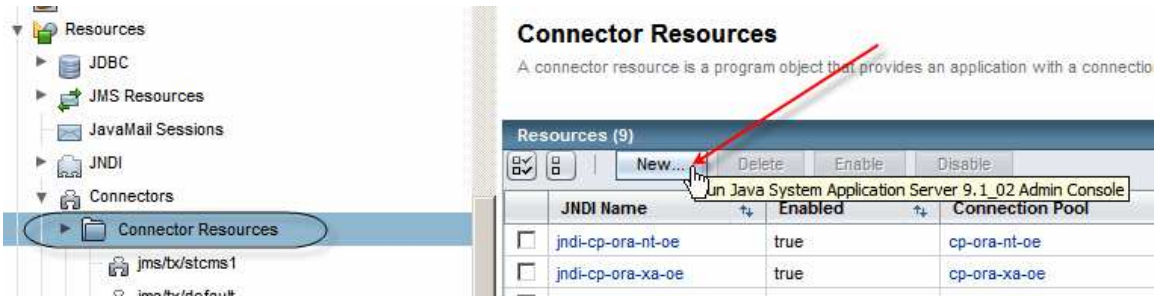

Figure 4-8 Create a new Connector resource

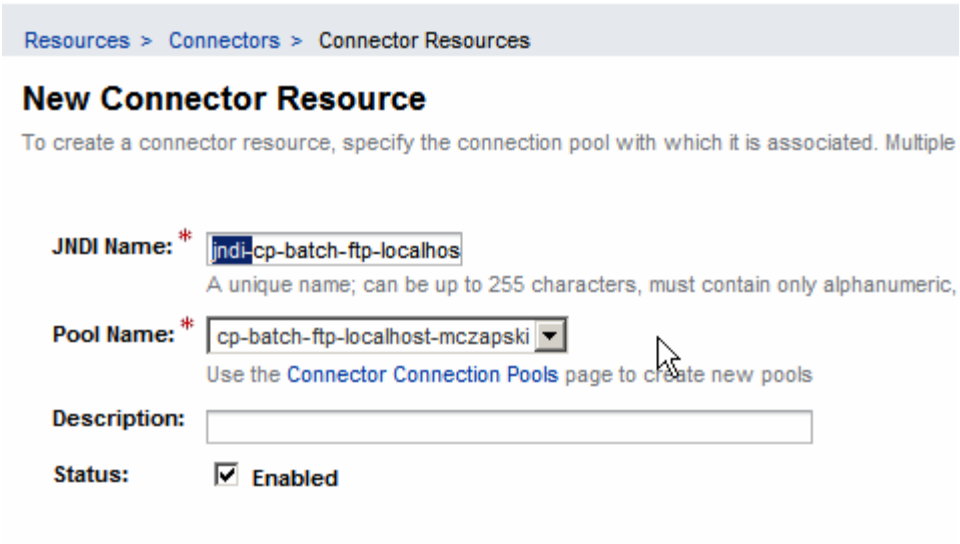

Figure 4-9 Name the JNDI resource and associate it with the pool

Note, in Figure 4-9, that the JNDI name is the name of the connection pool prefixed with the literal “jndi-“, “jndi-cp-batch-ftp-localhost-mczapski”. This, too, is my convention.

So, there is one pool with two parts, in the Resources node tree and in the CAPS node tree, both of which may require us to specify/change property values, and there is a JNDI name associated with the pool.

Much as has been done for the Batch FTP JCA-related pool and JNDI reference, the Batch Local File JCA-related pool and JNDI reference must be created and configured. Table 4-1 lists key properties that have to be set in the various resources. Since the steps will be the same as already described only key illustrations will be shown.

Node Tree	Section	Property name	Property Value
Resources -> Connectors -> Connector Connection Pools:		Name	cp-batch-localfile-c/temp/jc6jca/ftp_output_nn.dat
		Resource Adapter	sun-batch-adapter
		Connection Definition	…BatchLocalApplicationConnectionFactory
Figure			4-10
CAPS -> Connector Connection Pools:	Target Location	Target Directory Name	C:/temp/jc6jca
		Target File Name	ftp_output_%d.dat
		Target File Name is Pattern	Yes
Figure			4-11
Resources -> Connectors -> Connector Resources		JNDI Name	jndi-cp-batch-localfile-c/temp/jc6jca/ftp_output_nn.dat
		Poll Name	cp-batch-localfile-c/temp/jc6jca/ftp_output_nn.dat
Figure			4-12

Table 4-1 Batch Local File Pool and JNDI Reference configuration

Figure 4-10 Connection Pool configuration (partial)

