Co:Z Co-Processing Toolkit

z/OS Hybrid Batch Processing: Generating a multi-page PDF document with Co:Z

V 1.0 Edition

Published April, 2012

Introduction

A common function required by enterprise application software is to generate PDF documents tailored with data obtained from corporate databases. This function has at least the following inputs: a template PDF, and a data set of customer information. The output is a large PDF document with a page (or a few pages) for each customer, merged with the customer data. This function can require significant memory and CPU resources to complete.

While this type of application can easily be implemented to run completely as a z/OS batch job, it is enticing within the IBM zEnterprise hybrid processor architecture to consider moving PDF generation to a zBX blade. Assuming that the desired program can be relocated to a z/BX blade server running AIX, Linux, or Windows, what are the other considerations for moving all or part of this job?

- How will the program access required input and output data?
- How will the program be scheduled, possibly so as to fit into the context of a larger job stream?
- What operational impact will there be by moving all or part of the job? (log messages, monitoring, completion status, etc.)

The Co:Z Co-Processing Toolkit provides z/OS users with tools that mitigate these considerations while making it easy to exploit zBX blades from traditional batch workloads - an idiom we refer to as *z/OS hybrid batch processing*.

z/OS Hybrid Batch

We define *z/OS hybrid batch* as:

- The ability to execute a program or script on a distributed (target) system from a z/OS batch job
- The target program may already exist and should require little or no modification.
- The target program's input and output are redirected from/to z/OS spool files or data sets
- The target program may access other z/OS resources: DD's, data sets, UNIX files and programs
- The target program's exit code is adopted as the z/OS job step condition code.

Other hybrid architectures

Other models for zEnterprise hybrid computing are more recognized:

- zBX blades as user-facing edge servers, web servers, and application servers with z/OS providing back-end databases and transaction processing.
- zBX blades as special purpose appliances or optimizers.

The z/OS hybrid batch environment is a third model:

- zBX blades as general purpose application appliances to service z/OS batch workloads.

Of course, mixed models are not uncommon. For example: zBX blades as user-facing web/application servers with z/OS hybrid batch jobs scheduled to perform utility, integration, and maintenance processing on these servers. [1]

This article explores z/OS hybrid batch processing through a case study of an example application: PDF generation. Example source code and sample JCL and data are available from our web site so that you can try this application in your own environment. [2]

z/OS Hybrid Batch: PDF Generation Considerations

Before looking at the case study, consider the inputs and outputs of an example PDF generation application. Specifically, the following:

PDF Template

Typically, a "template" PDF is created and uploaded to z/OS. [3] This file may be stored as binary UNIX file or in a RECFM=U/V/VB data set, and contains named form fields that identify areas of a page that are to be customized. The sample data included in this case study is a single page customer letter with form fields for date, name, address, etc. Some fields, name and address for example, are populated from each customer record in the input data file. Other fields, date for example, are populated with a single value on all pages.

Field Data

This is a text file or z/OS dataset containing comma delimited data, which is assumed to be extracted from an z/OS database or converted from another source. Each row will be merged with the template PDF and the resulting page (or pages) will be appended to the output PDF. The sample data for this case study is customer name, address, state, etc.

Field Map

The field map is a text file or data set that contains a single comma delimited row. Each column contains the name of the form field that is populated with data from the Field Data data set value at the same column position.

Common Data

Common data values can be stored in a text file or data set. This file contains a single row of comma delimited values. Each column is a common data value that will be used to populate all merged PDF documents. For example: the sample template for this case study contains a date field. The sample common data file contains the date to be filled in on all pages of the generated PDF.

Common Map

The common map is also a text file or data set containing a single row of comma delimited columns. Each column contains the name of the form field that is populated with data from the common data file with the same column position.

Output PDF

[1]Also known as "Herding Penguins"

[2]A zEnterprise and zBX is not required to exploit z/OS hybrid batch applications.

[3]A template PDF can be created using Apache OpenOffice.org Writer. Refer to the *Creating a Simple Form* section in the Writer documentation for additional information. The names of Form Control Text Boxes are used by this example program when populating fields in the generated PDF. Once the Writer document is complete, used the Export to PDF option under the File menu to create a template PDF file.

This is the multi-page PDF document that is generated. It is stored as a binary file or data set, and will contain a copy of the template PDF, merged with the input field data - one for each Field Data row and in that order. On z/OS this may be stored as a RECFM=U/V/VB dataset or as a UNIX file.

