Getting Hundreds of Files using Batch Local File eWay in Java CAPS 6

Michael.Czapski@sun.com

May 2009

Table of Contents

Introduction

Occasionally one needs to pick up and process a large number of files, on the order of hundreds or thousands. With the Batch Inbound eWay/JCA Adapter it is not possible to pick up more then one file per poll. The Batch Local File, if triggered by some event other then an appearance of a file in a directory, perhaps a Scheduler trigger of a manual trigger, with correctly designed logic can process many files in a single invocation.

This document discusses how Batch Local File-based solution can be constructed to effectively process hundreds of files in a single pass.

Solution Outline

In this solution the file processing logic will be very simple. The Java Collaboration or a JCA MDB if one so desires, will receive a trigger message from a JMS Queue. This trigger message will be ignored as its sole purpose is to get the collaborations started. Once started, the collaboration will enter a loop in which a Batch Local File Adapter will perform a GET operation looking for a file, which fits a statically configured pattern and lives in a statically configured directory, get the payload and send the payload to a JMS Topic. Once done it will re-set the Batch Local File OTD and will resume the loop looking for the next file. The collaboration will exit normally only if a FileNotFound condition is encountered. If an exception occurs the collaboration will terminate with an exception.

Processing hundreds or thousands of files may take a rather long time, even if all the collaboration is doing is getting the payload and sending it to a JMS destination. During that time another trigger may arrive. To prevent another copy of the collaboration from starting the JMS Queue read will be serialized. To prevent triggers building up the trigger sender will configure an expiration time for each trigger. If not picked up before expiration the trigger will be expected to be discarded by the JMS implementation and never delivered.

In the solution discussed in this document triggers will be generated by a Scheduler eWay-triggered Java Collaboration, which will compose a JMS message, configure the Expiry property and submit the trigger message to the JMS Queue.

The Connectivity Map for the overall solution is shown in Figure 1.

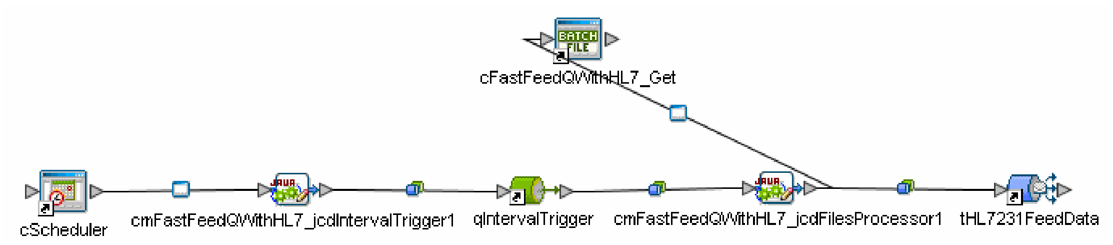

cFastFeedQWithHL7_Get
cScheduler
cmFastFeedQWithHL7_jcdIntervalTrigger1
qIntervalTrigger
cmFastFeedQWithHL7_jcdFilesProcessor1
tHL7231FeedData