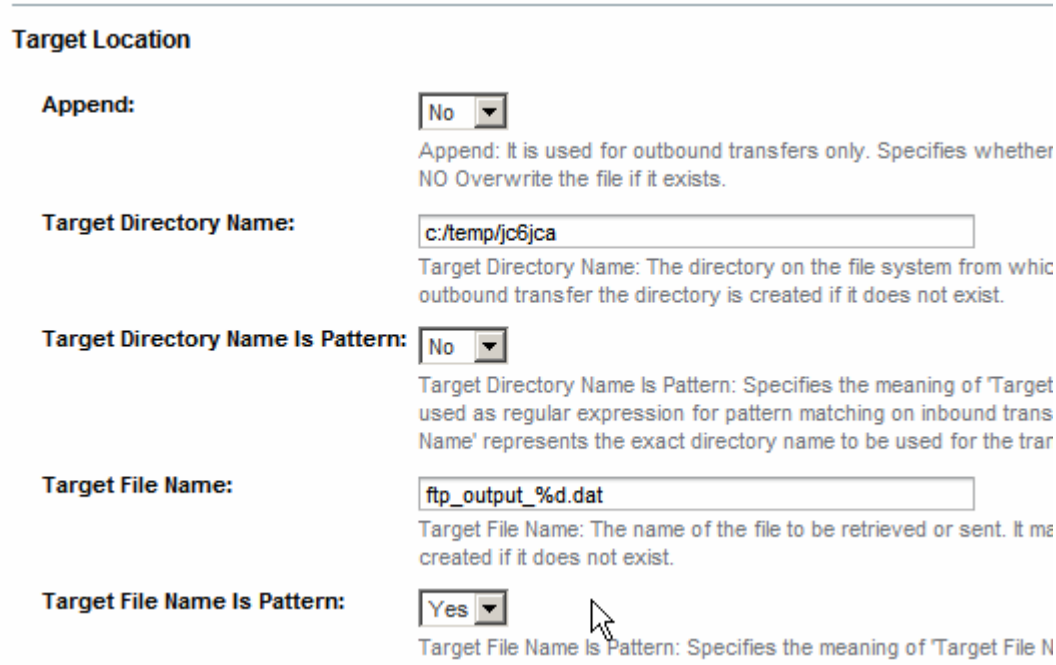

Figure 4-11 Batch Local File instance outbound file name and path

Figure 4-12 Connector Resource configuration

With the two connection pools configured and available we can proceed to develop the JCA Adapter-based FTP to Local File streaming solution.

5 Inbound FTP Streaming Solution

The solution will be triggered by arrival of a JMS message. It will then use the Batch FTP JCA and the Batch Local File JCA together to stream the payload from the FTP Server to the local file system.

We assume that the project group, JCABatchProjects_PG, already exists and is open. If not, see Section 2, “Create a Project Group JCABatchProjects_PG”, for details of how to create the project group.

Let’s create a new Enterprise -> EJB Module project, JCAFTPStreamingIn_EJBM. Once done, let’s create a new JCA Message-Driven Bean, jcaJMSTrigFTPToLocalFileStreamingIn, triggered by the JMS JCA Adapter. Figure 5-1 through 5-> illustrate key points.

