Streaming Large FTP Transfers
using Java CAPS 5.1.x and 6

Michael.Czapski@sun.com, April 2008

TABLE OF CONTENTS

1 Introduction

Transferring large payloads, on the order of tens or hundreds of megabytes, between a FTP server and a local file system, in either direction, requires selection of appropriate features of the Batch FTP and Batch Local File eWays, and tuning certain timing parameters.

Default timing parameter values result in timeout exceptions when transferring large payloads.

The Batch FTP eWay and the Batch Local File eWas are typically used to receive the entire payload before writing it out. This results in attempts to allocate memory many time the size of the payload being transferred and, for large files, causes memory exhaustion and application server failures.

This text points out which timing parameters need to be tuned to facilitate transfer of large payloads. It also presents sample Java code that uses facilities of the Batch FTP and Batch Local File for streaming payload between the FTP server and the local file systems without using excessive amount of memory.

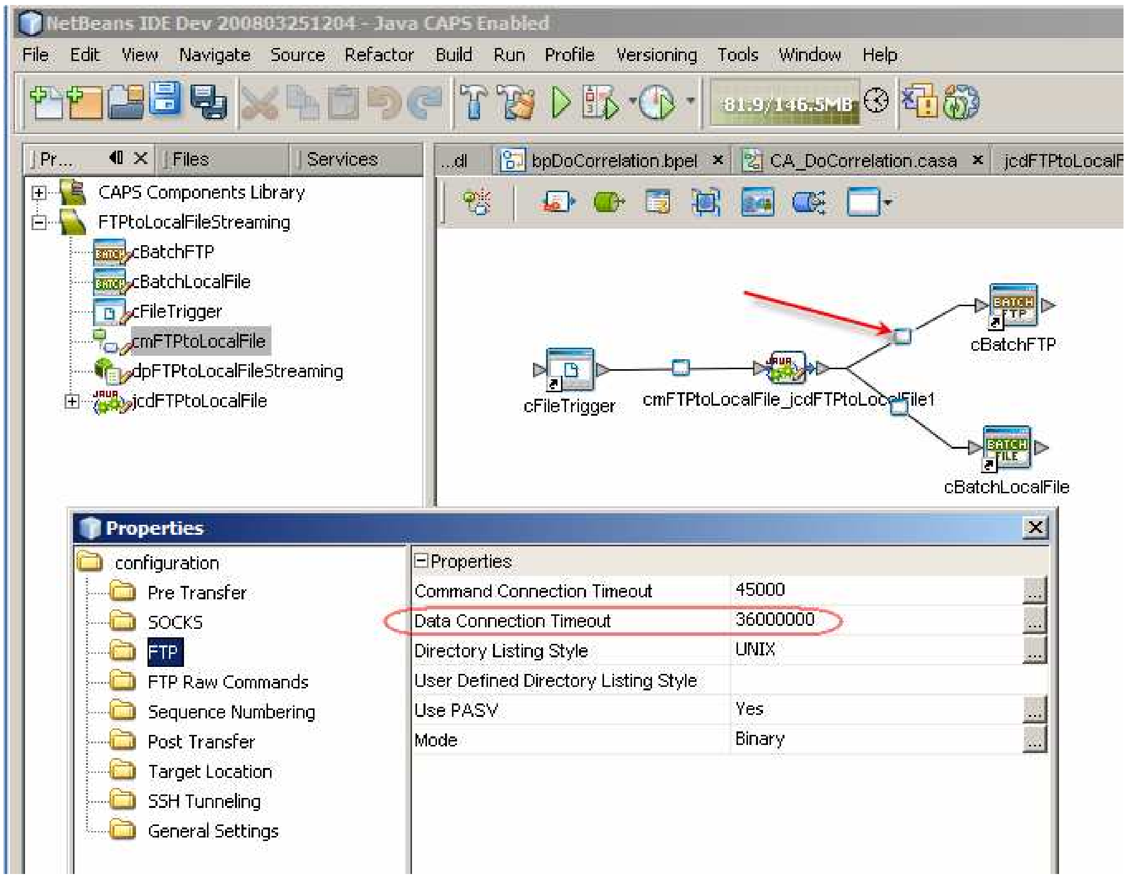
The material covered here was prepared using Java CAPS projects developed and tested in Java CAPS 5.1.0, exported, imported into Java CAPS Release 6 and tested again. It is expected that the code presented below will work in all versions of Java CAPS from 5.1.0 up.

2 Batch FTP eWay Read Timing Parameters

Reading and writing large payloads using the Batch FTP eWay may result in read and socket timeouts. To address this issue the eWay timers must be increased to accommodate longer transfer times.

