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 ENRRAW "NH.D76.SPC.CNAG.E3.M9306.STD"
DISP=OLD ;

OS
FILENAME ENRRAW "enr9306 data a"
DISP=OLD ;

Windows or OS/2
FILENAME ENRRAW "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 multiple sets of code, one for each operating system. A simpler solution was the use of &SYSSCP to maintain the set of code, and updating all other versions. A simpler system Without using &SYSSCP you would have to maintain SAS to choose which to run, depending on what operating system is being used. 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.

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 ;
libname descx xport 'c:\work\des\utility.tpt' ;
select function descx csex descx Clairn ;
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:

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

If you are naming 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 CJMPORT, 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:

```sql
//TO0770BN JOB (HARCHPG61012),
// 'RYAN CARR DESD001','
// CLASS=A,
// MSGCLASS=X,
// NOTIFY=T0077NH,
// JOBPARM P=PROC01,ROOM=CHP,
/*JOBFORM P=PROC01,ROOM=CHP
*/
/*COPYDEN PROC 1=1
/*COPY EXEC PGM=IEBGENER
/*SYSOUT DD SYSOUT=* ,
/*SYSOUT2 DD DSN=D0SN,
/*UNIT=CARC,
/*VOLN=RETAIN,,SER=423000,
/*EXEPT=9E00,
/*DCB=(FORMAT=NOCOMP)
/*SYSIN DD DSN=099,
/*FEND
/*S1 EXEC COPYDEN,
/*DSN='NH.D76.SPC.CAPN.C3.M9306.GENERIC' ,
/*ARHM=ARH3071
/*S EXEC COPYDEN,
/*DSN='NH.D76.SPC.CAPN.E3.M9306STD.MED',
/*ARHM=ARH075
```

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:

```sas
data claims ;
infile genraw ireel=349;
drop dobe ;
input @B7 rel $ebcdic.
if rel in ('2','3') ;
ininput @77 EESSN @B7 COY @96 DOSE @117 esEX $ebcdic0.
if dobe not in ('00000000', 'dob') = input{dobc,yymmdd} ;
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, $ebcdic8. to read packed decimal data, and $ebcdic6. 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 %dttype. = EBCDIC and %sysrcp. = WIN then %do;
  %let c=$ebcdic ;
  %let z=$s310fzd ;
  %let p=$s310fpd ;
  %end;
else %do;
  %let c=$ ;
  %let n= ;
  %let z= ;
  %let p= pd ;
tend ;
data claims;
  infile genraw lrec=349 ;
  drop dobe ;
  input @81 rei &c.l. @ if reI in (. 2' I ' 3') ;
  input @71 EESSN @87 COV @96 DOBe @1l7 DSEX 'c.lO. &p.4. &c.8. &z.1.
  if dobe not in {'OOO00000', 'dob'} then do;
    if rei = '3' then dobkey = dob ;
  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 &DTYFE. 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.

VII 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.

```
DATA CLMBS27; SET CLAIMS;
  CLMBS=trim(DOB);
  DOB=put(DOB,'8YMONDD.2.');
  CLMBS=trim(CLMB5);
  CLMB5=put(CLMB5,'6Z.2.');
run;
```

The only difference between this JCL and our standard system JCL for SAS programs is in the SYSIN DD.

```
/* Descriptive comments */
/* JCL for standalone simulations */
/* This JCL is designed for running standalone simulations */
/* on MVS systems. */
/* The JCL allows for the execution of the simulation program */
/* and the preparation of the output file. */
/* */
/* OPTIONS */
/* */
/* PROCEDURE */
/* */
/* DATABASE */
/* */
/* EXEC */
/* */
/* INPUT */
/* */
/* OUTPUT */
/* */
/* END */
/* */
/* EXEC */
/* */
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.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 library as below:

```
data descnag.enrb9306 ;
merge descnag.enrb9306(in=E) descnag.enrb9306(in=D) ;
by eesn ;
if d = 1 ;
  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.depn9306 out=depn ;
by eesn ;
run ;
data des.enrb9306 ;
merge enrb9306(in=E) depn(in=D) ;
by eesn ;
if c = 0 ;
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 %LETs.

```
Program: %WINDOW, the
Purpose: This program controls the flow of other programs in the
         Dependence Enrollment Simulation. It sets up a main
         library and then calls each functional routine to create
         a complete enrollment file with dependents and the
         corresponding dependent claims files.
Input params: in out
             depn_depn
             date
Output: enrb9306
code: data descnag.enrb9306
```

```
while 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 library as below:

```
data descnag.enrb9306 ;
merge descnag.enrb9306(in=E) descnag.enrb9306(in=D) ;
by eesn ;
if d = 1 ;
  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.depn9306 out=depn ;
by eesn ;
run ;
data des.enrb9306 ;
merge enrb9306(in=E) depn(in=D) ;
by eesn ;
if c = 0 ;
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 %LETs.

```
Program: %WINDOW, the
Purpose: This program controls the flow of other programs in the
         Dependence Enrollment Simulation. It sets up a main
         library and then calls each functional routine to create
         a complete enrollment file with dependents and the
         corresponding dependent claims files.
Input params: in out
             depn_depn
             date
Output: enrb9306
code: data descnag.enrb9306
```

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.