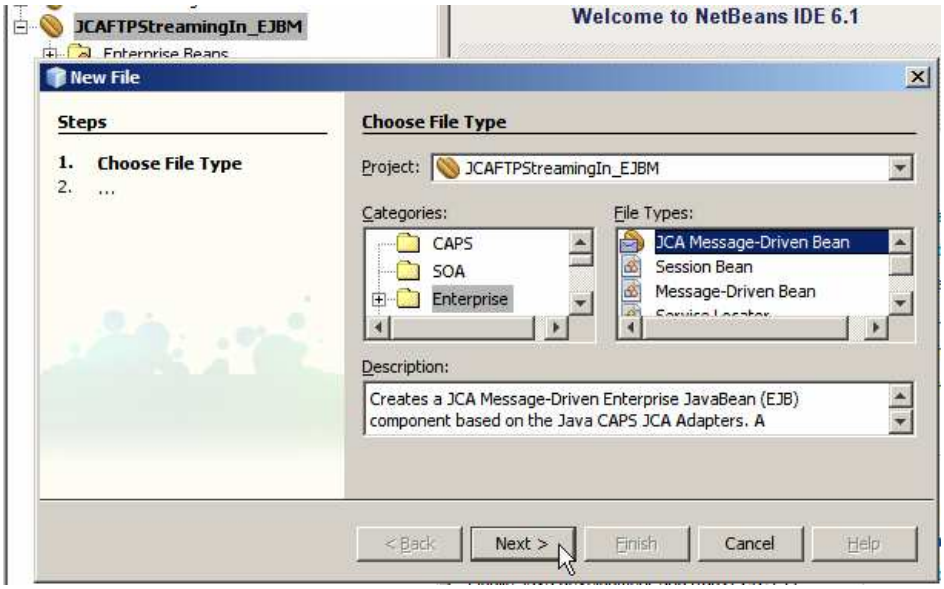

Figure 5-1 New JCA MDB

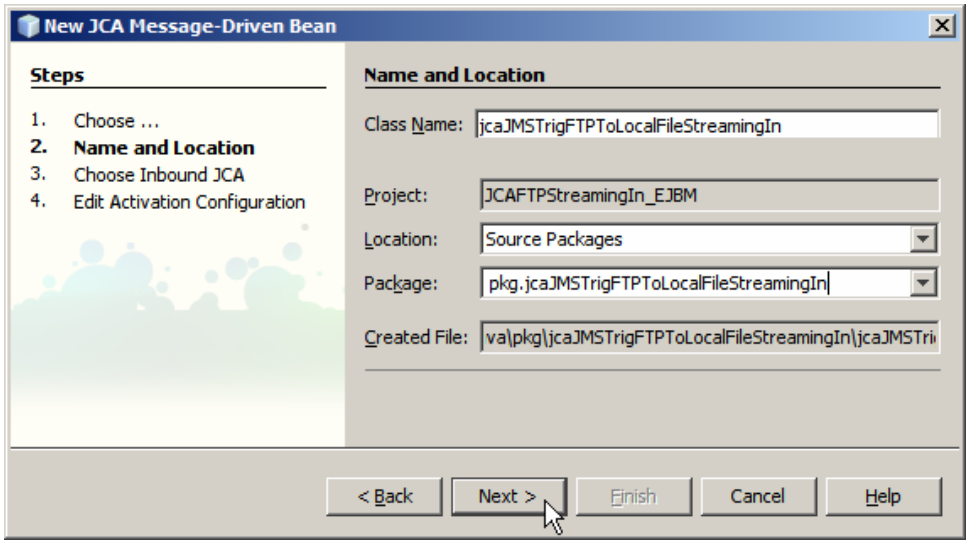

Figure 5-2 Name the bean and the package

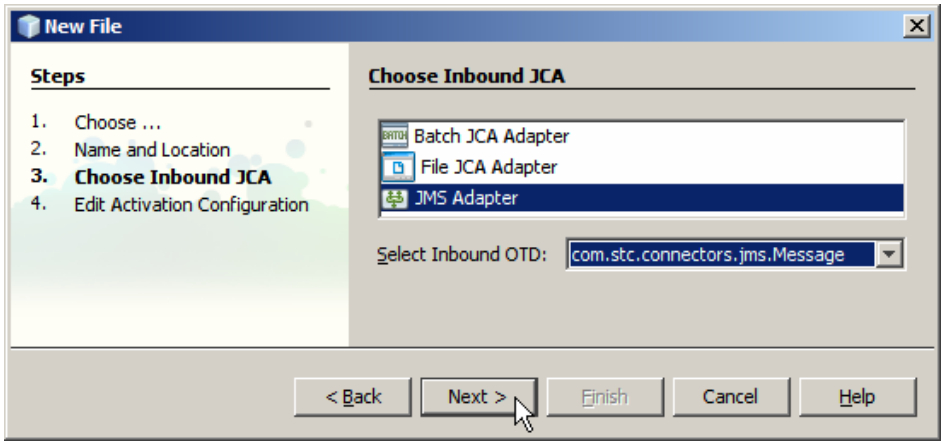

Figure 5-3 Choose JMS Adapter and con.stc.connectors.jms.Message OTD

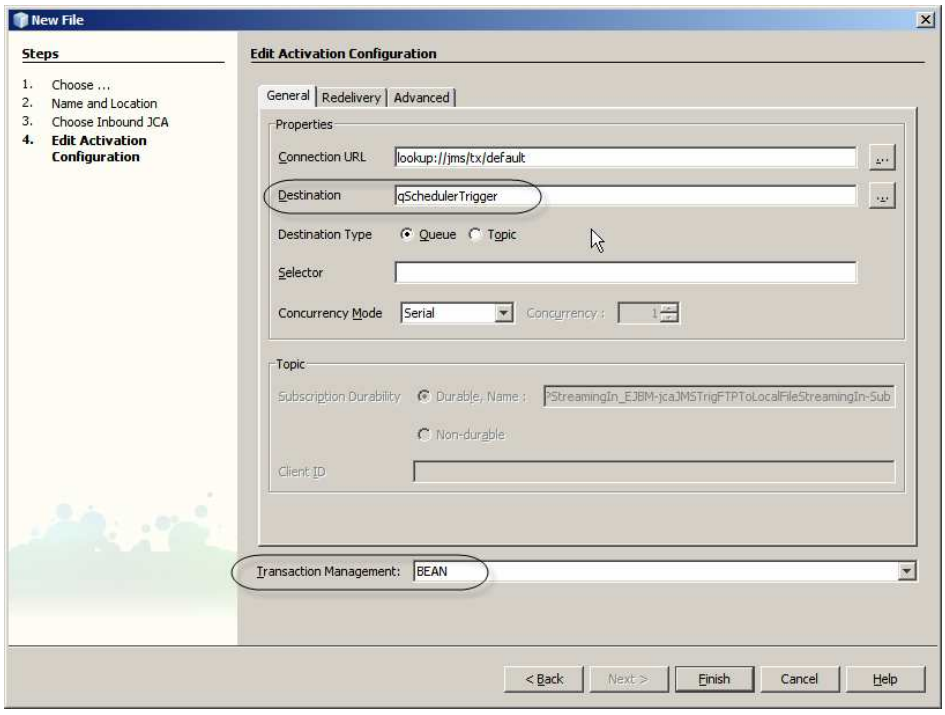

Figure 5-4 Hardcode destination name and choose BEAN managed transaction

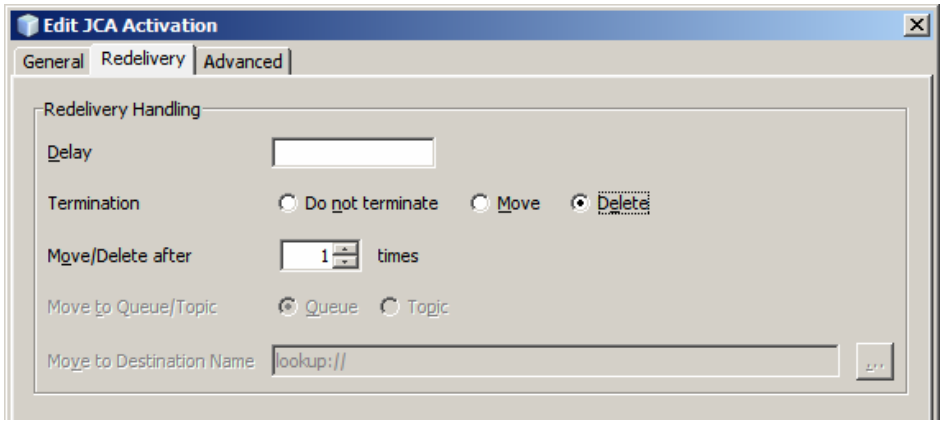

Figure 5-5 Configure redelivery to delete the trigger after 1 failed attempt

Once the wizard completes we are presented with a skeleton Java bean source, to which we will add the FTP JCA Adapter and the Batch Local File JCA Adapter.