The Batch FTP eWay External System Container does not have any transfer-related timer properties.

The Batch FTP eWay Connector in the Connectivity Map, shown in Figure 1 in the Java CAPS Release 6 IDE but also the same in Java CAP 5.1.x, has a configurable Data Connection Timeout property.



Properties	
Command Connection Timeout	45000
Data Connection Timeout	36000000
Directory Listing Style	UNIX
User Defined Directory Listing Style	
Use PASV	Yes
Mode	Binary

Figure 1 *Data Connection Timeout Batch FTP eWay Connector property*

Setting the Data Connection Timeout property on the Batch FTP eWay Connector to a value larger then the default 45 seconds does not eliminate timeout exceptions if the transfer time exceeds 45 seconds. This is because there are two other properties, which can only be configured programmatically, that affect transfer timeouts.

Assuming the Batch FTP eWay is added to a Java Collaboration and named vFTPIn, the three properties being set in Listing 1 must be configured to the appropriate values to ensure timeout exception does not occur. Here *iTimeoutMillis* is a class variable set to the desired timeout value.

Listing 1 *Timeout properties*

```
vFTPIn.getConfiguration().setDataConnectionTimeout( iTimeoutMillis );
vFTPIn.getProvider().setDataSocketTimeout( iTimeoutMillis );
vFTPIn.getProvider().setSoTimeout( iTimeoutMillis );
```

The first property, DataConnectionTimeout is the same as the Data Connection Timeout property of the connector in the Connectivity Map. The other two properties can only be set programmatically.

3 Streaming from FTP Server to Local File System

Data streaming features of the Batch FTP and the Batch Local File eWays can be used to stream data from the source to the destination. This is different from what one would normally do, that is get the entire payload then put it to the file system, which if fine for small payloads but disaster-prone for large payloads.

Streaming keeps memory consumption to a minimum and allows transfer of arbitrarily large payloads. Relevant statements from a Java Collaboration are reproduced in Listing 2. The project, of which this code is a part, is provided as an addendum to this document.

Listing 2 *Java code for streaming FTP to Local File System transfer*

```
com.stc.eways.common.eway.standalone.streaming.OutputStreamAdapter osa = null;
osa = vLocalFileOut.getClient().getOutputStreamAdapter();
vFTPIn.getClient().setOutputStreamAdapter( osa );
vFTPIn.getClient().get();
```

We first set the Batch FTP OTD instance’s OutputStreamAdapter to the value of the Batch Local File OTD instance’s OutputStreamAdapter. This will cause the Batch FTP eWay to stream data to the Batch Local File.

The Batch FTP OTD Client’s get() method causes the streaming operation to be executed.

That’s it! This code will stream arbitrarily large payloads from a FTP server to the local file system.

The four lines of code could have been compressed to two by directly providing the Local File OTD’s OutputStreamAdapter to the FTP OTD’s OutputStreamAdapter setter. The code is shown over multiple lines for readability.

Listing 3 shows a sample Java Collaboration that hardcodes the file names and directory paths, configures read timeout properties, executes streaming transfer and logs some timing statistics.

