

The Other Five Percent

Writing SAS Code for Multiple Operating Systems

Ryan K. Carr and Daniel R. Bretheim
William M. Mercer, Inc.

I. INTRODUCTION

The vast majority of code in the SAS® system is identical across platforms. This has been of great value to us in programming under changing environments from MS DOS, Windows and OS/2, to CMS and MVS. With computing trends moving towards downsizing, rightsizing, and client-server models SAS programmers need to be concerned with not only changes in operating systems, but also with supporting multiple systems at once.

William M. Mercer, Inc. is currently considering downsizing its National Healthcare Analysis Unit's programs from an MVS mainframe to some unspecified windows based platform. This raised some questions for us as we began developing a new system DES (Dependent Enrollment Simulation) which creates dependent records in insurance enrollment files. Should we write the system for MVS, which is our current platform, or Windows one of the tentative new platforms. Given that it was already decided to do the system in SAS (since that was the language the earlier version was written in) and knowing how easy it had been to transition from one system to another as SAS programmers, we chose to write code which would run on either system. This code would also be easily adaptable to any other system in the event that a direction other than Windows was chosen for the downsizing.

II. THE BEGINNING - SAS AUTOMATIC MACRO VARIABLE &SYSSCP.

In order to implement this strategy it was essential that, unless the code for both operating systems was 100% compatible, the program needed to be able to tell which system it was running on and then branch to the appropriate code. This problem was easily solved using SAS's automatic macro variable &SYSSCP. which returns to SAS a name representing the current operating system (i.e. OS for MVS, WIN for Windows, OS2 for OS/2...). Here is a simple example of how to test &SYSSCP. on your operating system and the value for MVS:

```
%PUT SAS is fun on &SYSSCP. ;
SAS is fun on OS
```

Given that this major hurdle was already handled by SAS, the next step was to identify areas where the code differs across operating systems along with any other operating system dependent non-code related features. Finally strategies to handle these problem areas needed to be developed for each system, and implemented.

III. LIBNAME AND FILENAME

The SAS statements which interact directly with the operating system are the most obvious candidates for change according to the operating system. The most common of these (and the only ones used in the DES system) are LIBNAME and FILENAME. The cross platform problems posed by both of these statements were simple: the naming structure of each operating system, and the options available on each operating system. Here's an example of comparable FILENAME statements on various operating systems:

```
MVS
filename enrrow "NH.D76.SPC.CNAG.E3.M9306.STD"
        DISP=OLD ;

CMS
filename enrrow "enr9306 data a"
        DISP=OLD ;

Windows or OS/2
filename enrrow "c:\work\des\cnag\enr9306.dat" ;
```

Without using &SYSSCP. you would have to maintain multiple sets of code, one for each operating system. Even simple changes would have created a nightmare in keeping track of which changes had been made on which set of code, and updating all other versions. A simpler solution was the use of &SYSSCP. to maintain the system dependent lines in only one set of code and allow SAS to choose which to run, depending on what operating system is being used. In the DES system we implemented all of the LIBNAME and FILENAME statements in code similar to this:

```
%macro syslib ;
    %if &syscp. = WIN %then %do ;
        filename desprog% 'C:\work\des' ;
        libname descnag "r:\work\des\cnag" ;
    %end ;
    %if &syscp. = OS %then %do ;
        filename desprog% 'nh.test.des.heg' ;
        LIBNAME DESCNAG TAPE
            "NH.D76.SPC.CNAG.E3.M9306.DES"
            UNIT=CART
    %end ;
%endmacro
```

```

DISP=(NEW,CATLG,DELETE) ;

tend ;
%mend syslib ;
%syslib ;
options maautosource sasautos=desprogs ;

```

Examining the above code, you can see that it was minimally more complex to write than the two sets of LIBNAME and FILENAME statements. The only extra code needed was the %MACRO and %MEND statements to create the macro, the %IF and %END statements to alternatively run the different code on different systems, and the %SYSLIB statement which invokes the macro. After that, the libref DESCNAG and the fileref DESPROGS could be used throughout the program (as in the options statement) regardless of operating system.

