Co:Z Co-Processing Toolkit

z/OS Hybrid Batch Processing: Running SAS Programs with Co:Z

V 1.0 Edition

Published March, 2012

Introduction

Note

Before using this information and the products that it refers to, read the information in *Appendix A, Legal Notices*

SAS programs have been run on the mainframe for over fifty years. They are used for data analysis certainly, but are also widely used to produce reports; many enterprises use SAS programs to process System Management Facility (SMF) data and produce usage and audit reports.

While the use of SAS on z/OS is still prevalent, there has been a significant shift over time to move this processing off platform; licensing cost consideration and overall workload reduction goals often factor into these decisions. MXG Software, offered by Merrill Consultants is a popular software package that is used by many organizations to process their SMF data. While Merrill offers licenses for z/OS, the software is notably popular for running SAS programs on UNIX and Windows systems to process offloaded SMF data.

IBM's recent roll out of zEnterprise hardware and associated BladeCenter extensions (zBX) make the practice of running SAS programs on distributed platforms even more enticing. Our Co:Z Co-Processing Toolkit provides z/OS enterprise customers the opportunity to exploit this new hardware in a powerful way, and makes it easy to tightly integrate distributed processing with 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 not need to be modified.
- The target program's input and output are redirected from/to z/OS spool files or datasets
- The target program may access other z/OS resources: DD's, datasets, UNIX files and programs
- The target program's exit code is adopted as the z/OS job step condition code.

This article explores these ideas by examining the technical considerations and showing through two case studies how SAS processing is an ideal candidate for the z/OS hybrid batch operating environment.

z/OS Hybrid Batch: SAS Considerations

Before looking at the first case study, it's worth taking some time to consider some of the technical issues involved in moving SAS programs into the hybrid batch environment. While there is a great deal of compatibility between SAS implementations on z/OS and other platforms, there are some important differences. Specifically, the following items require special attention:

Program source

On z/OS, the SAS PROC expects its source to be available on the SYSIN DD. On other platforms, the SAS command line executable is invoked with a filename argument that refers to the SAS program to execute.

%INCLUDE statements

Used to include additional source into a SAS program. On z/OS, these statements take the form of `%INCLUDE SOMELOC(MEMBER)`. On non-z/OS platforms, this syntax will attempt to include the external file `member.sas` from the path found in the `SOMELOC` environment variable.

Input data source

On z/OS, the SAS program will usually specify an `INFILE` statement that refers to a DD name where the input data is located. On other platforms, the INFILE statement can be used to read from an external file, an unamed pipe, or a named pipe (FIFO).

Input data formats

On z/OS, input data may be plain text (i.e. alphanumeric), or a mixture of text and z/OS binary data (e.g. packed/zoned decimal, binary). Reading and processing text data on other platforms is straightforward, but special SAS `informats` need to be used to process z/OS binary data.

Output

On z/OS, SAS program log output is generally sent to the SASLOG DD and program (print) output is sent to the SASLIST DD. On other platforms this output is sent by default to `program_name.log` and `program_name.lst` respectively.

It is a goal of any conversion is to minimize the changes required to existing source programs and to keep operational mechanisms intact. But it should also be easy! In the case studies that follow, we will employ some innovative programming techniques and specific Co:Z Toolkit features to meet these goals. These include:

Dataset Pipes Utilities

The Co:Z Toolkit Dataset Pipes **fromdsn** and **todsn** 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 SAS input and output data.

The **toasa** command is a filter utility that converts documents with ASCII form-feed characters into ASA carriage control documents

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 Dataset Pipes commands to simplify the target program's access to z/OS file and dataset resources.

Named pipes

Where process substitution is not possible, temporary named pipes (FIFOs) can be used.

Case Study: Analyzing Population Data

The National Center for Health Statistics (NCHS) publishes demographic data on birth and death rates broken down by gender. The public raw data for this case study covers the period from 1979-1988, and contain records at the national, state and county level. Each record is 140 characters long and contains alphanumeric data arranged in fixed columns.