Figure 1 Connectivity Map

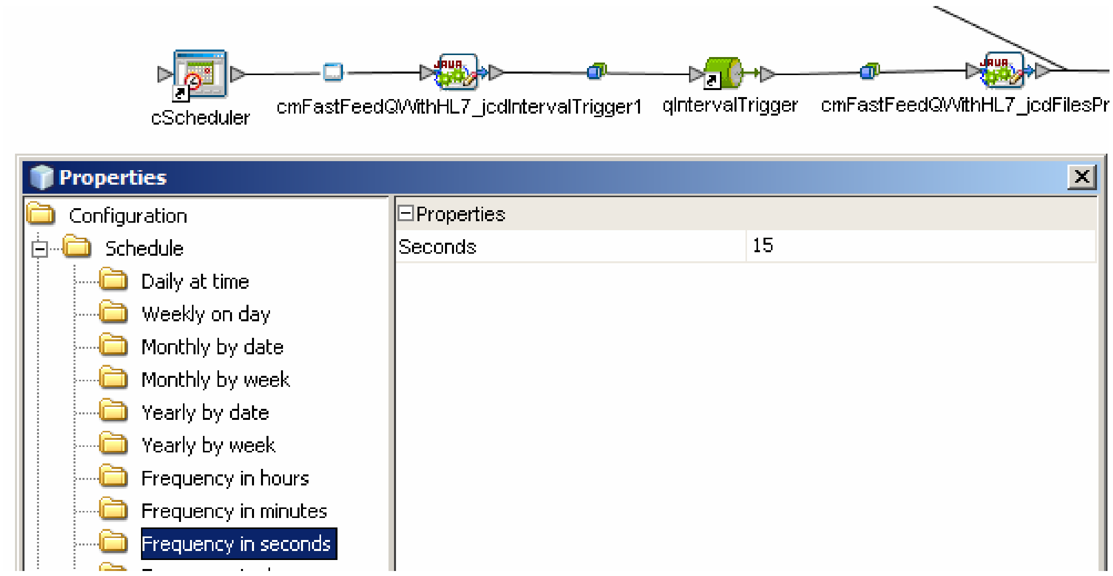
jcdIntervalTrigger

The jcdIntervalTrigger, shown in Listing 1, has trivial logic. Most of this logic is devoted to logging a timestamp to server.log.

Listing 1 jcdIntervalTrigger logic

```
public void start
        (com.stc.schedulerotd.appconn.scheduler.FileTextMessage input
        , com.stc.connectors.jms.JMS W_JMS)
            throws Throwable {

    long lTime = System.currentTimeMillis();
    java.util.Date dtNow = new java.util.Date(lTime);
    logger.debug("\n===>>> Triggered at " + dtNow);

    com.stc.connectors.jms.Message jmsMsg = W_JMS.createTextMessage();
    jmsMsg.getMessageProperties().setExpiration(20 * 1000); // 20 seconds
    jmsMsg.setTextMessage("Trigger, " + System.currentTimeMillis() + "," + dtNow);

    W_JMS.send(jmsMsg);
}
```

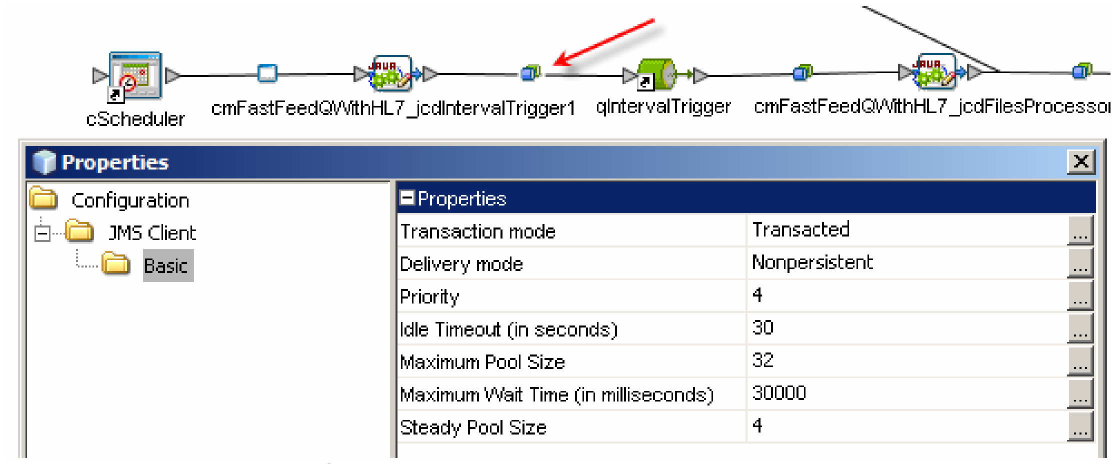
The Scheduler connectivity map connector, cScheduler, is configured for a 15 second interval, as shown in Figure 2. This interval can be different.