It is a goal of any conversion to minimize the changes required to existing source programs and to keep operational mechanisms intact. For this case study, all inputs and outputs remain on z/OS. No code changes are required to the PDF generation code; we simply need to deploy the program to the target blade. [4] The following techniques and Co:Z Toolkit features are used to meet these goals:

Dataset Pipes Utilities

The Co:Z Toolkit Dataset Pipes **fromdsn**, **todsn**, **fromfile** and **tofile** commands have a rich set of options, making it possible to stream data from and to z/OS from the remote process in a variety of ways. These options can greatly simplify the transfer and treatment of input and output data.

Process substitution

When the Co:Z Launcher starts a process on a target UNIX or Windows system, it runs a shell by default. On these systems, **bash** (Bourne Again SHell) is typically available. Bash has a very useful feature called "process substitution" which is a concise syntax for replacing filenames with input and output redirection. This feature can be used with data set Pipes commands to simplify the target program's access to z/OS file and data set resources.

[4]It would be possible to download the program to the blade each time the job is run.

Case Study: Generating Multi-page Customized PDF

The PDF generator in this case study is a Java program with the main method defined in class com.dovetail.coz.itext.ItextPdfGenerator. This program uses the following open source libraries: *iText* for PDF functions, *openCsv* for parsing input delimited files and *Apache Log4J* for application logging. The source for the PDF generator, along with sample JCL and data, can be downloaded from our website: *http://dovetail.com*. The javadoc provides detail on program features as well as execution arguments and system properties. This case study focuses only on information required to understand the JCL for running the program on z/OS and with the Co:Z Launcher.

The program usage is the following:

```
Usage:
  com.dovetail.coz.itext.ItextPdfGenerator
     [--common-data <common-pipe-delimited.csv>]
     [--common-map <common-field-map.csv>]
     [--field-data <data-pipe-delimited.csv>]
     [--field-map <data-field-map.csv>]
     [--field-data-limit <data-row-limit>]
     [--pdf-template <template.pdf>]
     [--pdf-output <multipage.pdf>]
```

The program arguments are:

- `--common-data` for the Common Data file name
- `--common-map` for the Common Map file name
- `--field-data` for the Field Data file name
- `--field-map` for the Field Map file name
- `--pdf-template` for the Template PDF file name
- `--pdf-output` for the Output PDF file name

The Java program supports several system properties. Refer to the javadoc for a complete list. Default values are used for this case study.

The program may be run as a traditional z/OS batch job with the following JCL:

```
//ITEXTZOS JOB (),MSGCLASS=H,NOTIFY=&SYSUID
//PROCLIB JCLLIB ORDER=HLQ.COZ.JCL
//*
// SET PRFX=VENDOR.ITEXT.SAMPLE
//*
//DELOLD EXEC PGM=IEFBR14 ❶
//PDFOUT   DD DSN=&PRFX..PDFOUT,
//            DISP=(MOD,DELETE,DELETE),
//            SPACE=(CYL,1)
```

```
//*
//JAVA EXEC PROC=JVMPRC60,   ❷
// JAVACLS='com.dovetail.coz.itext.ItextPdfGenerator' ❸
//*
//FLDDATA  DD DISP=SHR,DSN=&PRFX..FLDDATA ❹
//COMDATA  DD DISP=SHR,DSN=&PRFX..CONFIG(COMDATA)
//COMMAP   DD DISP=SHR,DSN=&PRFX..CONFIG(COMMAP)
//FLDMAP   DD DISP=SHR,DSN=&PRFX..CONFIG(FLDMAP)
//TEMPLATE DD DISP=SHR,DSN=&PRFX..TEMPLATE
//PDFOUT   DD DSN=&PRFX..PDFOUT, ❺
//            DISP=(NEW,CATLG),
//            SPACE=(CYL,(10,10),RLSE),
//            DCB=(RECFM=U,BLKSIZE=27998)
//MAINARGS DD * ❻
--common-data     //DD:COMDATA
--common-map      //DD:COMMAP
--field-data      //DD:FLDDATA
--field-map       //DD:FLDMMAP
--pdf-template    //DD:TEMPLATE
--pdf-output      //DD:PDFOUT
//*
//CEEDUMP DD SYSOUT=*
//STDENV DD *
# Configures environment variables for the Java JVM.
# Variables must be exported to be seen by the application.
. /etc/profile
COZITEXT_HOME=/u/vendor/cozitext ❼
cd $COZITEXT_HOME
export JAVA_HOME=/usr/lpp/java/J6.0 ❽
### Set and export PATH, LIBPATH, CLASSPATH, and JVM options ❾
export PATH=/bin:"${JAVA_HOME}"/bin
LIBPATH=/lib:/usr/lib:"${JAVA_HOME}"/bin
LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390
LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390/j9vm
LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic
export LIBPATH="$LIBPATH":

APP_HOME=$COZITEXT_HOME
CLASSPATH=$APP_HOME:"${JAVA_HOME}"/lib:"${JAVA_HOME}"/lib/ext
# Add Application required jars to end of CLASSPATH
for i in "${APP_HOME}"/*.jar; do
    CLASSPATH="$CLASSPATH":"$i"
    done
export CLASSPATH="$CLASSPATH":

# Configure JVM options
IJO="-Xms16m -Xmx32m"
IJO="$IJO -Dfile.encoding=ISO8859-1"
export IBM_JAVA_OPTIONS="$IJO "
//
```