One trait of the OS LIBNAME above which will be addressed later is the problem introduced through the use of system dependent features such as tape processing on MVS.

IV. MOVING INFORMATION

Wanting the DES system to run on both Windows and MVS, we quickly encountered problems in moving information between the two systems. The files being moved were to be in many formats; SAS code, SAS datasets, and flat files for input data. While any of these file types could be moved with PROC UPLOAD / DOWNLOAD, this wasn't an option at my site, so we needed to come up with other solutions. Each file types presented different problems, some of which were easy to resolve, others were somewhat trickier, but none were too tough.

For SAS Code, we decided to set the standard that all changes would be made and tested first on the PC. With this constraint, the only hurdle which needed to be overcome was a simple upload of each program when it was created or changed. Most PC terminal emulation programs include an upload / download utility, we used the one in Access for Windows shown below.

In uploading programs the options selected were; translate to EBCDIC, translate CRLF, and the ASCII (437) translation table. The only non-standard option used (and one other programmers should watch out for) was the change in the translation table. We needed to try out a few options here since the standard ASCII to EBCDIC conversion didn't properly translate the '|' character which we use in the border of SAS comments, and as the concatenation operator ||. Other characters to watch out for are the tab character, and any line-drawing characters. Depending on what specialized characters you use, you may have to try other translation tables or even create one of your own for moving programs back and forth between systems.

Since we were using versions of SAS above 6.04 on both systems, transporting SAS datasets was also rather simple. Here is a sample of the code used to prepare some utility SAS datasets for the DES from Windows to MVS:

```

libname desc 'c:\work\des' ;
libname desx xport 'c:\work\des\utility.tpt' ;
proc copy in=desc out=desx ;
    select function csx cage cclaim ;
run ;

```

We were then able to move the datasets as one file by either copying the library to tape, or uploading it to disk through the same utility used for the programs. When using the upload utility, be sure NOT to use any translation options. The uploaded library could then be translated back into native SAS datasets using these statements:

```

libname desc 'nh.d76.des.utility.data' ;
libname desx xport 'nh.d76.des.utility.xport' ;
proc copy in=desx out=desc ;
    select function csx cage cclaim ;
run ;

```

If you are running older versions of SAS, another alias for the xport engine which can be used is sasv5xpt. The SAS system has other methods available for transporting data such as PROC CPORT and CIMPORT, but we find these simple LIBNAME and PROC COPY statements much easier.

The final type of file which needed to be moved was the input data. For the DES system, there were two distinct types of input data; enrollment and claims. While the enrollment data file was small and easy to handle, the claims data had packed decimal fields which made translation from EBCDIC to ASCII tricky and often ran into tens of megabytes. At these sizes, downloading the claims data through the Access for Windows utilities was no longer an option. Luckily, we had a cartridge reader attached to our network. The only preparation needed for the input data files was to copy them to non-compressed tapes. Using the following JCL inline proc:

```
//T0077NH JOB (HEACHP61H01Z),
//          'RYAN CARR DESDOWN',
//          CLASS=A,
//          TIME=1,
//          MSGCLASS=X,
//          NOTIFY=T0077NH
//*
/*JOBPARM P=PROC01,ROOM=CHP
/*
//COPYDSN PROC L=1
//COPY EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=(SHR),
//          DSN=&DSN
//SYSUT2 DD DSN=&DSN,
//          DISP=(NEW,KEEP),
//          UNIT=CART,
//          LABEL=(&L,SL),
//          VOL=(,RETAIN,,SER=&AXHN),
//          EXPDT=98000,
//          DCB=(TRTCH=NOCOMP)
//SYSIN DD DUMMY
//          PEND
/*
//S1 EXEC COPYDSN,
//          DSN='NH.D76.SPC.CNAG.C3.M9012.GENERIC',
//          AXHN=AXH071
/*
//S1 EXEC COPYDSN,
//          DSN='NH.D76.SPC.CAPH.E3.M9306.STD.MED',
//          AXHN=AXH075
/*
```

Once the tapes were created, the files needed to be moved from the EBCDIC tapes onto the PC network drives using the cartridge reader's utilities. All of the options such as record length and blocksize for the utility were set the same as they were in the JCL that created the tapes. Given the packed decimal (a data type inherent to MVS EBCDIC files) fields in the claims data, the only option which presented a

problem was to translate or not to translate... that was the question.