Seconds 15

Figure 2 Scheduler configured for a 15 second interval

The JMS Publish connector is configured to be transactional (as distinct from XA) non-persistent, as shown in Figure 3. We don’t mind missing a few triggers.


cScheduler
cmFastFeedQWithHL7_jcdIntervalTrigger1
qIntervalTrigger
cmFastFeedQWithHL7_jcdFilesProcessor
Properties
Configuration
JMS Client
Basic

Properties

Property	Value
Transaction mode	Transacted
Delivery mode	Nonpersistent
Priority	4
Idle Timeout (in seconds)	30
Maximum Pool Size	32
Maximum Wait Time (in milliseconds)	30000
Steady Pool Size	4

Figure 3 JMS Publisher configurations

Nothing on the publishing side determines that messages are to be serialized.

The jcdIntervalTrigger collaboration will deliver a message to the JMS Queue qIntervalTrigger every 15 seconds.

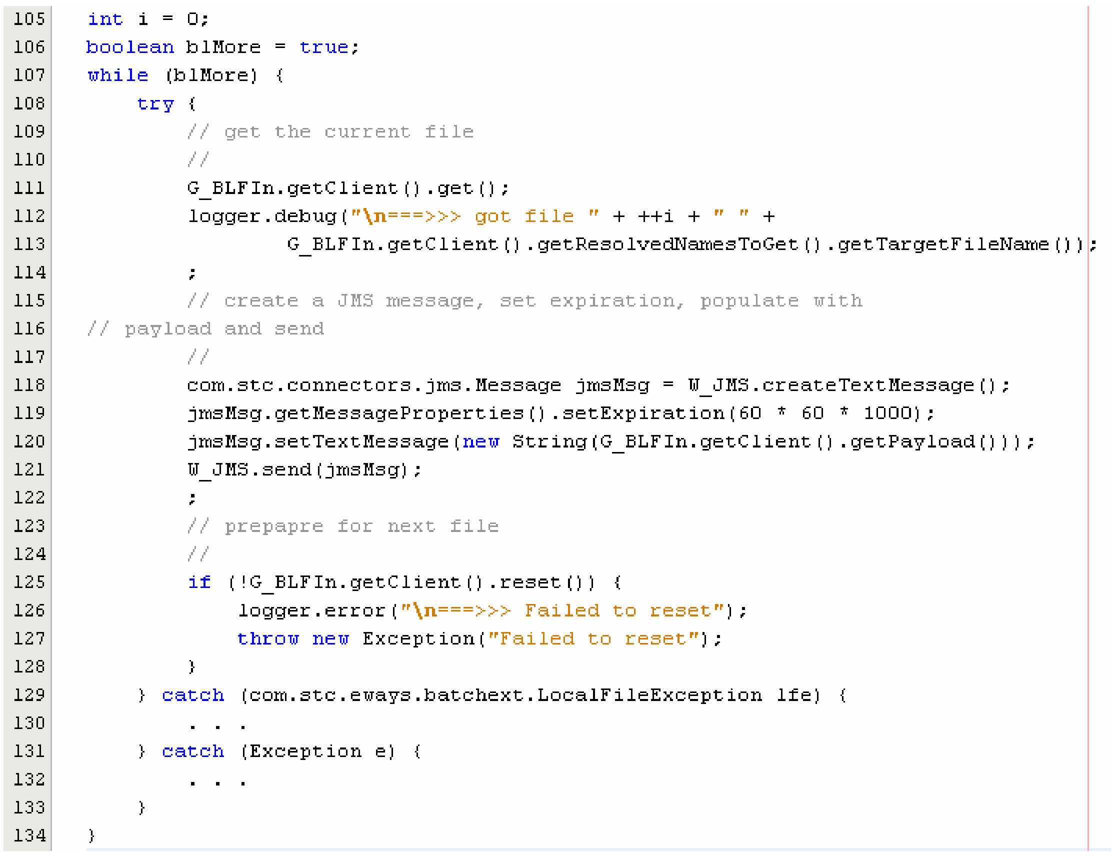
jcdFilesProcessor

jcdFilesProcessor is where interesting stuff happens. The logic was briefly described in Solution Outline. Listing 2 shows the complete collaboration.

