Java CAPS 6 Using JCA and JBI, Note 3

Batch Inbound, through Batch Local File to BPEL 2.0

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 and OneWay WSDL	5
5	Create JCA Message-Driven Bean	6
6	Create BPEL 2.0 Process	.17
7	Create a Composite Application	.25
8	Exercising the solution	.28
9	Summary	.28

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 mixed Java CAPS 5.x-like integration solution that retrieves a file from the local file system using JCA Adapters and passes its content to a BPEL 2.0 process executing in the JBI container. 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 send the payload as a string to a BPEL 2.0 Business Process, which will be triggered by this message and will execute in the JBI container. 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 OneWay WSDL interface and the JBI NMR to send it, as a String, to a BPEL 2.0 process. Both the JCA MDB and the BPEL process will be a part of the same JBI Composite Application and will communicate with one another using the Normalized Message Router (NMR).

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.

If the BatchInbound_generic resources already exist, perhaps because they were created when working through Note 2, they can be reused and the following steps in this section can be skipped.

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.

Resources > Connectors >	Connector Connection Pools
New Connector Co	nnection Pool (Step 1 of 2)
Create a Connector Pool, selec	t the associated Resource Adapter and Connection Definition, then click Next.
Name: *	BatchInbound_generic A unique name; can be up to 255 characters, must contain only alphanumeric, underscore, dash, or dot characters
Resource Adapter: *	Sun-batch-adapter Choose from the list of deployed resource adapters (connector modules)
Connection Definition: *	com sto connector batchadanter annoonn localfile Batchi ocalApplicationConnectionEactory

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.

Resources > Connectors >	Resources > Connectors > Connector Connection Pools			
New Connector C	onnection Pool (Step 2 of 2)			
Verify the Connection Pool se	attings, add properties defining the value for each property, and click Finish.			
General Settings				
Name:	BatchInbound_generic			
Resource Adapter:	sun-batch-adapter			
Connection Definition:	com.stc.connector.batchadapter.appconn.localfile.BatchLocalApplicationConnectionFactory			
Description:				
Pool Settings				
Initial and Minimum Po	ol Size: 8 Connections			
	Minimum and initial number of connections maintained in the pool			
Maximum Pool Size:	32 Connections			
	Maximum number of connections that can be created to satisfy client requests			
Pool Resize Quantity:	2 Connections			
	Number of connections to be removed when pool idle timeout expires			
Idle Timeout:	300 Seconds			
	Maximum time that connection can remain idle in the pool			
Max Wait Time:	60000 Milliseconds			
	Amount of time caller waits before connection timeout is sent			

Connection Validation

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.

Resources > Con	nectors > Connector Resources
New Conne	ctor Resource
To create a connec	ctor resource, specify the connection pool with which it is associate
JNDI Name: *	jndiBatchInbound_generic
	A unique name; can be up to 255 characters, must contain only alp
Pool Name: *	BatchInbound_generic
	Use the Connector Connection Pools page to create new pools
Description:	I
Status:	Enabled

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".

If the project group already exists, perhaps because it was created when implementing the solution described in Note 2, the following steps in this section can be skipped.

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 and OneWay WSDL

Create a new Enterprise -> EJB Module project,

BInboundThroughBLFToBPEL20_EJBM, making sure to pick the correct folder for project files. This has been done before so no picture will be shown ③

The JCA MDB we will create in the next section will interact with the BPEL 2.0 Process through the JBI NMR. To facilitate that interaction we must create a WSDL interface document in which the abstract part defines the message to be sent and the operation. Because the NetBeans IDE will not help us to create a Web Service Client infrastructure unless the concrete part of the WSDL is also defined we will define the concrete part including the binding and the service, pretending that it is a SOAP service.

The entire payload, as a string, will be sent to the BPEL process so the message will have one part of type xsd:string.

Let's create a new WDL Document, name it wsdlPayloadToBPEL, name the operation opSubmitPayload, make sure to change the type to One-Way operation, accept the SOAP binding default and complete the wizard. Figures 4-1 through 4-? Illustrate key steps in the process.