Take not of the “receive” method. We will be adding code related to the adapters to that method.

Let’s drag the Batch JCA from the Palette to the source code window iside the receive method, as illustrated in Figure 5-6.

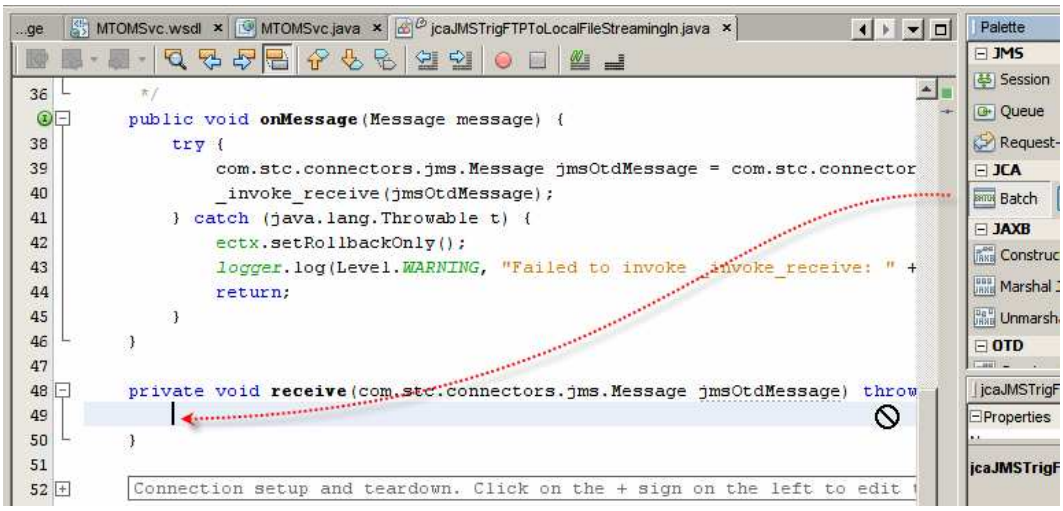

Figure 5-6 Add invocation of the Batch JCA Adapter

Select the BatchFTP OTD and click Next, as shown in Figure 5-7.

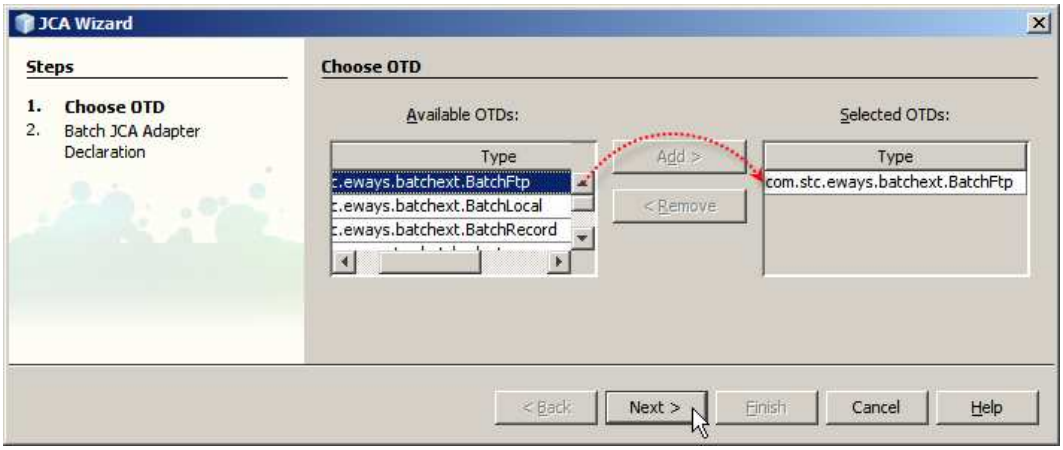

Figure 5-7 Select the BatchFTP OTD

Accept the method name, select the pool using the JNDI reference we configured ahead of time, cp-batch-ftp-localhost-mczapski, and name the variable G_BatchFTP, as shown in Figure 5-8. Here you see the use of some of my personal conventions. G_xxxx indicates that I am using the OTD to get stuff, and I am indicating what kind of ODT I am dealing with, BatchFTP.