❷ This job step executes the IBM JZOS batch launcher, which is part of the z/OS Java SDK.

❶ The `DELOLD` step deletes the output data set so that a new one can be re-allocated on completion of the program.

❸ Identifies the Java program's main class.

❹ The `FLDDATA`, `COMDATA`, `COMMAP`, `FLDMAP`, and `TEMPLATE` DD statements identify all five inputs to the program. These data sets are expected to be pre-allocated and initialized prior to running this JCL.

❺ The `PDFOUT` DD statement allocates a data set for the output of the PDF generator program.

❻ The `MAINARGS` DD statement supplies the Java program arguments, here identifying the input and output DD statements.

❼ The `COZITEXT_HOME` environment variable is set (and exported) to the path of the program's installation directory. Once the variable is set, the cd command is executed in order to change to the program's directory.

❽ The `JAVA_HOME` environment variable is set (and exported) to the z/OS Java home directory. The Java version must match the JVMPRCxx proc.

❾ The remaining lines set and export environmental variables for the path, libpath, classpath and JVM options. Depending on the size of the field data to be processed, the JVM memory parameters and job REGION size may need to be adjusted.

When this program runs, STDOUT looks similar to the following:

```
JVMJZBL1001N JZOS batch Launcher Version: 2.3.0 2011-08-23
JVMJZBL1002N Copyright (C) IBM Corp. 2005. All rights reserved.
java version "1.6.0"
Java(TM) SE Runtime Environment (build pmz3160sr10-20111208_01 (SR10))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 z/OS s390-31 jvmmz3160sr10-20111207_96808 (JIT enabled, AOT enabled)
J9VM - 20111207_096808
JIT  - r9_20111107_21307ifx1
GC   - 20110519_AA)
JVMJZBL1023N Invoking com.dovetail.coz.itext.ItextPdfGenerator.main()...
JVMJZBL1024N com.dovetail.coz.itext.ItextPdfGenerator.main() completed.
JVMJZBL1021N JZOS batch launcher completed, return code=0
```

STDERR looks similar to the following:

```
2012-04-13 12:56:16,097 INFO  Configuration -
  --common-data //DD:COMDATA (csv delimted by '|')
  --common-map //DD:COMMAP (csv delimted by '|')
  --field-data  //DD:FLDDATA (csv delimted by '|')
  --field-map  //DD:FLDMAP (csv delimted by '|')
  --field-data-limit generate pages for all data rows
  --pdf-template //DD:TEMPLATE
  --pdf-output //DD:PDFOUT
2012-04-13 12:56:16,150 INFO  CommonFieldMap - Common form field names parsed: 2. Columns to be skipped: 0.
2012-04-13 12:56:16,150 INFO  CommonFieldMap - Common file expected to have 2 columns.
2012-04-13 12:56:16,228 INFO  DataFieldMap - Data form field names parsed: 5. Columns to be skipped: 3.
2012-04-13 12:56:16,229 INFO  DataFieldMap - Data file expected to have 8 columns.
2012-04-13 12:56:18,962 INFO  ItextPdfGenerator - Form fields with data or common field mappings: 5
2012-04-13 12:56:23,309 INFO  ItextPdfGenerator - Generation completed. Data rows processed: 15. Data rows
skipped due to error: 1. Execution time: 4347 ms.
```

The PDF output data set is allocated and populated with the generated PDF document.