Pr	ojec	I × Files] Se	rvices	Start Page ×		
	S BIr	nboundThroughBL	ToBPEL20	FIRM			
Ī	Ē. 🖪	Enterprise Beans		New	•	🕵 Web Service Clier	nt
		Java Collaborations		Build		Web Service from	WSDL
gu	re 4-1	Server Resources	Wizard	Clean and Build Clean		Java Class	
N	ew WSDL	. Document					
ite	ps		Name and L	Location			
2	Choose I Name a	File Type and Location	File <u>N</u> ame:	wsdlPayloadToBPEL			
ŀ.	Concrete	e Configuration	Project:	BInboundThrough	BLFToBPEL20_EJBM		
			Fo <u>l</u> der:	src\java			Bro <u>w</u> se
			<u>C</u> reated File:	tchProjects_PG\BI	(nboundThroughBLFT)	oBPEL20_EJBM\src\java\wsdlPa	yloadToBPEL.wsd
			<u>T</u> arget Name	space: http://j2ee.	.netbeans.org/wsdl/w	sdlPayloadToBPEL	
			Import XN	4L Schema File(s)			
			XML Schema([s):			Browse
_							1

Figure 4-2 Name the WSDL

🗊 New WSDL Document				×
Steps	Abstract Configu	ration		
1. Choose File Type	Port Type Name:	wsdlPayloadToBPELPortType		
Name and Location Abstract Configuration	Operation <u>N</u> ame:	opSubmitPayload		
 Concrete Configuration 	Operation <u>Type</u> :	One-Way Operation		-
	Input:	Message Part Name	Element Or Type	
		sPayloadText	xsd:string	
		<u>A</u> dd <u>R</u> emove		
	Generate par	tnerlinktype automatically.		
		< Back Next >	Einish Cancel	Help

Figure 4-3 Name operation, nominate its type and name the message part

🗊 New WSDL Document		×
Steps	Concrete Config	uration
 Choose File Type Name and Location Abstract Configuration Concrete Configuration 	Binding <u>N</u> ame: Binding Type: Binding S <u>u</u> btype:	wsdIPayloadToBPELBinding SOAP © RPC Literal © Document Literal © RPC Encoded
	<u>S</u> ervice Name: <u>P</u> ort Name:	wsdlPayloadToBPELService wsdlPayloadToBPELPort
		< Back Next > Einish Cancel Help

Figure 4-4 Complete the Wizard

Since we will not actually implement the service at this point we don't need to concern ourselves with the end-point address.

5 Create JCA Message-Driven Bean

Create a new JCA Message Driven Bean, jcaBInboundThroughBLFToBPEL20. Figures 5-1 and 5-2 illustrate the initial steps.

Projec I Files	es 🛛 Start Page 🗙 😽 ws	dlPayloadToBPEL.wsdl ×
🖃 📎 BInboundThroughBLFToBPEL20_EJB	M Course MCDI	
🕀 🕞 Enterprise Beans	New 🕨	WSDL Document
🕀 🎒 Java Collaborations	Build	🔦 Web Service Client
🕀 🔂 Configuration Files	Clean and Build	Web Service from WSDL
Server Resources	Clean	🔊 Java Class
🖨 🕣 Source Packages	Clean .	
🖻 🚎 <default package=""></default>	Verify	Session Bean
wsdlPayloadToBPEL.wsdl	Generate Javadoc	🚔 JCA Message-Driven Bean
pkg.jcaBInboundThroughBLFToBF	Run	Cracle Otd Wizard

Figure 5-1 Choose JCA Message-Driven Bean

🗊 New JCA Message-Driven Bean		×
Steps	Name and Location	
 Choose Name and Location 	Class Name: jcaBInboundThroughBLFToBPEL20	
 Choose Inbound JCA Edit Activation Configuration 	Project: BInboundThroughBLFToBPEL20_EJBM	
	Location: Source Packages	-
	Package: pkg.jcaBInboundThroughBLFToBPEL20	-
	Created File: \\src\java\pkg\jcaBInboundThroughBLFToBPEL20\jcaBInboundThroughBLFToBPEL20.jav	a
	< Back Next > Finish Cancel Help	

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.