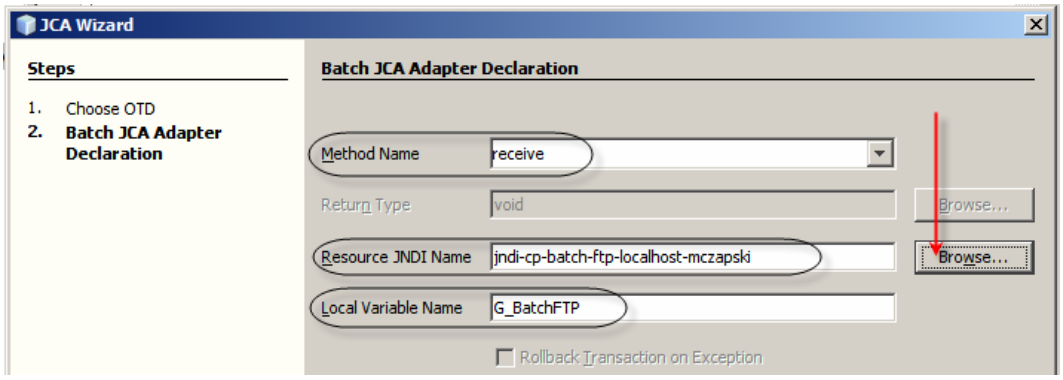

Figure 5-8 Name the method, choose the pool and name the variable.

Notice that some additional boilerplate code was added to the bottom of the source and that the signature of the receive method now contains additional argument, G_BatchFTP. We will look at the method signature a bit later.

Let's now add the Batch Local File JCA Adapter. Drag the Batch OTD from the Palette to the source window again. This time choose the BatchLocal OTD, choose the connection pool JNDI reference, jndi-cp-batch-localfile-c/temp/jc6jca/ftp_output_nn.dat, and name the variable P_BatchLocal. See Figures 5-9 and 5-10.

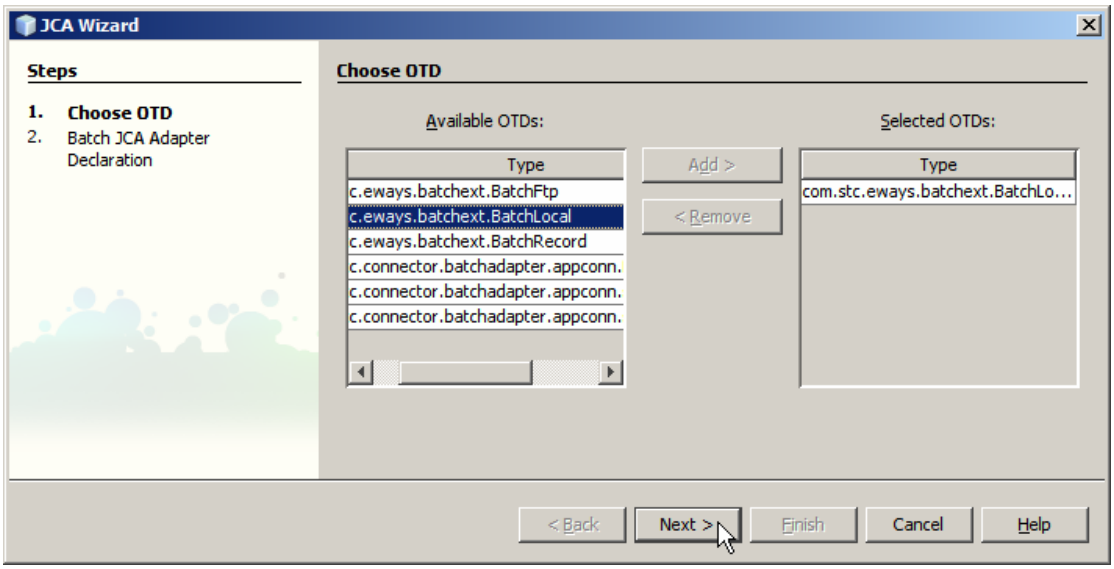

Figure 5-9 Choose BatchLocal OTD

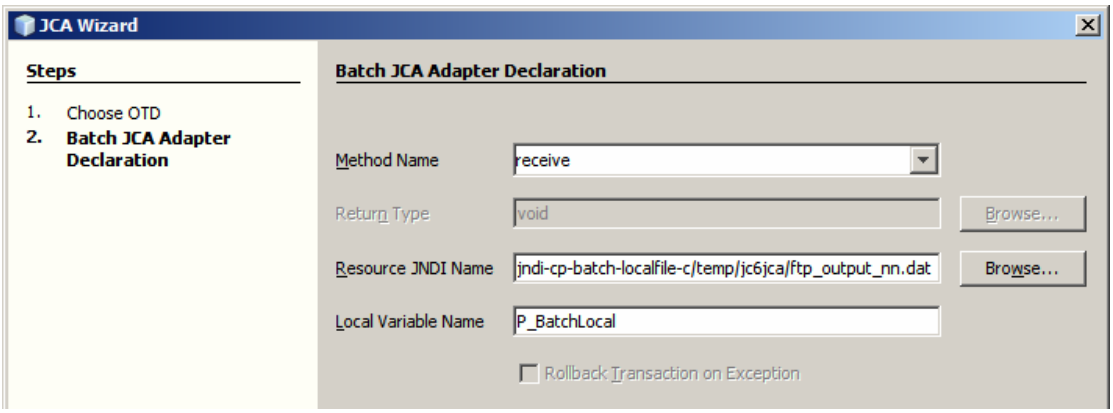

Figure 5-10 Choose connection pool JNDI reference and name the variable

To see what we have let's rename the com.stc.connectors.jms.Message argument from *jmsOtdMessage* to *input* and reformat the method signature for the receive method as shown in Figure 5-11.

```
private void receive
        (com.stc.connectors.jms.Message input
        ,com.stc.eways.batchext.BatchFtp G_BatchFTPOTD
        ,com.stc.eways.batchext.BatchLocal P_BatchLocalOTD)
            throws java.lang.Exception {

}
```

Figure 5-11 receive method signature

Java CAPS 5.1 JCD developer will see immediately that this method signature is almost identical to what would be the signature of the receive method in a Java Collaboration

Definition that is triggered by JMS and uses a Batch FTP and a Batch Local File eWays. The body of the code, as 5.1 developers will find, will be the same as well.