Listing 2 jcdFilesProcessor

```
import com.stc.eways.batchext.LocalFileException;
import com.stc.eways.common.eway.standalone.streaming.StreamingException;
import com.stc.eways.batchext.BatchException;
import java.io.FileNotFoundException;

public class jcdFilesProcessor {

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive
            (com.stc.connectors.jms.Message input
            , com.stc.eways.batchext.BatchLocal G_BLFIn
            , com.stc.connectors.jms.JMS W_JMS)
                throws Throwable {

        long lNow = System.currentTimeMillis();
        java.util.Date dtNow = new java.util.Date(lNow);
        logger.debug("\n===>>> Received trigger "
                + input.getTextMessage() + " at " + lNow + ", " + dtNow);
        ;
        int i = 0;
        boolean blMore = true;
        while (blMore) {
            try {
                // get the current file
                //
                G_BLFIn.getClient().get();
                logger.debug("\n===>>> got file " + ++i + " " +
G_BLFIn.getClient().getResolvedNamesToGet().getTargetFileName());
                ;
                // create a JMS message, set expiration, populate with
                // payload and send
                //
                com.stc.connectors.jms.Message jmsMsg = W_JMS.createTextMessage();
                jmsMsg.getMessageProperties().setExpiration(60 * 60 * 1000);
                jmsMsg.setTextMessage(new String(G_BLFIn.getClient().getPayload()));
                W_JMS.send(jmsMsg);
                ;
                // prepapre for next file
                //
                if (!G_BLFIn.getClient().reset()) {
                    logger.error("\n===>>> Failed to reset");
                    throw new Exception("Failed to reset");
                }
            } catch (com.stc.eways.batchext.LocalFileException lfe) {
                //
                // File Not Found is expected and benign
                // That exception is so deeply nexted that the following code
                // is needed to determine if this is what cause the excetion
                //
                if (lfe.getNestedException() instanceof LocalFileException) {
                    LocalFileException lfe1
                            = (LocalFileException) lfe.getNestedException();

                    if (lfe1.getNestedException() instanceof StreamingException) {
                        StreamingException lfe2
                                = (StreamingException) lfe1.getNestedException();

                        if (lfe2.getNestedException() instanceof BatchException) {
                            BatchException lfe3
                                    = (BatchException) lfe2.getNestedException();

                            if (lfe3.getNestedException()
                                             instanceof FileNotFoundException) {
                                FileNotFoundException lfe4
                                        = (FileNotFoundException)
                                              lfe3.getNestedException();

                                logger.error
                              ("\n===>>> Ignoring expected File Not Found Exception: "
                                        + lfe4.getMessage());
                                blMore = false;
                            } else {
                                logger.error("\n===>>> Unexpected LocalFileException:"
                                        + lfe.getClass() + "\n", lfe);
                                blMore = false;
                            }
                        } else {
                            logger.error("\n===>>> Unexpected LocalFileException: "
                                    + lfe.getClass() + "\n", lfe);
                            blMore = false;
                        }
                    } else {
                        logger.error("\n===>>> Unexpected LocalFileException: "
                                + lfe.getClass() + "\n", lfe);
                        blMore = false;
                    }
                } else {
                    logger.error("\n===>>> Unexpected LocalFileException: "
                            + lfe.getClass() + "\n", lfe);
                    blMore = false;
                }
            } catch (Exception e) {
                logger.error("\n===>>> Exception getting file "
                        + e.getCause() + "\n", e);
                blMore = false;
            }
        }
    }
}
```

Let’s ignore, for the moment, the large block of exception handling code and look at the file processing loop. Figure 4 show the abbreviated logic.



```
105  int i = 0;
106  boolean blMore = true;
107  while (blMore) {
108      try {
109          // get the current file
110          //
111          G_BLFIn.getClient().get();
112          logger.debug("\n===>>> got file " + ++i + " " +
113                  G_BLFIn.getClient().getResolvedNamesToGet().getTargetFileName());
114          ;
115          // create a JMS message, set expiration, populate with
116  // payload and send
117          //
118          com.stc.connectors.jms.Message jmsMsg = W_JMS.createTextMessage();
119          jmsMsg.getMessageProperties().setExpiration(60 * 60 * 1000);
120          jmsMsg.setTextMessage(new String(G_BLFIn.getClient().getPayload()));
121          W_JMS.send(jmsMsg);
122          ;
123          // prepapre for next file
124          //
125          if (!G_BLFIn.getClient().reset()) {
126              logger.error("\n===>>> Failed to reset");
127              throw new Exception("Failed to reset");
128          }
129      } catch (com.stc.eways.batchext.LocalFileException lfe) {
130          . . .
131      } catch (Exception e) {
132          . . .
133      }
134  }
```