Listing 3 *Sample FTP to Local File Streaming Collaboration*

```
public void receive
    ( com.stc.connector.appconn.file.FileTextMessage input
    , com.stc.eways.batchext.BatchFtp vFTPIn
    , com.stc.eways.batchext.BatchLocal vLocalFileOut )
        throws Throwable
{
    try {
        long lStartMillis = System.currentTimeMillis();
        int iTimeoutMillis = 40 * 60 * 1000;
        ;
        vFTPIn.getConfiguration().setTargetDirectoryName( "//" );
        vFTPIn.getConfiguration().setTargetDirectoryNameIsPattern( false );
        vFTPIn.getConfiguration().setTargetFileName( "eGate_User_Guide_SRE.pdf" );
        vFTPIn.getConfiguration().setTargetFileName( "JavaCAPS.exe" );
        vFTPIn.getConfiguration().setTargetFileName( "JCAPS6_M8_clean_new.7z" );
        vFTPIn.getConfiguration().setTargetFileNameIsPattern( false );
        ;
        String sFileName = vFTPIn.getConfiguration().getTargetFileName();
        ;
        vLocalFileOut.getConfiguration().setTargetFileName( sFileName );
        vLocalFileOut.getConfiguration().setTargetFileNameIsPattern( false );
        vLocalFileOut.getConfiguration().setTargetDirectoryName( "g:/____FileIn" );
        vLocalFileOut.getConfiguration().setTargetDirectoryNameIsPattern( false );
        vLocalFileOut.getConfiguration().setPostDirectoryName( "g:/____FileIn" );
        vLocalFileOut.getConfiguration().setPostDirectoryNameIsPattern( false );
        vLocalFileOut.getConfiguration().setPostFileName( "%f.done" );
        vLocalFileOut.getConfiguration().setPostFileNameIsPattern( true );
        vLocalFileOut.getConfiguration().setPostTransferCommand( "rename" );
        vLocalFileOut.getConfiguration().setResumeReadingEnabled( true );
        logger.debug( "\n===>>>" + vLocalFileOut.getConfiguration().toString() );
        ;
        ;
        vFTPIn.getConfiguration().setDataConnectionTimeout( iTimeoutMillis );
        vFTPIn.getProvider().setDataSocketTimeout( iTimeoutMillis );
        vFTPIn.getProvider().setSoTimeout( iTimeoutMillis );
        ;
        com.stc.eways.common.eway.standalone.streaming.OutputStreamAdapter osa = null;
        osa = vLocalFileOut.getClient().getOutputStreamAdapter();
        vFTPIn.getClient().setOutputStreamAdapter( osa );
        vFTPIn.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.debug( "\n===>>> Execution took " + lMillis + " Milliseconds" );
        logger.debug( "\n===>>> Execution took " + iSeconds + " Seconds" );
        logger.debug( "\n===>>> Execution took " + iMinutes + " Minutes" );
    } catch ( Exception e ) {
        logger.error( "Exception somewhere performing transfer", e );
        return;
    }
    ;
}
```

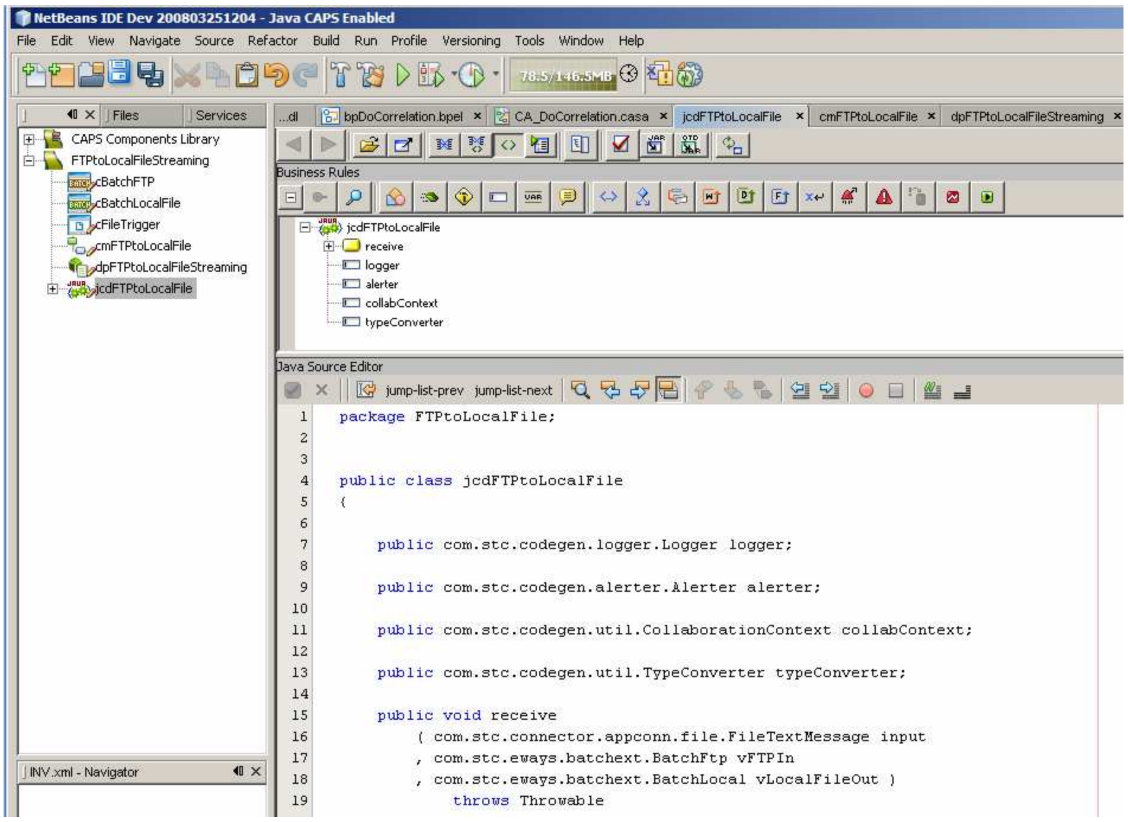
Rather then hardcoding file and path values you can statically configure them through the Connectivity Map Connectors for the Batch FTP and Batch Local File eWay.