Let's copy a slab of code from Listing 3 in the "Java CAPS 5.1 and Java CAPS 6 - Streaming Large FTP Transfers" Note available at http://blogs.sun.com/javacapsfieldtech/entry/streaming_large_ftp_transfers_with, renaming vFTPIn to G_BatchFTPOTD and vLocalFileOut to to P_BatchLocalOTD.

Since we no longer have the logger we need to modify logger-related code. Let's replace all instance of "logger.debug(" with "Logger.getLogger(this.getClass().getName()).log(Level.FINE," and "logger.error" with "Logger.getLogger(this.getClass().getName()).log(Level.SEVERE,".

Let's modify the code so that it does not dynamically set output directory or the output file name but rather uses these statically configured in the pool and so that it does not hardcode the directory and file name at the FTP server but again uses these configured statically I the pool.

The code is reduced to configuring FTP timing parameters, performing streaming transfer and producing transfer timing output.

The code is reproduced in Listing 5-1, below.

```
try {
    long lStartMillis = System.currentTimeMillis();
    int iTimeoutMillis = 40 * 60 * 1000;

    G_BatchFTPOTD.getConfiguration().setDataConnectionTimeout(iTimeoutMillis);
    G_BatchFTPOTD.getProvider().setDataSocketTimeout(iTimeoutMillis);
    G_BatchFTPOTD.getProvider().setSoTimeout(iTimeoutMillis);

    com.stc.eways.common.eway.standalone.streaming.OutputStreamAdapter osa = null;
    osa = P_BatchLocalOTD.getClient().getOutputStreamAdapter();
    G_BatchFTPOTD.getClient().setOutputStreamAdapter(osa);
    G_BatchFTPOTD.getClient().get();

    long lMillis = System.currentTimeMillis() - lStartMillis;
    int iSeconds = 0;
    int iMinutes = 0;
    int iHours = 0;
    iSeconds = (int) (lMillis / 1000);
    iMinutes = iSeconds / 60;
    iHours = iMinutes / 60;
    Logger.getLogger(this.getClass().getName()).log(Level.FINE
        , "\n===>>> Execution took " + lMillis + " Milliseconds");
    Logger.getLogger(this.getClass().getName()).log(Level.FINE
        , "\n===>>> Execution took " + iSeconds + " Seconds");
    Logger.getLogger(this.getClass().getName()).log(Level.FINE
        , "\n===>>> Execution took " + iMinutes + " Minutes");
} catch (Exception e) {
    Logger.getLogger(this.getClass().getName()).log(Level.SEVERE
        , "Exception somewhere performing transfer", e);
    return;
}
```

Listing 5-1 Collaboration Code

Let's build and deploy the solution.

Let's make sure hat we have the file, jms-1_1-fr-spec.pdf, in the root directory of the user at the FTP Server we configured under the CAPS -> Connector Connection Pools -> cp-batch-

ftp-localhost-mczapski pool or whatever we used for the name of the Batch FTP connector pool.

Let's configure verbose logging for the STC.eWay.batch logging category to see what the two instances of the Batch JCA do. Use the Applicatin Server Administration Console to change logging levels dynamically. Figures 5-12 and 5-13 shows the places where the logging category can be specified and configured.