1. 2. 3. 4.	Choose Name and Location Choose Inbound JCA Edit Activation Configuration	Batch JCA Adapter File JCA Adapter
	<u> </u>	c Next > Emish Cancel Help

Figure 5-3 Choose Batch JCA Adapter.

 Choose Name and Location Choose Inbound JCA Edit Activation Configuration Configuration Configuration Click to Edit > Projectinfo Click to Edit > EvayActivationSpecMBeanClass Connector management ice.system.miceans.S Click to Edit > Steady Pool Size Resize Quantity Max Pool Size Pool Idle Timeout in Seconds Max Wait Time in Milliseconds 10000 	teps	Edit Activation Configuration	
Choose Inbound JCA EwayEndpointClass com stc. connector.batchadapter.system.BatchEndpoint Edit Activation Configuration EwayActivationSpecMBeanName CAPS:GUID={2300000-E46DCBC0130100-8199BCS Configuration < Click to Edit > Image: Click to Edit > Projectinfo < Click to Edit > Image: Click to Edit > EwayActivationSpecMBeanClass com stc. connector.management.ica.system.misans.S EvayActivationSpecMBeanClass com stc.connector.management.ica.system.misans.S EvayActivationSpecMBeanClass com stc.connector.management.ica.system.misans.S Max Pool Size 0 Pool Idie Timeout in Seconds 600	Choose Name and Location	⊡Properties	
Edit Activation Configuration EwayActivationSpecMBeanName CAPS:GUID=(2300000-E46DCBC0130100-8199BCS Configuration < Click to Edit > Projectinfo < Click to Edit > EwayActivationSpecMBeanClass com.stc.connector management.jca.system.mbcanS.S EAdvanced Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000	Choose Inbound JCA	EwayEndpointClass	com.stc.connector.batchadapter.system.BatchEndpoint
Configuration < Click to Edit > ProjectInfo < Click to Edit > EwayActivationSpecMBeanClass com stc.connector management ica.system means S Advanced Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000	4. Edit Activation	EwayActivationSpecMBeanName	CAP5:GUID={23000000-E46DCBC0130100-8199BC5
Projectinfo < Click to Edit > EwayActivationSpecMBeanClass com.stc.connector.management.jca.system.mbeans.S EAdvanced 0 Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000	Configuration	Configuration	< Click to Edit >
EwayActivationSpecMBeanClass com.stc.connector.management.jca.system.miteans.S Advanced 0 Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		ProjectInfo	< Click to Edit >
Advanced Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		EwayActivationSpecMBeanClass	com.stc.connector.management.jca.system.mbeans.S
Steady Pool Size 0 Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		⊟Advanced	
Resize Quantity 16 Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		Steady Pool Size	0
Max Pool Size 64 Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		Resize Quantity	16
Pool Idle Timeout in Seconds 600 Max Wait Time in Milliseconds 10000		Max Pool Size	64
Max Wait Time in Milliseconds 10000		Pool Idle Timeout in Seconds	600
		Max Wait Time in Milliseconds	10000

Figure 5-4 Trigger configuration editor

Let's configure the Batch Inbound properties as shown in Figure 5-5, click Close and click Finish. This assumes that the project developed in Note 2 is not deployed,

otherwise the two Batch Inbound instances will compete for the same file. If this is the case just change the name of the trigger file.

🗊 activationnode - Co	nfiguration	×			
Properties					
Directory Name	C:\Temp\JC6JCA				
Directory Name is Pattern	false				
File Name	^trigger_[0-9][0-9]*.txt\$				
File Name is Pattern	true				
Schedule Interval	15000				
Schedule Interval					
schedule-interval	N				
	6				
	_				
		Close			