The Collaboration is triggered by a File eWay. A presence of a named file will cause the collaboration to start, configure the FTP eWay, configure the Local File eWay, execute transfer and log statistics. The collaboration could have been triggered by a Scheduler eWay, a JMS message or some other means that are appropriate to your solution.

Note that while the pre- and post-transfer activities are not performed in the FTP eWay some string must be statically configured for the pre- and post-transfer file name and directory in the FTP eWay connector to avoid warnings at runtime.

Note that while we dynamically configure the file name and the directory it is necessary to also specify some string for the target file name and the target directory properties of the FTP eWay connector to avoid runtime error.

As a teaser, Figure 2 shows what a Project Explorer and the Java Collaboration Editor look like in Java CAPS Release 6 ☺


NetBeans IDE Dev 200803251204 - Java CAPS Enabled
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Files Services
CAPS Components Library
FTPtoLocalFileStreaming
cBatchFTP
cBatchLocalFile
cFileTrigger
cmFTPtoLocalFile
dpFTPtoLocalFileStreaming
jcdFTPtoLocalFile
INV.xml - Navigator
...dl bpDoCorrelation.bpel CA_DoCorrelation.casa jcdFTPtoLocalFile cmFTPtoLocalFile dpFTPtoLocalFileStreaming
Business Rules
jcdFTPtoLocalFile
receive
logger
alerter
collabContext
typeConverter
Java Source Editor
jump-list-prev jump-list-next

```
package FTPtoLocalFile;


public class jcdFTPtoLocalFile
{

    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.connector.appconn.file.FileTextMessage input
        , com.stc.eways.batchext.BatchFtp vFTPIn
        , com.stc.eways.batchext.BatchLocal vLocalFileOut )
            throws Throwable
```

Figure 2 *Java Collaboration Editor in Java CAPS Release 6*

Java CAPS Release 6 supports both the “CAPS Repository-based” projects, which is what the 5.1.x style projects are called, and the regular, repository-less projects.

This project was developed and exercised in Java CAPS 5.1.0, exported, imported into Java CAPS Release 6, Milestone 8, built and exercised. No changes were required to make this happen.

4 Streaming from Local File System to FTP Server

Relevant statements from a Java Collaboration used to stream data from a file in a local file system to an FTP Server are reproduced in Listing 4. The Java CAPS Release 6 CAPS Repository-based project, of which this code is a part, is provided as an addendum to this document.

Listing 4 *Java code for streaming Local File to FTP transfer*

```
com.stc.eways.common.eway.standalone.streaming.InputStreamAdapter isa = null;
isa = vLocalFileIn.getClient().getInputStreamAdapter();
vFTPOut.getClient().setInputStreamAdapter( isa );
vFTPOut.getClient().put();
```

We first set the Batch FTP OTD instance’s InputStreamAdapter to the value of the Batch Local File OTD instance’s InputStreamAdapter. This will cause the Batch FTP eWay to stream data from the Batch Local File.

The Batch FTP OTD Client’s put() method causes the streaming operation to be executed.

That's it! This code will stream arbitrarily large payloads from a local file system to a FTP Server.

The four lines of code could have been compressed to two by directly providing the Local File OTD's InputStreamAdapter to the FTP OTD's InputStreamAdapter setter. The code is shown over multiple lines for readability.

Listing 5 shows a sample Java Collaboration that hardcodes the file names and directory paths, configures transfer timeout properties, executes streaming transfer and logs some timing statistics.