V. ASCII VS EBCDIC

For the enrollment data translation didn't matter since all of the fields were either character or zoned decimal. The claims data however contained the packed decimal data mentioned above. The problem here was that there were no direct translation for EBCDIC packed decimal to ASCII. We could have written a specific translation in Basic, FORTRAN, or C for each file type with packed decimal. Other than being cumbersome, this would also have created a new record layout under Windows, generating unnecessary differences in code between platforms.

However, since SAS provides EBCDIC formats on its ASCII platforms we could just read the files as is... without translation. Here is a sample program to read in EBCDIC files under Windows:

```
data claims ;
  infile genraw lrecl=349 ;
  drop dobc ;
  input @87 rel $ebcdic1. @ ;
  if rel in ('2','3') ;
  input @77 ESSN $ebcdic10.
        @87 COV $370fpd4.
        @96 DOBC $ebcdic8.
        @117 DSEX $370fzdl.
        ;
  if dobc not in ('00000000',' ') then do
    ;
    dob = input(dobc,yyymmdd8.) ;
    end ;
    if rel = '3' then dobkey = dob ;
  run ;
```

In the above program there are two points of interest. First is the use of the \$ebcdic. informat to read the character data, \$370fpd. to read packed decimal data, and \$370fzd. to read zoned decimal data. There are other formats which may be of interest listed in the SAS Language Reference Guide. The second point is the use of the \$ebcdic8. informat to read in date data. Since there is no informat for reading in EBCDIC encoded data directly into a SAS date variable, we simply read in the data first as character and then used a put() function to create a numeric SAS date variable.

Since EBCDIC is the native encoding system for MVS, these informats are not needed on the mainframe. Still wishing to have only one set of code, we implemented the following simple macro which also allows us to use ASCII data easily in the

future if such a change to the system becomes possible.

```

if &dtype. = EBCDIC and &syscp. = WIN then do
;
  %let c=%ebcdic ;
  %let n=s370fib ;
  %let z=s370fzd ;
  %let p=s370fpd ;
end ;
else do ;
  %let c= ;
  %let n= ;
  %let z= ;
  %let p= ;
end ;

data claims ;
  infile genraw lrecl=349 ;
  drop dobc ;
  input @87 rel &c.1. @ ;
  if rel in ('2','3') ;
  input      @77 EESSN    &c.10.
            @87 COV      &p.4.
            @96 DOBC     &c.8.
            @117 DSEX    &z.1.
            ;
  if dobc not in ('00000000','') then do
;
    dobc = input(dobc,yyymmdd8.) ;
  end ;
  if rel = '3' then dobkkey = dobc ;
run ;

```

With this code, we can choose whether to use the standard SAS informats, or the specialized EBCDIC informats. This choice is made depending on a combination of the macro variable &DTYPE. which is set as a parameter in the DES system and the automatic macro variable &SYSSCP.

One option we haven't been able to find, which may prove to be useful, is the ability to read an ASCII file on MVS in a similar manner. Given the volume of SAS documentation this doesn't mean that it isn't possible, only that we haven't stumbled across the correct page yet. Of course if it can't be done and you know it, let us know so that we can finally stop looking.