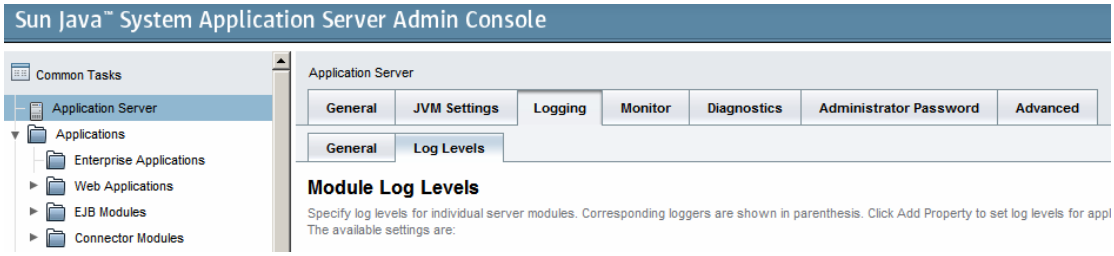

Figure 5-12 Navigate to Module Log Levels page

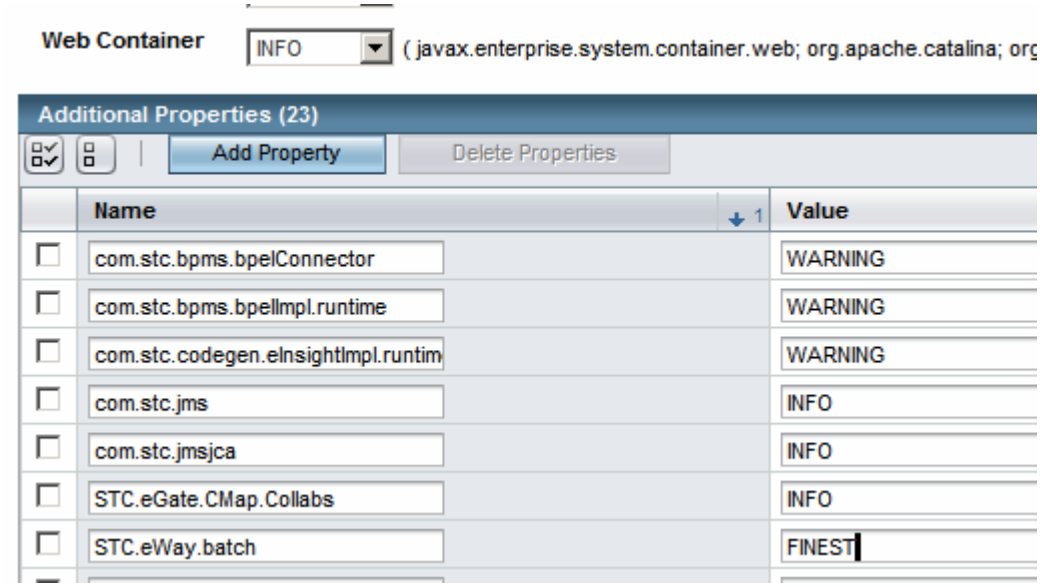

Figure 5-13 Change the logging level and save

Let's now use the Enterprise Manager to submit a text message with arbitrary content to qScheulerTrigger and observe the server.log to see the outcome.

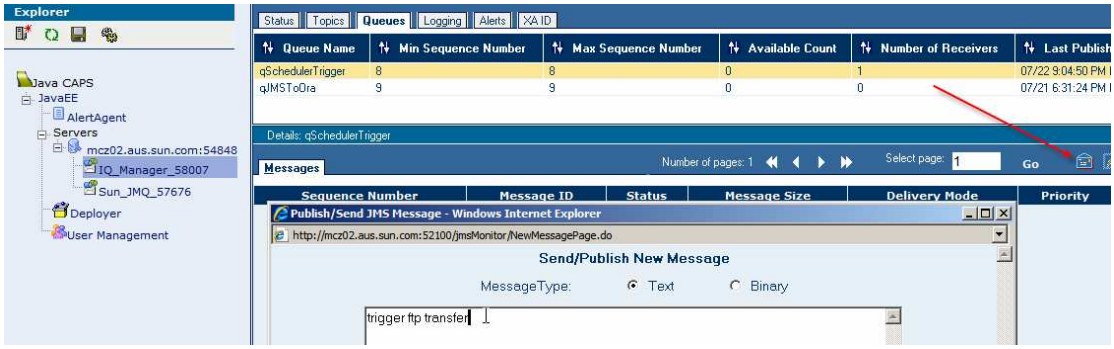

Figure 5-14 Submit a trigger message

If all went well the file will be transferred from the FTP Server to the local directory.

If things went badly, as they could if the configuration is not right, inspect the log to see what might have gone wrong.

Pool configuration changes are not dynamic. A running Batch JCA Adapter will not re-read its configuration. If you need to change pool configuration you need to disable/enable the EJB

Module or re-deploy it. Disabling/Enabling can be accomplished through the NetBeans IDE Services TAB or through the Application Server Administration Console. Figures 5-15 and 5-16 illustrate this.