Listing 5 *Sample Local File to FTP Streaming Collaboration*

```
public void receive
    ( com.stc.connector.appconn.file.FileTextMessage input
    , com.stc.eways.batchext.BatchLocal vLocalFileIn
    , com.stc.eways.batchext.BatchFtp vFTPOut )
        throws Throwable
{
    long lStartMillis = System.currentTimeMillis();
    int iTimeoutMillis = 40 * 60 * 1000;

    try {
        vLocalFileIn.getConfiguration().setTargetFileName( "eGate_User_Guide_SRE.pdf.done" );
        vLocalFileIn.getConfiguration().setTargetFileName( "JavaCAPS.exe.done" );
        vLocalFileIn.getConfiguration().setTargetFileName( "JCAPS6_M8_clean_new.7z.done" );
        vLocalFileIn.getConfiguration().setTargetFileNameIsPattern( false );
        vLocalFileIn.getConfiguration().setTargetDirectoryName( "g:/____FileIn" );
        vLocalFileIn.getConfiguration().setTargetDirectoryNameIsPattern( false );
        vLocalFileIn.getConfiguration().setPostDirectoryName( "g:/____FileIn" );
        vLocalFileIn.getConfiguration().setPostDirectoryNameIsPattern( false );
        vLocalFileIn.getConfiguration().setPostFileName( "%f.done" );
        vLocalFileIn.getConfiguration().setPostFileNameIsPattern( true );
        vLocalFileIn.getConfiguration().setPostTransferCommand( "rename" );
        vLocalFileIn.getConfiguration().setResumeReadingEnabled( true );
        logger.debug( "\n===>>>" + vLocalFileIn.getConfiguration().toString() );

        String sFileName = vLocalFileIn.getConfiguration().getTargetFileName();

        vFTPOut.getConfiguration().setTargetDirectoryName( "//" );
        vFTPOut.getConfiguration().setTargetDirectoryNameIsPattern( false );
        vFTPOut.getConfiguration().setTargetFileName(sFileName);
        vFTPOut.getConfiguration().setTargetFileNameIsPattern( false );

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

        com.stc.eways.common.eway.standalone.streaming.InputStreamAdapter isa = null;
        isa = vLocalFileIn.getClient().getInputStreamAdapter();
        vFTPOut.getClient().setInputStreamAdapter( isa );
        vFTPOut.getClient().put();

        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.debug( "\n===>>> Execution took " + lMillis + " Milliseconds" );
        logger.debug( "\n===>>> Execution took " + iSeconds + " Seconds" );
        logger.debug( "\n===>>> Execution took " + iMinutes + " Minutes" );
    } catch ( Exception e ) {
        logger.error( "Exception somewhere performing transfer", e );
        return;
    }
}
```

Rather then hardcoding file and path values you can statically configure them through the Connectivity Map Connectors for the Batch FTP and Batch Local File eWay.

The Collaboration is triggered by a File eWay. A presence of a named file will cause the collaboration to start, configure the Local File eWay, configure the FTP eWay, execute transfer and log statistics. The collaboration could have been triggered by a Scheduler eWay, a JMS message or some other means that are appropriate to your solution.

Note that while the pre- and post-transfer activities are not performed in the FTP eWay some string must be statically configured for the pre- and post-transfer file name and directory in the FTP eWay connector to avoid warnings at runtime.

Note that while we dynamically configure the file name and the directory it is necessary to also specify some string for the target file name and the target directory properties of the FTP eWay connector to avoid runtime error.

5 Empirical Timings

The FTP to Local File streaming solution, both the Java CAPS 5.1.0 project and the Java CAPS Release 6 CAPS Repository-based project were exercised with three files of different sizes. The environment was Dell Latitude D600, 1.6 GHz, 2Gb Memory, 2 disks, Windows XP SP2, FreeFTPd 1.0.11 (http://www.freeftpd.com/).

Timings for FTP Server to Local File System transfers are presented in Table 1.

File Size in Kb	File Size in Mb	Transfer Time Milliseconds	Transfer Time Seconds	Transfer Time Minutes
	Java CAPS 5.1.0 project			
7,451	7	3,045	3	< 1
631,398	631	189,072	189	3
1,010,666	1,010	421,216	421	7
	Java CAPS Release 6 CAP Repository-based project			
7,451	7	3,005	3	< 1
631,398	631	167,450	167	2
1,010,666	1,010	298,079	298	4

Table 1 *FTP Server to Local File System streaming timings*

Timings for Local File System to FTP Server transfers are presented in Table 2.

File Size in Kb	File Size in Mb	Transfer Time Milliseconds	Transfer Time Seconds	Transfer Time Minutes
	Java CAPS Release 6 CAP Repository-based project			
7,451	7	1,482	1	< 1

631,398	631	145,289	145	2
1,010,666	1,010	220,306	220	3

Table 2 *Local File System to FTP Server streaming timings*

Note that timings in minutes are truncated so 4 minutes may well be just less than 5 minutes.

These empirical findings are not representative of all environments.

Note that the JVM memory consumption did vary noticeably before, during or after transfer even when a 1Gb file was being transferred.

6 Summary

Java CAPS can be used to stream very large payloads between FTP servers and the local file system, in both directions, without causing timeout exceptions and exhausting JVM memory.

Sample projects discussed in this paper illustrate the Java code required to engage appropriate Batch FTP and Batch Local File eWay facilities to effect very large transfers in streaming mode.

As a side benefit it was shown that a Java CAPS 5.1.0 project can be imported “as is” into the new Java CAPS Release 6 environment, built and executed without changes.