VI. SORT SEQUENCE

Another stumbling block which arises from the different encoding schemes is the sort sequence. The most obvious illustration of the differences is that in the ASCII sequence numbers are lower than characters, while in EBCDIC characters are lower than numbers. This is shown in the sample sort below.

```

CODE:
proc sort data=claims out=descnag.clms9306 ;
  by eessn relation dobkkey ;
run ;

```

OUTPUT ON WINDOWS (ASCII Sort Seq):

OBS	EESSN	REL	DSEX	DOB	DOBKKEY
1	000000100	2	2	2822	.
2	000000100	3	2	10733	10733
3	000000566	3	1	9136	9136
4	000000717	2	2	-5353	.

OUTPUT ON MVS (EBCDIC Sort Seq):

OBS	EESSN	REL	DSEX	DOB	DOBKKEY
1	BADSSN305	3	2	10534	10534
2	000000100	2	2	2822	.
3	000000100	3	2	10733	10733
4	000000566	3	1	9136	9136

Luckily, SAS also foresaw this as a problem and included a simple tool to help out the SORTSEQ option on PROC SORT. While there are others, the two options of SORTSEQ which apply here are SORTSEQ=ASCII or SORTSEQ=EBCDIC. By simply setting SORTSEQ=&DTYPE. in all PROC SORT's, you can standardize the order in which your data will appear when sorted across operating systems.

VII. SYSTEM RESTRICTIONS

In referring to "system restrictions" we mean system dependent issues which can't be isolated and so limit how you write or implement your SAS code. While there are probably many more of these system pitfalls which could be identified, the three we found in the DES system were; the limits of Tape processing on MVS, the limits of Batch processing on MVS, and how to incorporate the JCL on MVS.

Of these three areas, the incorporation of the JCL was the least limiting on the code. Having isolated the external file references in the SAS LIBNAME and FILENAME statements, the only purpose for the JCL was to set up the job in the batch processor, invoke SAS, and then read in the main program. In order To accomplish this I used the following simple JCL:

```

//T0077NH JOB (HEACLT67H022),
//          'M9312 CNAG DEPSIM',
//          CLASS=H,
//          TIME=10,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1),
//          NOTIFY=T0077NH
/*JOBPARM ROOM=NBI
/*ROUTE PRINT DFIELD
//*****
/* MEMBER NAME : DESJCL
/* AUTHOR      : RYAN CARR
/* CREATION DATE: 02-27-94
/* DESCRIPTION : JCL FOR DEPENDENT SIMULATION
/*              USE THIS MEMBER TO CALL EITHER
/*              DESTABLE, ORDESMAN
//*****
//SAS        EXEC SAS
//SYSIN      DD DSN=NH.TEST.DES.HEA(DESMAIN),

```

```
//          DISP=SHR
/*
```

The only difference between this JCL and our standard system JCL for SAS programs is in the SYSIN DD statement where we associate an actual program name instead of using * to read inline information.

It is probably appropriate to mention one limit of the SAS system which can be compensated for in the JCL here. When associating multiple input files with one DDNAME, or FILEREF, SAS does not support the MVS DD option AFF=DDNAME. This allows you to associate only one tape drive with multiple input files which are to be concatenated as in the following statement:

```
//GENERIC DD DSN=NH.D76.SPC.CNAG.C3.M9212.GENERIC,
//          DISP=OLD
//          DD DSN=NH.D76.SPC.CNAG.C3.M9303.GENERIC,
//          DISP=OLD,UNIT=AFF=GENERIC
//          DD DSN=NH.D76.SPC.CNAG.C3.M9306.GENERIC,
//          DISP=OLD,UNIT=AFF=GENERIC
//          DD DSN=NH.D76.SPC.CNAG.C3.M9309.GENERIC,
//          DISP=OLD,UNIT=AFF=GENERIC
//          DD DSN=NH.D76.SPC.CNAG.C3.M9312.GENERIC,
//          DISP=OLD,UNIT=AFF=GENERIC
```

This option may be significant when reading in a large number of input datasets, since SAS's FILENAME statement will attempt to mount all of the tapes at once which may be impossible if enough tape drives are not available. If this is a problem for you, just issue a DD statement similar to the one above in the JCL calling your SAS program to override the SAS FILENAME statement.