Figure 5-5 Batch Inbound configuration

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

```
package pkg.jcaBInboundThroughBLFToBPEL20;
import javax.ejb.MessageDriven;
import com.stc.connector.batchadapter.appconn.BatchAppconnMessage;
import com.stc.connector.batchadapter.system.BatchListener;
/***
 * @author mczapski
 */
@MessageDriven(name="pkg.jcaBInboundThroughBLFToBPEL20.jcaBInboundThroughBLFToBPEL20")
public class jcaBInboundThroughBLFToBPEL20 implements BatchListener {
    public jcaBInboundThroughBLFToBPEL20 () {
    }
    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. Figures 5-7 and 5-8 highlight key points.



Figure 5-7 Triggering the Edit JCA Configuration Dialogue Box



Figure 5-8 Accessing Inbound JCA Adapter Configuration

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 5-9, choose the Batch Local File OTD, as illustrated in Figure 5-10, 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 5-11. 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 5-12. After a while it will change to look similar to that shown in Figure 5-13, 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 5-9 Adding Batch JCA to the MDB

🇊 JCA Wizard			×
Steps	Choose OTD		
1. Choose OTD 2. Batch ICA Adapter	<u>A</u> vailable OTDs:	1	Selected OTDs:
Declaration	Туре	A <u>d</u> d >	Туре
	com.stc.eways.batchext.BatchFtp	in the second	com.stc.eways.batchext.BatchLocal
	com.stc.eways.batchext.BatchLocal	< <u>R</u> emove	
	com.stc.eways.batchext.BatchRecord		•
	com.stc.connector.batchadapter.appconn.ftps.FTPO		
	com.stc.connector.batchadapter.appconn.sftp.SFTP		
	com.stc.connector.batchadapter.appconn.scp.SCP		
	<u> </u>		
		< <u>B</u> ack	Next > Einish Cancel Help

Figure 5-10 Choosing the Batch Local File OTD

Steps	Batch JCA Adapter	Declaration	
Choose OTD Batch JCA Adapter Declaration	Method Name	βLFGet ▼	
	Retur <u>n</u> Type	void	Browse
	Resource JNDI Name	jms/tx/default	Browse
	<u>L</u> ocal Variable Name	batch	/
10. Jul		C Log Exception	

Figure 5-11 Invent the method name and start the process of choosing JNDI Reference

Batch JCA Adapter Declaration				
Method Name	receive			
Retur <u>n</u> Type	void			
Descures INDI Nesse	includefault			
Resource SND1 Name	mis(c) dei aut			
Local Variable Name	batch			
	Rollback Iransa			

Figure 5-12 Waiting for the JNDI resource list to be assembled

Thoose JNDI name of Connector Resource	×
€ GlassFish V2	
E Resources	
End Connectors	
Connector Resources	
🕀 👘 MyOraXAPoolJNDI	
🕀 👘 javacaps/ora/localtx/jcaps511/oe_pool	
🕀 👘 jms/tx/stcms1	
🕀 👘 jndiBatchInbound_generic	
🕀 🔂 jms/notx/jmq1	
🕀 🕞 jms/notx/stcms1	
± 0	
OK Cancel <u>H</u> elp	

Figure 5-13 Choose JNDI Reference to the connection pool

Slabs of boilerplate code are added to the Java source. Figure 5-14 shows some of that code.



Figure 5-14 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 5-15 illustrates this.



Figure 5-15 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 5-14, 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 5-16.



Figure 5-16 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 5-17 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 5-17 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 5-18 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().setTargetFileName(sGUIDFileName);
G_BatchLocalFile.getConfiguration().setTargetFileName(sGUIDFileName);
G_BatchLocalFile.getConfiguration().setTargetFileNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setPostDirectoryNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setPostDirectoryNameIsPattern(false);
G_BatchLocalFile.getConfiguration().setPostFileName(sOrigFileName + ".~in");
G_BatchLocalFile.getConfiguration().setPostFileNameIsPattern(false);
G_BatchLocalFile.getClient().get();
```

Figure 5-18 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 and send it, as a string, to a BPLE 2.0 Business Process.

Recall, from Section 4, "Create EJB Module and OneWay WSDL", the WSDL we created. It can be seen in Source Packages -> <default package> as wsdlPayloadToBPEL.wsdl. Figure 5-19 shows this.



Figure 5-19 wsdlPayloadToBPEL.wsdl, created in Section 4

Using NetBeans facilities let's create a Web Service Client reference. Figures 5-20 through 5-22 illustrate key steps in the process.