The following SAS program reads the raw population data records and reports (via `PROC PRINT`) on the number of annual births by race/gender at the national level, discarding all state and county level records.

```
PROC FORMAT;
  VALUE RG 1 = "W/M"  2 = "W/F"  3 = "B/M" 4 = "B/F" 5 = "O/M" 6 = "O/F";

DATA POP7988;
  INFILE POPDATA MISSOVER LRECL=140;  ❶

  INPUT
    YEAR      6-9  ❷
    RACEGEND  10
    BIRTHS    8.
    CNTY    $ 115-139
    RECTYPE   140
    ;

  FORMAT RACEGEND RG.;

  if RECTYPE ne 1 then delete;
  drop RECTYPE;

PROC PRINT NOOBS SPLIT='*' DATA=POP7988;
  TITLE1 'NUMBER OF US BIRTHS ANNUALLY BY RACE/GENDER';
  TITLE2 'Source: National Center for Health Statistics (1979-1988)';
  LABEL
    YEAR     = 'Year*===='
    RACEGEND = 'Race/Gender*==========='
    BIRTHS   = '# of Births*==========='
    CNTY     = 'Location*========'
    ;

RUN;
```

❶ The `INFILE` statement specifies that the file reference `POPDATA` contains the raw population data.

❷ The `INPUT` statements illustrate that the raw data is column based. All of the columns are interpreted as numeric except for `CNTY`, where the `$` indicates that the field is character data.

The program is run with SAS on z/OS with the following JCL:

```
//POPZOS JOB ZOS,'Population Report &USERID',NOTIFY=&USERID
//*
//SASPOP   EXEC PROC=SAS
```

```
//SYSIN     DD DISP=SHR,DSN=HLQ.SAMPLE.SAS(POPSAS) ❶
//POPDATA   DD DISP=SHR,DSN=HLQ.POP7988 ❷
//
```

❶ The SYSIN DD statement points to the SAS program source, shown above.
❷ The POPDATA DD statement points to the raw population data. Note that this name matches the INFILE file reference in the SAS program.

When this program runs, the SASLIST DD (defined by the SAS PROC) contains the program output which looks like this:

```
NUMBER OF US BIRTHS ANNUALLY BY RACE/GENDER                          1
Source: National Center for Health Statistics (1979-1988)
                                        19:04 Friday, February 24, 2012
     Year    Race/Gender    # of Births    Location
     ====    ===========    ===========    ========

     1979        W/M          1460512        U.S.
     1979        W/F          1382355        U.S.
     1979        B/M           282869        U.S.
     1979        B/F           274815        U.S.
     1979        O/M            47886        U.S.
     1979        O/F            45961        U.S.
     1980        W/M          1509357        U.S.
     1980        W/F          1426994        U.S.
     1980        B/M           287996        U.S.
     1980        B/F           280084        U.S.
     1980        O/M            55263        U.S.
     1980        O/F            52564        U.S.
     ...
```

Details of the program's execution are sent to the SASLOG DD, also defined by the SAS PROC.

Converting to z/OS hybrid batch

It is simple to convert this program to run on a UNIX system using the Co:Z Toolkit. No changes to the source code are necessary, the program data and source stay in their existing locations on z/OS, and job step output (SAS log and list) stay 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 minimizing any "data at rest" on the UNIX system.

To accomplish this level of transparency for the end user and operations staff, the JCL itself does need to change: [1] Instead of invoking SAS directly, the Co:Z Launcher is used and the SASLOG and SASLIST DDs are explicitly defined in the job step. The modified JCL is shown below with the new/modified statements highlighted.

Note

Although the SAS system from the SAS Institute is available on many distributed platforms, this

[1] In fact, it is possible to substitute a compatible Co:Z Launcher PROC in place of the SAS PROC.

example uses the World Programming System (WPS) UNIX platform to run the program. For more information, see: *http://teamwpc.co.uk/products/wps*

```
//POPCOZ JOB ZOS,'Population Report &USERID',NOTIFY=&USERID
//PROCLIB JCLLIB ORDER='HLQ.COZ.JCL'
//*
//RUNCOZK EXEC PROC=COZPROC,ARGS='user@zbxlinux', ❶
//   REGSIZE='32M',LIBRARY='VENDOR.COZ.LOADLIB'
//SYSIN   DD DISP=SHR,DSN=HLQ.SAMPLE.SAS(POPSAS)
//POPDATA DD DISP=SHR,DSN=HLQ.POP7988
//SASLIST DD SYSOUT=*,DCB=(BLKSIZE=264,LRECL=260,RECFM=VBA) ❷
//SASLOG  DD SYSOUT=*,DCB=(BLKSIZE=141,LRECL=137,RECFM=VB)
//COZCFG DD *
saf-cert=SSH-RING:RSA-CERT ❸
ssh-tunnel=false
ssh-host=myzos
/*
//STDIN DD *
PGM=/usr/local/wps-3.0.0.2/bin/wps ❹
#PGM=/usr/local/sas91/sas

sysin=/tmp/coz.$$.$random.sas ❺
mkfifo $sysin
trap 'rm -rf -f $sysin' EXIT

fromdsn DD:SYSIN > $sysin & ❻

POPDATA=<(fromdsn DD:POPDATA)              \ ❼
$PGM -print >(toasa | todsn -z DD:SASLIST) \ ❽
     -log   >(todsn -z DD:SASLOG)          \
     -logparm "write=immediate"            \
     $sysin ❾
//*
```