Since DASD is scarce at our site, we often use tape processing on MVS. Processing SAS datasets on tape imposes many limits on coding techniques. The most notable of these is the restriction of being able to use only one dataset from a specific tape library at a time as either input or output. This restricts such common SAS practices such as sorting a dataset to itself, or setting a dataset into itself, or even into another dataset in the same library as below:

```
data descnag.enrb9306 ;
  merge descnag.enra9306(in=E)
        descnag.depn9306(in=D) ;
  by eessn ;
  if e ;
  if not d then do ;
    spouse = 0 ;
    childw_c = 0 ;
  end ;
run ;
```

We dealt with this simply through extensively using work datasets. The modified versions of the code segments above appear in the DES system like this:

```
proc sort data=des.enra9306 out=enra ;
  by eessn ;
run ;

proc sort data=des.depn9306 out=depn ;
  by eessn ;
run ;

data des.enrb9306 ;
  merge enra(in=E)
        depn(in=D) ;
  by eessn ;
  if e ;
  if not d then do ;
    spouse = 0 ;
    childw_c = 0 ;
  end ;
run ;
```

While you could set up an elaborate system to execute segments of the above code alternatively according to the automatic variable &SYSSCP, and some other variable indicating disk or tape processing, for my purposes we felt this wasn't necessary.

The final limit encountered was due to the implementation of the system in MVS batch mode. We had originally wanted to present the system with a simple %WINDOW statement so that users would not have to see and change the code. Since this is impossible in a batch mode (can't display a window if there's no terminal) we substituted a control program with a series of %LET's.

```
Program:      DESMain.sas
Purpose:      This program controls the flow of other programs in the
               Dependent Enrollment Simulation. It sets up a macro
               library and then calls each functional macro to create
               a complete enrollment file with dependents from an
               enrollment file with only employee enrollment and the
               corresponding dependent claims file(s).

Input progs:  cin          ein          moex          dcnt
              cmrg         tf_(clt)     tier          sime
              smc          wentr

Output:       desmain.log   desmain.lst

Update Log:   Notes                                               Date      Programmer
-----
Original version.                                           Dec 16, 1993 Ryan Carr
Added info for MVS.                                         Feb 12, 1994 Ryan Carr
Added calls to tier() and tf_(clt)                         Mar 31, 1994 Ryan Carr

Usage Notes: GENERAL Set _c as the Client value (like CNAG)
                     _car as the Carrier value (like AETHN)
                     _y as the year for the processing period
                     _m as the month of the processing period
                     _mod as the processing mode (PRINT/PROD/TEST)
                     _cst as the first claim date being processed
                     _est as the first enrollment date in the file

Dates should be entered like '01jan94'd

Call each macro program with tname or comment
it out if you are re-running only part of the
program by using 'mname, where mname is the
macro program's name.
```

```

.....
;
options nosource nonotes ;

%let _c=CNAG ;
%let _y=93 ;
%let _m=06 ;
%let _car=UNIT ;
%let _cst='01JUL89'd ;
%let _est='01JUL89'd ;
%let _t= ;
%let _dt=ebodic ;

```

While not as pretty as a nice %WINDOW, it is functional. However with some additional code and no changes to existing code, we can easily implement a combination of the %LET's and a %WINDOW at a future date.

VIII. SUMMARY

One point we wish to emphasize here is that this application is not intended to be client-server by any means. It is however designed to be easily portable between multiple operating systems. While this is by no means a comprehensive list of problems one may encounter when attempting to write code to run on multiple operating systems, it is a good start. The flexibility in choosing what platform to run this system on has been an advantage to us in at least three ways so far. The first is of course a shortened development cycle. This was achieved by running the preliminary test data on PC, eliminating the waiting time in long queues we often experience on the mainframe. Next is the ability to run small to medium size client data through the system on the PC, which lowers our dependence on the overcrowded mainframe. The final, least tangible, and perhaps most valuable benefit is the ability to easily transport the application to whatever the final platform is with minimal or no changes.

REFERENCES

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.