Figure 5-20 Trigger Web Service Client wizard

New Web Service Client	
iteps	WSDL and Client Location
Choose File Type WSDL and Client Location	Specify the WSDL file of the Web Service.
and a	pkg

Figure 5-21 Find the 'local file' that contains the WSDL and select it

WSDL and Clie	ent Location	
Specify the WSD	DL file of the Web Service.	
C Project:		Brows <u>e</u>
	IProjects\JCABatchProjects_PG\BInboundThroughBLFToBPEL20_EJBM\src\java\wsdIPayloadToBPEL.wsdI	Browse
C WSDL URL:		Set Proxy
Project: Package:	controlution in rook age>	
Client <u>S</u> tyle:	JAX-WS Style	
🖵 Generate D	ispatch code	
	WSDL and Clie Specify the WSU Project: Uccal File: WSDL URL: Specify a location Project: Pgckage: Client Style: Generate D	WSDL and Client Location Specify the WSDL file of the Web Service. Project: Local File: IProjects]/CABatchProjects_PG\BInboundThroughBLFToBPEL20_EJBM\src\java\wsdIPayloadToBPEL.wsdI WSDL LiRL: Specify a location for the client. Project: BInboundThroughBLFToBPEL20_EJBM Pgckage: <cdefault package=""> Client Style: JAX-WS Style Generate Dispatch code</cdefault>

Figure 5-22 Finish the wizard

If you have difficulties locating the WSDL file right click on the WSDL file name in the Source Packages -> <default package>, choose Proeprties and see where the file is hiding.

After due activity, which is logged in the Output window, NetBeans adds a Web Service Reference node tree to our project. Figure 5-23 illustrates this.



Figure 5-23 Web Service Reference created by NetBeans, based on the WSDL

To invoke the web service, or as will be the case in this project, to send a message to the BPEL 2.0 process, let's drag the web service operation, opSubmitPayload, from the Web Service Reference tree into the Java source window following the line that reads "G_BatchLocalFile.getClient().get();". Figure 5-24 illustrates this.



Figure 5-24 Adding Web Service invocation to the Java source

NetBeans adds a slab of boilerplate code which needs formatting and modification, see Figure 5-25.



Figure 5-25 Boilerplate code for web service invocation

All we need to do to pass the payload to the BPEL process, which we will develop next, is to modify the statement that reads port.opSubmitPayload(...), as shown in Figure 5.26.



Figure 5-26 Sending file payload, as string, to the BPEL process.

I used the expression "to the BPEL process". This is a reflection of what this Note is supposed to achieve rather then a reflection of what the code is doing. The service implementation which is being invoked could be any service implementation, whether JBI-based or not, whether Java or .NET, as long as it implemented the WSDL-mandated interface.

To make sure all is well, let's build this project, but not deploy it.

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 to a service implementation that complies with the WSDL we used.

6 Create BPEL 2.0 Process

Let's create the BPEL 2.0 process which will receive the file payload, as string, from the JCA solution we built in the previous section. The process will be a simple one. It will receive the string and write it to a file using the File Binding Component (FILE BC).

We start with a BPLE Module project, BInboundThroughBLFToBPEL20_BPELM, as illustrated in Figures 6-1 and 6-2.

Choose Project	
Categories: CAPS Java Web Enterprise Mobility UML SOA Ruby C/C++ NetBeans Modules Encerprise Samples	Projects: Composite Application BPEL Module Data Integrator Module SQL Module v2 Preview XSLT Module v2 Preview
Description: Creates an empty BPEL Module	e, which may contain multiple BPEL processes.
	Choose Project Categories: CAPS Web Enterprise Mobility UML SOA Ruby C/C++ NetBeans Modules E Description: Creates an empty BPEL Module

Figure 6-1 Creating a BPEL Module project

🝞 New BPEL Module			×
Steps	Name and Loca	tion	
 Choose Project Name and Location 	Project <u>N</u> ame:	BInboundThroughBLFToBPEL20_BPELM	
	Project Location:	C:\JC6JBIProjects\JCABatchProjects_PG	Browse
	Project Fol <u>d</u> er:	cts\JCABatchProjects_PG\BInboundThroughBLFToBPEL20_BPELM	
	, Set as <u>M</u> ain Pr	oject	
		<back next=""> Einish Cancel</back>	Help

Figure 6-2 Naming the module and choosing project location

To write to a file from a BPEL 2.0 process, as with anything to do with interaction between a BPEL 2.0 process and the external world, requires a WSDL. This WSDL will represent both the message structures exchanged between the BPEL process and the File BC, and the configuration of the File BC.