Rather than running the SAS PROC directly, the Co:Z Toolkit Launcher is run, targeting a Linux system running on a zBX blade. ❶ In preparation for the output from the remote execution of the SAS program, the `SASLIST` and `SASLOG` DDs are defined, directed to SYSOUT. ❷

The Co:Z Launcher configuration begins at ❸. 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).

`STDIN` is an inline DD that contains the shell script that will run on the zBX Linux virtual server. The first line of the shell script ❹ creates a variable to point the World Programming System executable. Beginning at ❺, a temporary named pipe is created. This is necessary because **wps** requires that the program source filename include a `.sas` suffix. The **trap** command ensures that it will be automatically removed on script exit.

At ❻, **fromdsn** is run in a separate process to "reach back" into the z/OS job step to read the SAS program source and redirect to the named pipe so that it can be made available to **wps**. This is a simple invocation of fromdsn; the default behavior is to treat the data as text, and to translate from the codepage on z/OS (EBCDIC) to the local platform (ASCII).

Having performed the necessary setup, the program can be invoked. *process substitution* is used at ❼ to read the population data. Process substitution creates an anonymous pipe at `/dev/fd/nn` where it writes the results of its commands (in this case **fromdsn DD:POPDATA**). This fd pathname is in turn assigned to the environment variable `POPDATA` which is the name of the fileref in the SAS program's INFILE statement.

The wps program is run with several options. To route the program print and log output to be sent back to z/OS rather than be written to local files, process substitution is again used for the `-print` and `-log` options. ❽ In this case, the **todsn** command is used to write the SASLIST and SASLOG DDs. The `-z` option is used to supress any todsn error messages if nothing is written. Again, the default invocation of the todsn command treats the data as text, and the desired codepage translation from ASCII to EBCDIC takes place.

Note that on non-z/OS platforms, the print output will contain ASCII form-feed characters rather than the ASA carriage control characters expected on z/OS. The Dataset Pipes **toasa** filter utility converts this stream to ASA format so that it is suitable for the z/OS SASLIST `RECFM=FBA` DCB.

The `-logparm` option instructs wps to write anything written to the log immediately, without buffering. This enables the program log output to be made available on z/OS while the program executes. This makes it possible to monitor the job's progress by examining the z/OS spool file. The `$sysin` on the final line of the script is the reference to the SAS program that was set up earlier in the script.

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 SASLIST 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:

```
NOTE: The file POPDATA is:
File Name=/dev/fd/61, ❶
Access Permissions=prw-------,Number of Links=1,
Owner Name=goetze,Group Name=goetze,
File Size=0,Last Modified=Mar 01 2012,
Created=Mar 01 2012,Lrecl=140, Recfm=V
```

❶ Evidence of the process substitution used to read the population data.

This example shows how an existing SAS 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.

Case Study: SMF 119 Reporting

The Population analysis case study is a fairly simple SAS program. In particular, the raw data are arranged in fixed alphanumeric columns, making it easy to transfer from the EBCDIC z/OS platform to the ASCII Linux platform using the standard codepage conversion built into the Dataset Pipes **fromdsn** and **todsn** utilities. Furthermore, the SAS source code is contained in a single dataset, requiring no special handling for including additional code segments.

What about a more complex problem; one that involves multiple source files and variable length data records containing z/OS binary field data? An ideal example is a SAS program that analyzes SMF data and produces audit reports based on certain record type/subtypes.