Figure 4 Abbreviated file processing logic

The line numbers don't correspond to source lines. They are here merely for the convenience of reference.

The loop starts at line 107.

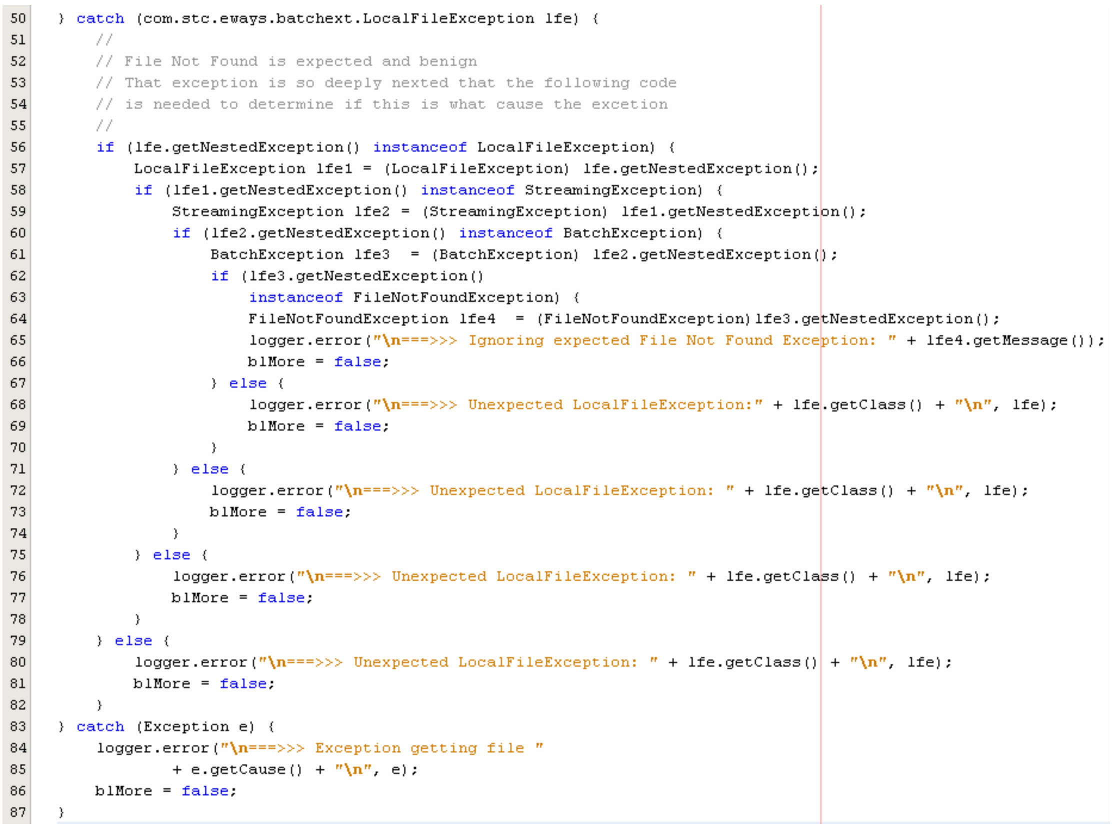
The logic is surrounded by try-catch. We need to exit the loop normally when FielNotFoundException is encountered and abnormally when any other exception is encountered. FileNotFoundException is a nested exception. We will discuss how to work out if this exception occurred a little later.