Converting to z/OS hybrid batch

It is straightforward to change this job to run on a UNIX system using the Co:Z Toolkit. No changes to the source code are necessary; however, the program does need to be deployed on the target UNIX system. [1] The program data remain in their existing locations on z/OS, and the job's output stays on z/OS as well. By using the Co:Z Dataset Pipes **fromdsn** and **todsn** utilities, the data is pipelined throughout the program execution, optimizing performance and avoiding any "data at rest" on the UNIX system.

[1]Alternatively, the Co:Z **fromfile** command could be used to copy the program JARs from z/OS to a temporary server directory at the beginning of the job.

To accomplish this level of transparency for the end user and operations staff, the JCL itself does need to change: Instead of running Java on z/OS in a JZOS batch step, the Co:Z Launcher is used to run the program on a target blade server. The modified JCL is shown below with the new/modified statements highlighted.

```
//ITEXTCOZ JOB (),MSGCLASS=H,NOTIFY=&SYSUID
//PROCLIB JCLLIB ORDER=HLQ.COZ.JCL
//*
// SET PRFX=COZUSER.ITEXT.SAMPLE
//*
//DELOLD EXEC PGM=IEFBR14
//PDFOUT   DD DSN=&PRFX..PDFOUT,
//            DISP=(MOD,DELETE,DELETE),
//            SPACE=(CYL,1)
//*
//RUNCOZK EXEC PROC=COZPROC,
//   ARGS='-LI user@zbxlinux', ❶
//   REGSIZE='32M',LIBRARY='VENDOR.COZ.LOADLIB'
//FLDDATA  DD DISP=SHR,DSN=&PRFX..FLDDATA ❷
//COMDATA  DD DISP=SHR,DSN=&PRFX..CONFIG(COMDATA)
//COMMAP   DD DISP=SHR,DSN=&PRFX..CONFIG(COMMAP)
//FLDMAP   DD DISP=SHR,DSN=&PRFX..CONFIG(FLDMAP)
//TEMPLATE DD DISP=SHR,DSN=&PRFX..TEMPLATE
//PDFOUT   DD DSN=&PRFX..PDFOUT, ❸
//            DISP=(NEW,CATLG),
//            SPACE=(CYL,(10,10),RLSE),
//            DCB=(RECFM=U,BLKSIZE=27998)
//COZCFG DD * ❹
saf-cert=SSH-RING:RSA-CERT
ssh-tunnel=false
ssh-host=myzos
/*
//STDIN DD *
cd $HOME/Apps/coz-itext ❺
export CLASSPATH="."
for i in ./*.jar; do ❻
    CLASSPATH="$CLASSPATH":"$i"
    done
exec java \
com.dovetail.coz.itext.ItextPdfGenerator \
--common-data     <(fromdsn DD:COMDATA) \ ❼
--common-map      <(fromdsn DD:COMMAP) \
--field-data      <(fromdsn DD:FLDDATA) \
--field-map       <(fromdsn DD:FLDMAP) \
--pdf-template    <(fromdsn -b DD:TEMPLATE) \ ❽
--pdf-output      >(todsn   -b DD:PDFOUT)
/*
//
```

Rather than running the JZOS batch laucher, the Co:Z Toolkit Launcher is run, targeting a Linux system running on a zBX blade. ❶ Inputs are configured by the following DDs: FLDDATA, COMDATA, COMMAP, FLDMAP and TEMPLATE. ❷ In preparation for the output from the remote execution of the Java program, the `PDFOUT` DD is defined. ❸

The Co:Z Launcher configuration begins at ❹. The connection to the remote system (made via the ssh) is authenticated by a SAF digital certificate using the `saf-cert` property. The `ssh-tunnel=false` property allows us to bypass ssh for data transfer and is used for performance purposes. Because Co:Z is running on a zEnterprise system with attached zBX blades, the data transfer will take place over the secure Intraensemble Data Network (IEDN).