Neil Duffee at the University of Ottawa has generously made available a sample program that does this specifically. The JCL below runs a SAS program on z/OS that analyzes SMF 119 subtype 3 (FTP / Co:Z SFTP client completion records) and reports on user and transferred files frequencies. This SAS program is constructed via a combination of dataset concatenations for the program logic along with embedded SAS `%INCLUDE` statements for the various SMF record layouts. Here is the original JCL:

```
//SMFZOS JOB ZOS,'FTP count &USERID',NOTIFY=&USERID
//*  -----------------------------------------------------
//SASDATA  SET SASDATA=HLQ.SMF.DUMP
//SASSRCE  SET SASSRCE=HLQ.SAMPLE.SAS
//*  -----------------------------------------------------
//STATS   EXEC PROC=SAS
//SYSIN     DD DISP=SHR,DSN=&SASSRCE(FTPCNT00)  -extract data  ❶
//          DD DISP=SHR,DSN=&SASSRCE(FTPCNT02)  -only select IP Addr
//          DD DISP=SHR,DSN=&SASSRCE(FTPCNT05)  -extract cont'd
//          DD DISP=SHR,DSN=&SASSRCE(FTPCNT10)  -tabulate job names
//          DD DISP=SHR,DSN=&SASSRCE(FTPCNT08)  -sort by userId
//          DD DISP=SHR,DSN=&SASSRCE(FTPCNT09)  -print results
//SASINC    DD DISP=SHR,DSN=&SASSRCE ❷
//WORK      DD SPACE=,DATACLAS=XLARGE  -use SMS sizing
//SMFDATA   DD DISP=SHR,UNIT=(,,DEFER),VOL=(,RETAIN),
//             DSN=&SASDATA
```

❶ Program logic is constructed here via dataset concatenation

❷ SMF record layouts are included in the program logic via `%INCLUDE SASINC(SMFxxxx)` statements.

Because the SMF dump data contains binary fields, we will want to process it in its original (binary) format. To do this properly on the Linux platform, we will need to change some of the program source, but these changes are straightforward [1]. We will:

1. Move the SMF record layouts to the remote Linux system and change the INPUT statement `informats` to be

[1]Another approach might be to use the SAS user macro language to automatically rewrite the source code, which would eliminate the need to make manual modifications.

compatible with the z/OS data and modify the record layout filenames so that the existing `%INCLUDE` statements will work on Linux [2].

2. Modify the `INFILE` statement associated with the SMF data to mark the data as z/OS variable block spanned (VBS)

Modifying the SAS SMF Record Layouts

In a SAS program, INPUT informats are platform specific. For example, on z/OS, the character data associated with $CHAR field is EBCDIC. On Linux, the data associated with the $CHAR informat is ASCII. Because the z/OS SMF data will be read in binary on Linux, we will want to convert the data from its native format to the platform format. Fortunately, the SAS language includes informats that do exactly this.

The first step is to download all of the members from the SASINC PDS into a SAS include directory on the remote Linux system. This can be done easily via Co:Z sftp. Don't try this with IBM's Ported Tools sftp; it doesn't handle MVS datasets!

```
user@Linux:~$ cd /my/sas/smf/layouts
user@Linux:/my/sas/smf/layouts$ sftp user@zos
Connecting to zos...
sftp> cd //HLQ.SMF.REPORTS.LAYOUTS.SAS
sftp> get *
sftp> quit
```

Now that the SMF layouts are present on the remote Linux system, they can be modified to use the proper informats. We need to make the following changes:

Positive binary integers
Change `PIBw.d` to `S370FPIBw.d`

Fixed length strings
Change `$CHARw.` to `$EBCDICw.`

Variable Length strings
There is no direct EBCDIC analog to the `$VARYINGw.` format, whose length is determined at runtime. In order to properly interpret this z/OS data on Linux, the character data should be read in with the $VARYING informat, then converted from EBCDIC using the INPUTC function. For example, the original:

```
INPUT smf119FT_FCFileName $VARYING. smf119S2Len;
```

would become:

```
INPUT smf119FT_FCFileName $VARYING. smf119S2Len;
smf119FT_FCFileName=INPUTC(smf119FT_FCFileName,'$EBCDIC.',smf119S2Len);
```

[2] Alternatively, we could leave the include source on z/OS and modify the `%INCLUDE` statements to use a pipe file and the **fromdsn** command.

Source Inclusion via %INCLUDE

In the original JCL above, note the declaration of the `SASINC` DD. This DD is used to satisfy requests in the main SAS program to include additional source code via the SAS `%INCLUDE` statement. For example, consider the following snippet of code from the main source file `FTPCNT00`:

```
%INCLUDE SASINC(SM119HDR);
if  smf119stp ne 03 then delete; * ftp client completion sub-type;
```

The SMF119 header is checked to see if the requested subtype (3) is present. On z/OS the header layout is found in the `SM119HDR` member of the partitioned dataset referred to by the `SASINC` DD. However, these files now reside on the remote system in the `/my/sas/smf/layouts` directory. Will the program logic need to be changed to read these files? Fortunately, no - the program source will work as written if the following is true on the remote Linux system:

- An environment variable named `SASINC` is set to point to `/my/sas/smf/layouts`
- The include targets themselves (e.g. SMF119HDR) need to exist in this directory, but they must have lowercase names with a `.sas` suffix (e.g. `smf119hdr.sas`). This renaming process can be done with a one line shell script:

```
for f in *; do n=$(echo $f | tr 'A-Z' 'a-z'); mv $f $n.sas; done
```

Reading binary data

One final issue needs to be addressed: The reading and processing of the raw SMF data in binary on the remote Linux system. The INFILE statement used to read the SMF data:

```
infile smfdata
```

must be changed to:

```
infile smfdata recfm=s370vbs lrecl=32760
```

This tells the SAS program that the incoming data is a raw z/OS RECFM=VBS file containing block and record descriptor words (BDWs and RDWs). To deliver the data to the remote system in that format, we will once again rely on **fromdsn**, this time with some additional options, as shown in the modified JCL below:

The modified JCL

The last step in the process is to rewrite the original JCL to run the Co:Z Launcher rather than SAS directly:

```
//SMFCOZ JOB ZOS,'FTP count &USERID',NOTIFY=&USERID
//PROCLIB JCLLIB ORDER='HLQ.COZ.JCL'
//*  ------------------------------------------------------
//SASDATA  SET SASDATA=HLQ.SMF.DUMP
//SASSRCE  SET SASSRCE=HLQ.SAMPLE.SAS
```

```
//*  -------------------------------------------------------
//RUNCOZ EXEC PROC=COZPROC,ARGS='user@zbxLinux',
//   REGSIZE='32M',LIBRARY='VENDOR.COZ.LOADLIB'
//SYSIN   DD DISP=SHR,DSN=&SASSRCE(FTPCNT00)  -extract data
//        DD DISP=SHR,DSN=&SASSRCE(FTPCNT05)  -extract cont'd
//        DD DISP=SHR,DSN=&SASSRCE(FTPCNT10)  -tabulate job names
//        DD DISP=SHR,DSN=&SASSRCE(FTPCNT08)  -sort by userId
//        DD DISP=SHR,DSN=&SASSRCE(FTPCNT09)  -print results
//*SASINC    DD DISP=SHR,DSN=&SASSRCE ❶
//*WORK      DD SPACE=,DATACLAS=XLARGE  -use SMS sizing ❷
//SMFDATA DD DISP=SHR,UNIT=(,,DEFER),VOL=(,RETAIN),
//             DSN=&SASDATA
//SASLIST DD SYSOUT=*,DCB=(BLKSIZE=264,LRECL=260,RECFM=VBA)
//SASLOG  DD SYSOUT=*,DCB=(BLKSIZE=141,LRECL=137,RECFM=VB)
//COZCFG DD *
saf-cert=SSH-RING:RSA-CERT
ssh-tunnel=false
ssh-host=myzos
//STDIN DD *
PGM=/usr/local/wps-3.0.0.2/bin/wps
#PGM=/usr/local/sas91/sas


sysin=/tmp/coz.$$.$random.sas
mkfifo $sysin
trap 'rm -rf -f $sysin' EXIT

fromdsn DD:SYSIN > $sysin &

SMFDATA=<(fromdsn -b -o recfm=u DD:SMFDATA) \ ❸
SASINC=/my/sas/smf/layouts                  \ ❹
$PGM -print >(toasa | todsn -z DD:SASLIST)  \
     -log   >(todsn -z DD:SASLOG)           \
     -logparm "write=immediate"             \
     $sysin
/*
```

Once again, the new/modified statements are highlighted. This JCL is similar to the JCL used in the population case study, but there are some noteworthy differences:

Because the include files are now located on the remote Linux system, the z/OS include setup at ❶ is no longer used. In addition, the z/OS `WORK` library ❷ is not needed. The temporary SAS libraries will be created on the remote Linux system instead and removed automatically.

Of particular interest is the **fromdsn** command used to read the SMF data ❸. The `-b` option is specified to read the data in binary, and the `-o recfm=u` option is used to preserve the BDWs and RDWs as required by the SAS program. Also note the `SASINC` environment variable is set to the location of the modified SMF record layouts.❹

Conclusion

The use of SAS programs to analyze data and report is a common practice that reaches across all industries.

These case studies illustrate how Co:Z can be used to run SAS programs in the z/OS hybrid batch environment. If the existing SAS program is straightforward, as many are, the changes required to make this work are minimal. Even complex cases can be managed without a major effort, and the potential benefits are significant:

- Reduced z/OS workload
- Lower software licensing costs

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

Appendix A. Legal Notices

The phrase *SAS program* in this article is used to describe programs written in the SAS programming language. It does not refer to software sold by the SAS Institute.