Let's create a WSDL Document, wsdlOutFile, as shown in Figures 6-3 through 6-6. Make it a One-Way Operation with a one part message of type xsd:string, using the FILE binding.

I Proje I Files Services Image: Service state Image: Service state Service state Image: Service state Service state Service state	Start Page ×	
Process Files	New 🕨	🔂 BPEL Process
BInboundThroughBLFToBPEL20_EJBM	Build	🐺 WSDL Document
🗄 🗋 Enterprise Beans	Clean and Build	📲 Oracle Otd Wizard 🗟
🕀 🎒 Java Collaborations	Clean	
🕀 🕞 Web Service References		Other

Figure 6-3 Start the New WSDL Document wizard

📦 New WSDL Document		X
Steps	Name and Location	
Choose File Type Name and Location Abstract Configuration	File <u>N</u> ame: wsdlOutFile	
4. Concrete Configuration	Project: BInboundThroughBLFToBPEL20_BPELM	
	Folder: src	Bro <u>w</u> se
	Created File: jects\JCABatchProjects_PG\BInboundThroughBLFToBPEL20_BPELM\src\v	wsdlOutFile.wsdl
	Target Namespace: http://j2ee.netbeans.org/wsdl/wsdlOutFile	
	Import XML Schema File(s)	
	۲۷ML Schema(s):	Browse
\searrow		
	< Back Next > Finish Cancel	Help

Figure 6-4 Name the WSDL document

eps	Abstract Configu	ration		
Choose File Type	Port Type Name:	wsdlOutFilePortType		
 Name and Location Abstract Configuration Concrete Configuration 	Operation Name:	opWriteFile		
	Operation Type:	One-Way Operation		*
	Input:	Message Part Name	Element Or Type	
		sOutPayload	xsd:string	
		1		
	-	<u>A</u> dd <u>R</u> emove		
	I Generate par	tnerlinktype automatically.		
	✓ Generate par	tnerlinktype automatically.		

Figure 6-5 One-Way Operation with a single part message of type xsd:string

📦 New WSDL Document		X
Steps	Concrete Config	uration
 Choose File Type Name and Location Abstract Configuration Concrete Configuration 	Binding Name: Binding Type: Binding Subtype Service Name: Port Name:	wsdlOutFileBinding FILE * * wsdlOutFileService wsdlOutFilePort
		< Back Next > Einish Cancel Help

Figure 6-6 Binding Type: FILE

Let's configure the directory to which to write the file and the name of the file to write. Figured 6-7 and 6-8 call out the key parts of the WSDL. Let's name the directory path "C:\Temp\JC6JCA" and the file "output_%d.out", a name pattern.



Figure 6-7 Configuring directory path



Figure 6-8 Naming the file and indicating the name is a pattern

This provides the configuration of the outbound File BC.

With both WSDLs available, the WSDL we created to define the interface between the JCA and the BPEL process, and the WSDL we created just now for the File BC, we are ready to create a BPEL process.

Let's create a BPEL Process, bpelProcessPayload, as shown in Figures 6-9 and 6-10.



Figure 6-9 Start the new BPEL Process wizard

New BPEL Process	×
Steps	Name and Location
1. Choose File Type 2. Name and Location	File Name: bpelProcessPayload
	Project: BInboundThroughBLFToBPEL20_BPELM
	Folder: src Browse
	Created File: ABatchProjects_PG\BInboundThroughBLFToBPEL20_BPELM\:
	Target Namespace: :p://enterprise.netbeans.org/bpel/BInboundThroughBL
	< <u>B</u> ack Next > <u>Finish</u> Cancel <u>H</u> elp

Figure 6-10 Naming the BPEL process

From the Source Packages -> <default package> of the BInboundThroughBLFToBPEL20_EJBM EJB Module project, let's drag the wsdlPayloadToBPEL WSDL and drop it onto the 'target ball' of the BPEL process as shown in Figure 6-11.