At line 111 we issue the GET command to the Adapter. At this point we can obtain the actual expanded name of the file. As we will see later, we don't know what the name of the file being processed in this loop iteration might as we enter the loop iteration be because the Adapter is configured to look for files using a regular expression.

On lines 118 through 121 we construct, populate and send a JMS message. At line 120 we get file payload as bytes, convert them to a string and set them as JMS message payload. Once could also use JMS BytesMessage instead of JMS TextMessage and avoid conversion.

On line 125 we attempt to reset the OTD so as to be able to process the next file. This is the critical step that makes it possible to process multiple files in a single invocation. If successful, we go back to the top of the loop to process the next file.

Because the FileNotFoundException exception is a nested exception, and it is nested 4 levels deep, the logic required to determine if it occurred is larger then the logic required to do the work of the collaboration. Figure 5 shows the exception processing logic.



```
50   } catch (com.stc.eways.batchext.LocalFileException lfe) {
51       //
52       // File Not Found is expected and benign
53       // That exception is so deeply nexted that the following code
54       // is needed to determine if this is what cause the excetion
55       //
56       if (lfe.getNestedException() instanceof LocalFileException) {
57           LocalFileException lfe1 = (LocalFileException) lfe.getNestedException();
58           if (lfe1.getNestedException() instanceof StreamingException) {
59               StreamingException lfe2 = (StreamingException) lfe1.getNestedException();
60               if (lfe2.getNestedException() instanceof BatchException) {
61                   BatchException lfe3  = (BatchException) lfe2.getNestedException();
62                   if (lfe3.getNestedException()
63                       instanceof FileNotFoundException) {
64                       FileNotFoundException lfe4  = (FileNotFoundException)lfe3.getNestedException();
65                       logger.error("\n===>>> Ignoring expected File Not Found Exception: " + lfe4.getMessage());
66                       blMore = false;
67                   } else {
68                       logger.error("\n===>>> Unexpected LocalFileException:" + lfe.getClass() + "\n", lfe);
69                       blMore = false;
70                   }
71               } else {
72                   logger.error("\n===>>> Unexpected LocalFileException: " + lfe.getClass() + "\n", lfe);
73                   blMore = false;
74               }
75           } else {
76               logger.error("\n===>>> Unexpected LocalFileException: " + lfe.getClass() + "\n", lfe);
77               blMore = false;
78           }
79       } else {
80           logger.error("\n===>>> Unexpected LocalFileException: " + lfe.getClass() + "\n", lfe);
81           blMore = false;
82       }
83   } catch (Exception e) {
84       logger.error("\n===>>> Exception getting file "
85               + e.getCause() + "\n", e);
86       blMore = false;
87   }
```

Figure 5 Exception handling logic

Inspect the code in Listing 2 to see more clearly what is happening.

On line 50 we are catching the BatchLocalFileException exception. This is the outermost exception that is thrown when the java.io.FileNotFoundException is thrown.

On line 57 we get a nested exception and on line 58 we look to see if it is a StreamingException, which too is an outer exception to the java.io.FileNotFoundException which we are looking for.

On line 59 we get a nested exception and on line 60 we look to see if it is a BatchException, which too is an outer exception to the java.io.FileNotFoundException which we are looking for.