The next several lines are used to prepare for running the Java program. Change to the directory containing the Java program executable. ❺ Export the CLASSPATH variable set with all jars in the program directory. ❻

Having performed the necessary setup, the program can be invoked. *Process substitution* is used at ❼ to specify all input and output arguments. Process substitution creates an anonymous pipe at `/dev/fd/nn` where it writes the results of its commands (in this case **fromdsn** or **todsn**). The greater than (>) or less than (<) symbols in the command represent the direction of the pipe, and are analogous to the well known shell standard I/O redirection symbols. Because the PDF template and generated output are binary files, the **-b** switch on the **fromdsn** and **todsn** commands force the data to be transferred in binary. ❽

When the job completes, the program output is the same as it was for the z/OS version of the job, and is routed to the PDFOUT DD. The program log looks much like the the z/OS version, but does contain output that clearly indicates that this was no ordinary z/OS execution:

STDOUT looks similar to the following:

```
CoZLauncher[N]: version: 2.1.1 2012-03-16
CoZLauncher[N]: Copyright (C) Dovetailed Technologies, LLC. 2006. All rights reserved.
CoZLauncher[I]: Agent output WTO is OFF
CoZLauncher[I]: ssh tunnelling is ON
CoZLauncher[I]: CoZServer listener socket bound to: 127.0.0.1:8040
cozagent[N]: version: 1.1.0 2012-03-16
fromdsn(DD:STDIN)[N]: 21 records/1680 bytes read; 468 bytes written in 0.013 seconds (35.156 KBytes/sec).
fromdsn(DD:COMMAP)[N]: 1 records/26 bytes read; 27 bytes written in 0.002 seconds (13.184 KBytes/sec).
fromdsn(DD:TEMPLATE)[N]: 9 records/54153 bytes read; 54153 bytes written in 0.095 seconds (556.671 KBytes/sec).
fromdsn(DD:COMDATA)[N]: 1 records/38 bytes read; 39 bytes written in 0.006 seconds (6.348 KBytes/sec).
fromdsn(DD:FLDMAP)[N]: 1 records/94 bytes read; 95 bytes written in 0 milliseconds.
fromdsn(DD:FLDDATA)[N]: 16 records/1727 bytes read; 1743 bytes written in 0 milliseconds.
todsn(DD:PDFOUT)[N]: 110163 bytes read; 4 records/110163 bytes written in 1.306 seconds (82.374 KBytes/sec).
todsn(DD:STDERR)[N]: 1017 bytes read; 15 records/1003 bytes written in 3.287 seconds (309.401 Bytes/sec).
todsn(DD:STDOUT)[N]: 0 bytes read; 0 records/0 bytes written in 3.289 seconds (0.000 Bytes/sec).
CoZLauncher[N]: lisa@108.89.244.83 target command '<default shell>' ended with RC=0
CoZLauncher[I]: CoZLauncher ended with RC=0
```

STDERR looks similar to the following:

```
2012-04-23 14:41:45,371 INFO  Configuration -
  --common-data /dev/fd/63 (csv delimted by '|')
  --common-map /dev/fd/62 (csv delimted by '|')
  --field-data  /dev/fd/61 (csv delimted by '|')
  --field-map  /dev/fd/60 (csv delimted by '|')
  --field-data-limit generate pages for all data rows
  --pdf-template /dev/fd/59
  --pdf-output /dev/fd/58

2012-04-23 14:41:46,219 INFO  CommonFieldMap - Common form field names parsed: 2. Columns to be skipped: 0.
2012-04-23 14:41:46,220 INFO  CommonFieldMap - Common file expected to have 2 columns.
2012-04-23 14:41:46,225 INFO  DataFieldMap - Data form field names parsed: 5. Columns to be skipped: 3.
2012-04-23 14:41:46,225 INFO  DataFieldMap - Data file expected to have 8 columns.
2012-04-23 14:41:46,316 INFO  ItextPdfGenerator - Form fields with data or common field mappings: 5
2012-04-23 14:41:46,731 INFO  ItextPdfGenerator - Generation completed. Data rows processed: 15. Data rows
skipped due to error: 1. Execution time: 415 ms.
```

This example shows how an existing Java program can be easily moved into the z/OS hybrid batch computing environment with a high level of transparency and low operational disruption. The Co:Z Toolkit facilities coupled with several very useful shell features make this possible.

Conclusion

This case study illustrates how Co:Z can be used to run a Java program for generating a multi-page PDF document in a z/OS hybrid batch environment. The z/OS hybrid batch model allows enterprises to:

- Optimize the use of heterogeneous computing resources from a z/OS batch environment.
- Extend the reach of z/OS to other platforms and application processes using existing z/OS data sets.
- Preserve investments in existing applications and processes. Changing programs to use new APIs is not required.

The source code and JCL for these case studies can be downloaded directly from our website: *http://dovetail.com*.

Appendix A. Legal Notices