Figure 6-11 Add web service interface to the BPEL process

While is is not necessary, I like to rename the default partner links. Figure 6-12 illustrates this. The partner link name will be plkInJCA.



Figure 6-12 Ranamed Partner Link

Let's now add the outbound File BC WSDL to the process canvas as shown in Figure 6-13 and rename the partner link to plkOutFile.



Figure 6-13 Adding File BC WSDL to the process editor canvas

To receive a message and write it to a file we need a Receive activity, an Assign activity which maps the payload from the output of the JCA to the input of the File BC, and an Invoke activity, which will cause the write to take place. Let's drag these activities from the palette to the canvas. Figure 6-14 illustrates the addition of the Invoke activity. The other activities were added the same way.



Figure 6-14 Adding the Invoke activity to the BPEL Editor canvas

Select the Receive activity and click the Edit button, as shown in Figure 6-15. This will allow us to choose the partner and create a business process variable to contain the message we will receive.



Figure 6-15 Edit properties of the Receive activity

Choose Partner Link (this is where having renamed partner links comes handy) as shown in Figure 6-16 and click "Create ..." button alongside the Iput Variable data entry box and name the new variable vInPayload as shown in Figure 6-17.

	Process Start:	Exit Compensate CompensateSco Estructured Act
plkInJCA	Receive1 [Receive] - Property Editor Main Correlations Name: Receive1	Dr
	Partner Link: Ogeration: Input Variable: Create Instance	<u>Cr</u> eate

Figure 6-16 Choose plkInJCA Partner Link for the receive side

Receive1	[Receive] - Property Editor		×
<u>N</u> ame:	Receive1		
Partner Link:	plkinJCA		-
Operation:	opSubmitPayload		_
Input Variabl	e:	C <u>r</u> eate	Browse
Create Ir	📦 New Input Variable	×	
(Name: VInPayload		
	Type: tns:opSubmitPayloadRequest		
	Scope: bpelProcessPayload	-	Help
	Ok	Cancel	9] 🖟 🕜

Figure 6-17 Create and name Input Variable

Repeat the process for the Invoke activity, choosing the plkOutFile partner link and naming the variable vOutPayload as shown in Figure 6-18.

opSubmit		opWriteFile	voke1 [Invoke] - Prope D × tain perty Editc
	Main Correlation	oke] - Property Editor s Invoke1	
	Partner Link: Operation:	plkOutFile opWriteFile	
	Input Variable: Output Variable:	New Input Variable	Create Browse
I GlassFish V2 × BInbour		Iype: e.netbeans.org/wsdl/wsdlOutFile:opV Scope: bpelProcessPayload	VriteFileRequest
		Ok	Cancel

Figure 6-18 Configuring the File BC partner link

Finally, let's map the output of the JCA to the input of the File BC. Select the Assign activity and click the Mapper Tab as shown in Figure 6-19.



Figure 6-19 Switching to Mapper to complete the Assign activity

Select the vOutPayload variable's sOutPayload node in the Variables tree at the right hand side, select the vInPayload variable's sInPayload node in the Variables tree at the left hand side and drag a line fro it to the sOutPayload at the left hand side as illustrated in Figure 6-20.

Start Page 🗙 👫 wsdlOutFile	wsdl × 📴 bpelProcessPaylo	ad.bpel * × 🔹 🕨 🗖	
Source Design Mappe	er Logging		
😵 Operator 📑 Boolean 🔬 String 📲 Node 📰 Number 📸 Date & Time 🔂 BPEL			
All Output		Input All	
🖃 🖶 Variables		Variables 🖶 – E	
vOutPayload		vOutPayload 🎴 –	
SPayloadText —		->sQutPayload 🖻	
🗄 🖶 Partner Links		vInPayload 🏪 –⊕	
		Partner Links 🖶–🕀	

Figure 6-20 Map output to input

The process is now complete. Switch back to Design mode and see the completed process. It ought to look like that shown in Figure 6-21.