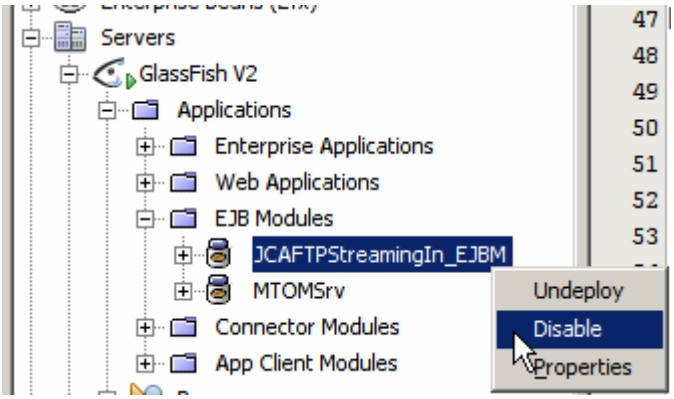

Figure 5-15 Disabling the EJB Module through the NetBeans IDE

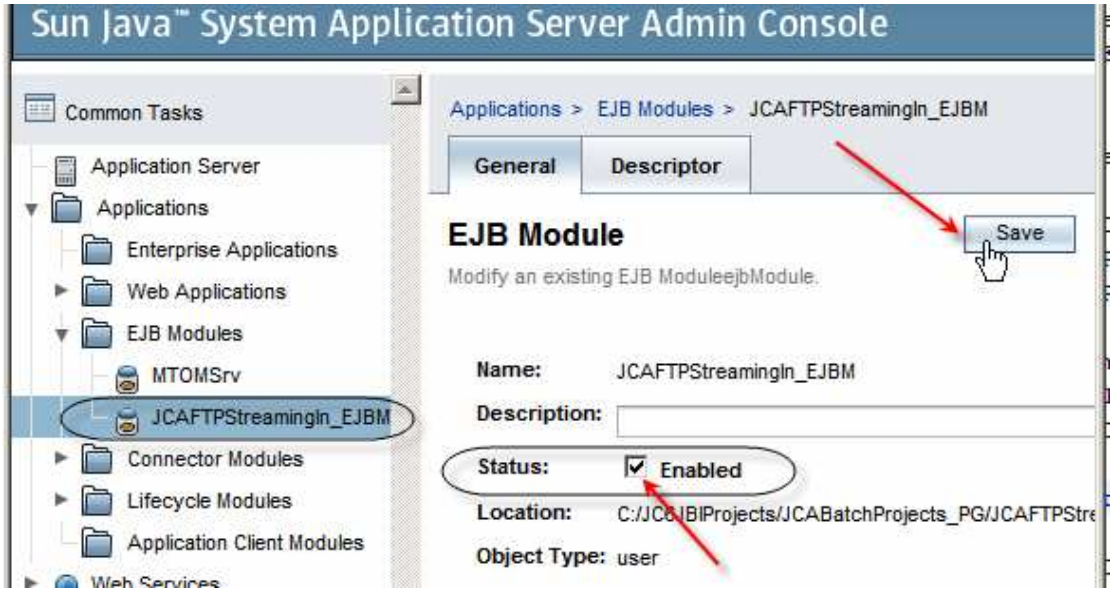

Figure 5-16 Enabling the EJB Module through the Application Server Admin Console

In a sample run server.log contained the following emtries:

```
[#|2008-07-25T15:09:26.187+1000|FINE|sun-
appserver9.1|STC.eWay.batch.com.stc.eways.batchext.FtpFileProviderImpl|_ThreadID=76;_T
hreadName=JMS Async
S51;ClassName=com.stc.connector.logging.JDKLogger;MethodName=debug;Context=JCAFTPStrea
mingIn_EJBM-jcaJMSTrigFTPToLocalFileStreamingIn-Context;_RequestID=168c9900-3006-44a7-
9f64-599a2379ac8d;|BATCH-DBG-D0159: FtpFileProviderImpl.listFiles(): Skip a line
[drwxr-xr-x   1 root       root            0 Jul 25 15:05 aaa].|#]


[#|2008-07-25T15:09:26.187+1000|FINE|sun-
appserver9.1|STC.eWay.batch.com.stc.eways.batchext.FtpFileProviderImpl|_ThreadID=76;_T
hreadName=JMS Async
S51;ClassName=com.stc.connector.logging.JDKLogger;MethodName=debug;Context=JCAFTPStrea
mingIn_EJBM-jcaJMSTrigFTPToLocalFileStreamingIn-Context;_RequestID=168c9900-3006-44a7-
9f64-599a2379ac8d;|BATCH-DBG-D0158: FtpFileProviderImpl.listFiles(): Accept a line [-
rw-r--r--   1 root       root     1234374343 Jul  9 21:43 jms-1_1-fr-spec.pdf].|#]
```

We were transferring 1.2Gb file between an FTP server and the local file system, where both used the same physical disk device.

Transfer timings looked like these shown below:

```
[#|2008-07-25T15:13:29.234+1000|INFO|sun-
appserver9.1|pkg.jcaJMSTrigFTPToLocalFileStreamingIn.jcaJMSTrigFTPToLocalFileStreaming
In|_ThreadID=76;_ThreadName=JMS Async S51;Context=JCAFTPStreamingIn_EJBM-
jcaJMSTrigFTPToLocalFileStreamingIn-Context;|

===>>> Execution took 243078 Milliseconds|#]


[#|2008-07-25T15:13:29.234+1000|INFO|sun-
appserver9.1|pkg.jcaJMSTrigFTPToLocalFileStreamingIn.jcaJMSTrigFTPToLocalFileStreaming
In|_ThreadID=76;_ThreadName=JMS Async S51;Context=JCAFTPStreamingIn_EJBM-
jcaJMSTrigFTPToLocalFileStreamingIn-Context;|

===>>> Execution took 243 Seconds|#]


[#|2008-07-25T15:13:29.234+1000|INFO|sun-
appserver9.1|pkg.jcaJMSTrigFTPToLocalFileStreamingIn.jcaJMSTrigFTPToLocalFileStreaming
In|_ThreadID=76;_ThreadName=JMS Async S51;Context=JCAFTPStreamingIn_EJBM-
jcaJMSTrigFTPToLocalFileStreamingIn-Context;|

===>>> Execution took 4 Minutes|#]
```

Not too bad for a 1.2Gb transfer. One expects the transfer between a remote FTP Server and local file system to take longer is the bandwidth of the network connection is low.

The point of this exercise was to demonstrate that an arbitrarily large payload can be streamed between locations without using anywhere near the amount of JVM Heap one would expect the JEE solution to require, if the payload was to be read into memory before being written out.

6 Summary

Much as was possible in 5.1.x, Java CAPS 6 Batch JCA Adapter allows one to implement a streaming solution that can transfer arbitrarily large payloads between FTP Servers and Local File System directories, in either direction. This Note discussed and illustrated a solution that accomplished inbound streaming from an FTP Server to the Local File System using “Java CAPS 5.1 and Java CAPS 6 - Streaming Large FTP Transfers” Note available at http://blogs.sun.com/javacapsfieldtech/entry/streaming_large_ftp_transfers_with. The Outbound streaming transfer can also be implemented using that Note as an example.