

Avoiding Platform-Dependent Code

Pete Anderson, Quintiles Inc., Kansas City, MO

ABSTRACT

Platform-dependent code makes it difficult to work on more than one platform or transport a SAS program to another system. While it is not possible to discuss all of the issues that can arise when transporting code, some features of SAS will be presented that allow more robust code to be developed. The discussion will include: the portability of certain characters, how to query and utilize the contents of a directory, and macro programming techniques. Examples of platform-dependent code as well as potential alternatives will be presented.

INTRODUCTION

Change is certain. Whether it is due to a company merger or simply the nature of our business, we can find ourselves maintaining code on multiple platforms or migrating code to new platforms. And no one would want to be the programmer responsible for maintaining two or more versions of a program.

The good news is that SAS is a collection of rich and diverse languages, and as such, generally allows more than one approach to solve a programming task. In this paper, we will outline methods for tackling some of the situations we have encountered.

PIPE COMMANDS

Occasionally, you may need to retrieve the names of the members in a directory. One way to approach this issue is with the use of a pipe command:

```
filename pipedir pipe "dir";  
filename pipedir pipe "ls";
```

While you can use this approach, the command for retrieving the contents of a directory is platform-dependent (for example, **dir** under MS-DOS and **ls** under UNIX). A more flexible approach involves the data step functions FILENAME, DOPEN, DNUM, DREAD, and DCLOSE.

```
rc=filename('dirloc','&in');  
dirid=dopen('dirloc');  
numsel=dnum(dirid);  
do i=1 to numsel;  
  filename=dread(dirid,i);  
  put filename;  
end;  
rc=dclose(dirid);
```

Please note that there is an issue with the DCLOSE function in some versions of SAS, you cannot have a self-referencing variable such as:

```
dirid=dclose(dirid);
```

Other useful data step functions are documented in Table A.3.1, "Function Arguments for %SYSFUNC", *SAS Macro Language*, pp. 377-380.

X STATEMENTS

It is possible to use an X statement in a SAS program to issue a host command. For instance, you can use an X statement to create a directory or execute a script. One such script might be used for post-processing a listing to add Page X of Y descriptors. However, since SAS possesses a wide array of tools for managing external files, it is also feasible to write a SAS macro program to post-process a listing to add Page X of Y descriptors.

A general approach for such a macro would involve reading the listing into a SAS data set. Simultaneously, you would check for occurrences of new pages and increment a page count variable appropriately. We have experienced occasional problems when writing directly out to the same file name, but if you use the FDELETE data step function to delete the existing file first, the problem is circumvented.

ANOTHER ALTERNATIVE

A discussion on avoiding platform-dependent code would not be complete without mentioning the SAS macro variable SYSSCP. This macro variable contains a string representing the platform on which SAS is executing (e.g. WIN for Windows). While setting up splitter logic in a macro program based on the value of this variable doesn't avoid platform-dependent code, it is useful for setting up logic that will operate on more than one platform. Generally, we use this variable in a utility macro program to determine the path delimiter (the colon for the Macintosh, a backslash for Windows, a slash for UNIX, and a right bracket for VMS). Having a separate utility macro to do this is beneficial because if you need to work on a new platform, you have isolated and limited the number of occurrences of code that would need to be modified.

SPECIAL CHARACTERS

Certain characters, generally from the extended ASCII character set, are not consistently usable across applications and platforms. For example, the caret symbol (^) is used in SAS to represent either a logical NOT or non-equality when used in conjunction with an equals sign (^=). However, this symbol doesn't consistently transfer across platforms, therefore, a programmer may want to use the NOT function and the mnemonic operator NE (not equal to) instead. Other characters that are known to be problematic include the vertical pipe (an exclamation mark may be substituted), as well as brackets and braces (parentheses may be an alternative).

FILENAME AND LIBNAME STATEMENTS

Avoid the use of FILENAME and LIBNAME statements in your programs. Instead, use an initialization file to specify platform-dependent file and data locations. This reduces the number of programs that have to be modified. For example, we use macros to add titles and footnotes to our reports – rather than hard-coding the location of these files, we assume the project will have a *fileref* called TITLE and FOOT, respectively (as defined in an initialization file).

EDITORS

Depending upon which platform and editors you use, we have found it helpful to begin lines in column two. For example, vi on UNIX exits out of insert mode in some instances when your cursor goes to column one. Likewise, JCL (Job Control Language) under MVS considers a slash character in column one immediately followed by an asterisk (/*) as the end-of-file marker.

CONCLUSION

These are just some of the cases that can arise when writing platform-independent code in SAS. We hope this paper has been of help.

ACKNOWLEDGMENTS

Thank you to Lori Griffin, George Hage, and John Morrill for their review.

REFERENCES

SAS Institute Inc., *SAS Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc., 1997. 304 pp.

SAS Institute Inc., *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 1990. 1042 pp.

CONTACT INFORMATION

Pete Anderson
Quintiles, Inc.
P. O. Box 9708
Kansas City, MO 64134-0708
Work Phone: 816-767-6416
FAX: 816-767-7331
E-mail: pete.anderson@quintiles.com