Figure 6-21 Complete JCA to File Business Process.

This process does not do anything useful. We developed this process to illustrate how a JCA-based service can trigger a JBI-based BPEL process.

To make sure the process builds, let's build it.

7 Create a Composite Application

This two modules, the EJB Module that contains the JCA project and the BPEL Module that contains the BPEL process, will be joined as a JBI Composite Application. Le't create a SOA Composite Application Module, BInboundThroughBLFToBPEL20_CA, as shown in Figures 7-1 and 7-2.

🗊 New Project		×
Steps	Choose Project	
 Choose Project 	Categories: CAPS ▲ Java ▲ Web ▲ Enterprise ▲ Mobility ↓ CAPS ▲ Web ↓ Enterprise ▲ Mobility ↓ Categories ↓ Web ↓ Categories ↓ Cat	Projects: Composite Application BPEL Module Data Integrator Module SQL Module v2 Preview XSLT Module v2 Preview tion project, which may include multiple BPEL
	< <u>B</u> ack	Next > K Einish Cancel Help

Figure 7-1 Choose SOA -> Composite Application project

The w Composite Application Pro	ject	×
Steps	Name and Location	
 Choose Project Name and Location 	Project Name: BInboundThroughBLFToBPEL20_CA Project Location: C:\JC6JBIProjects\JCABatchProjects_PG Project Folder: ojects\JCABatchProjects_PG\BInboundThroughBLFToBPEL20_CA	Browse
	☑ Set as <u>M</u> ain Project	
	< <u>B</u> ack Next > Einish Cancel	Help

Figure 7-2 Choose project name and location

Drag the BInboundThroughBLFToBPEL20_EJBM Module onto the JBI Modules swim line of the Service Assembly Editor canvas as shown in Figure 7-2.



Figure 7-3 Add BInboundThroughBLFToBPEL20_EJBM module to the Service Assembly

Drag the BInboundThroughBLFToBPEL20_BPELM BPEL Module onto the JBI Modules swim line of the Service Assembly Editor canvas as shown in Figure 7-3.



Figure 7-4 Add BInboundThroughBLFToBPEL20_BPELM Module to the Service Assembly

Right-click the name of the Composite Application module,

BInboundThroughBLFToBPEL20_CA, and choose Build. Once the build process completes yoy should see a Service Assembly drawn as shown in Figure 7-5.





The build process worked out connectivity between modules and added appropriate Binding Components. Since a complete WSDL was used on the inbound side of the BPEL process, the build process added a SOAP BC and connected it to the BPEL module. Since the JAC Module uses the same WSDL it too was connected to the BPEL Module. If we leave the Service Assembly as it is the BPEL Process will be able to be triggered both by the JCA Module, as we intended, and by a Web Service invocation. Since we don't wish the BPEL module to be exposed as a Web Service we simply select and delete the SOAP BC, the build the Composite Applciation module again. The Service Assembly after removal of the SOAP BC is shown in Figure 7-6.



Figure 7-6 Finial Service Assembly

Notice that there is no indication of how the EJB Module is triggered. Only the inspection of the module can tell as that.

Notice, too, that the communication between the EJB Module and the BPEL Module will go over the Normalized Message Router. It will not be SOAP over HTTP even though the WSDL we created at the beginning of this Note might suggest otherwise.

By adding both modules to the same JBI-based Composite Application we are eliminating this inefficiency.

Let's now deploy the Composite Application.

8 Exercising the solution

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 8-1. Not the regular expression pattern, we specified for the Batch Inbound.

```
DKLogger;MethodName=debug;_RequestID=33670be5-7e80-4e30-b91c-98c16fe800e6;|.accept: It is not a ma
at|#]
```

Figure 8-1 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 file with the name like output_0.out will be produced.

9 Summary

This document walked the reader, step-by-step, through the process of creating and exercising a Java CAPS 6 mixed JCA- and JBI-based solution. All of this work was done using JCA Adapters, EJBs in the JavaEE Service Engine, BPEL 2.0 Service Unit in the BPEL Service Engine and a File Binding Component, all hosted in the JBI and JEE Containers within the GlassFish Application Server.