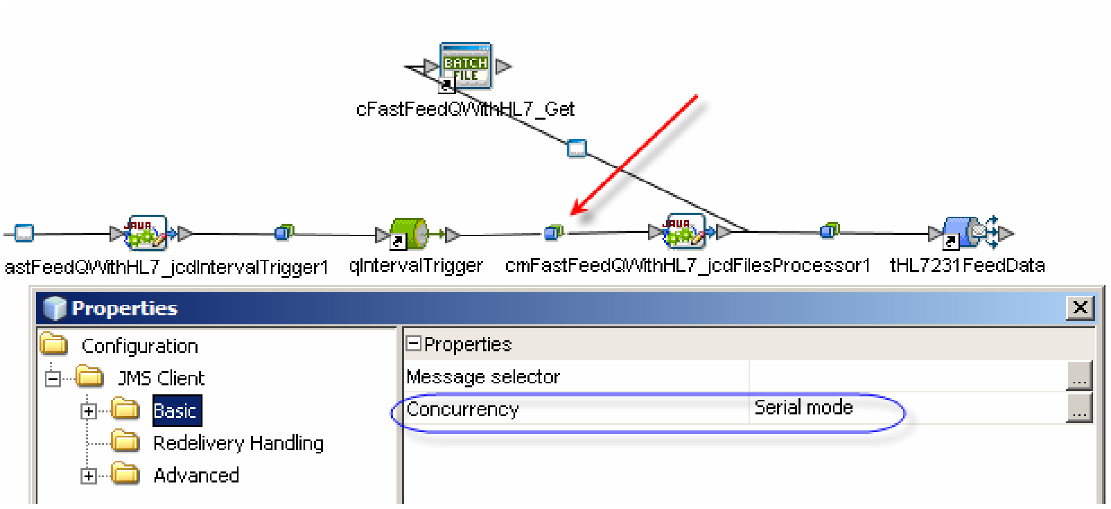
On line 61 we get the innermost nested exception.

On lines 62 through 66 we finally determine that we have the java.io.FileNotFoundException, which we expect to occur once all files are processed, and we handle it by logging a message and setting the Boolean to exit the loop.

In all other cases we log a message and set the Boolean to exit the loop. Perhaps we should have re-thrown an exception since only FileNotFoundException is a benign one, all other being unexpected and not desirable.

Let’s now take a look at the connector properties in the connectivity map.

Serial Mode concurrency at the receiver side will ensure that only one message will get picked up from the queue at a time. So long as the collaboration executes, processing files, no new message will get picked up so no new instance of the collaboration will try to process the same files an existing instance is already processing. Figure 6 shows the property setting.



Message selector

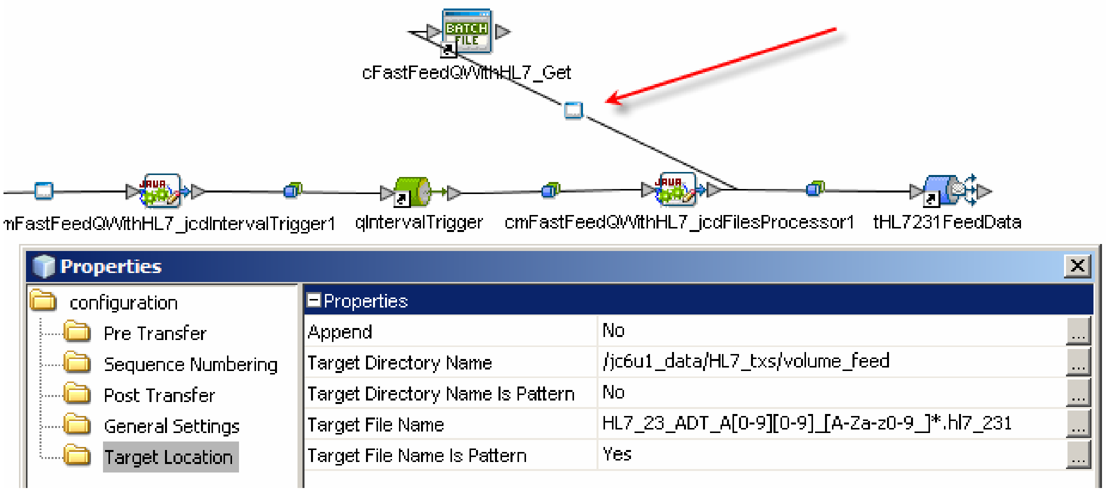
Concurrency Serial mode

Figure 6 Serial Mode concurrency at the receiver

The Batch Local File connector is configured to look in directory /jc6u1_data/HL7_txs/volume_feed for files whose names satisfy regular expression HL7_23_ADT_A[0-9][0-9]_[A-Za-z0-9_]*.hl7_231. Some examples of this kind of names are:

HL7_23_ADT_A03_ARMC_460374__06374.hl7_231
HL7_23_ADT_A03_ARMC_460390__06308.hl7_231
HL7_23_ADT_A03_ARMC_555555__06390.hl7_231
HL7_23_ADT_A03_ARMC_6699473__06385.hl7_231
HL7_23_ADT_A03_ARMC_782111073__06383.hl7_231
HL7_23_ADT_A03_ARMC_7899473__06384.hl7_231
HL7_23_ADT_A03_BIGH_002664__05942.hl7_231
HL7_23_ADT_A03_BIGH_1111111__06392.hl7_231
HL7_23_ADT_A03_BIGH_222222__06387.hl7_231
HL7_23_ADT_A03_STC_777777__06391.hl7_231
HL7_23_ADT_A03_STC_9999999999__06395.hl7_231

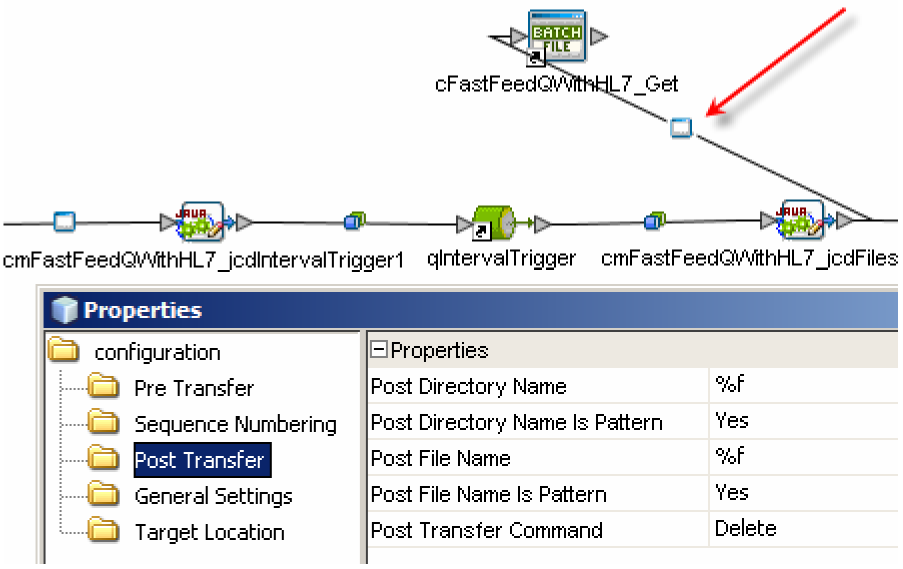
Figure 7 shows the configuration of the Target Location properties.



Properties	
Append	No
Target Directory Name	/jc6u1_data/HL7_txs/volume_feed
Target Directory Name Is Pattern	No
Target File Name	HL7_23_ADT_A[0-9][0-9]_[A-Za-z0-9_]*.hl7_231
Target File Name Is Pattern	Yes

Figure 7 Target Location properties

I copy files to be picked up from another directory so rather then preserving them I delete them. Post Transfer properties ensure that files are deleted once processed. Figure 8 shows these properties.



Properties	
Post Directory Name	%f
Post Directory Name Is Pattern	Yes
Post File Name	%f
Post File Name Is Pattern	Yes
Post Transfer Command	Delete

Figure 8 Post Transfer properties

The final component in the connectivity map is a JMS Topic. There is no current subscriber to this topic and there are no durable subscribers so all messages that are sent to it will be silently discarded by the JSM implementation. This is what I need for this example in order to see the behaviour of the file processor without having to concern myself with cleaning up or processing hundreds of messages. A proper solution would actually do something with the messages.

Summary

Occasionally one needs to pick up and process a large number of files, on the order of hundreds or thousands. With the Batch Inbound eWay/JCA Adapter it is not possible to pick up more then one file per poll. The Batch Local File, if triggered by some event other then an appearance of a file in a directory, perhaps a Scheduler trigger of a manual trigger, with correctly designed logic can process many files in a single invocation.

This document discussed a Batch Local File-based solution that effectively processes hundreds of files in a single pass.

An archive with 660 HL7 v2 files and the project export is available for download at http://mediacast.sun.com/users/Michael.Czapski-Sun/media/ProcessingHundredsOfFileWithBatchAdapter.zip/details