

SAS® and UNIX: Techniques for Developing Your Toolbox

Joe Novotny, GlaxoSmithKline Pharmaceuticals, Inc., Collegeville, PA

ABSTRACT

How many times have you had to write and run short SAS programs to determine the contents of a SAS data set or determine a simple frequency count of a variable? What if you could perform these tasks with a few simple keystrokes from the UNIX command line? Have you ever needed to create a SAS data set containing file information for numerous SAS files existing in a UNIX directory? This paper highlights several useful SAS features you should be aware of to take advantage of SAS's ability to interface with UNIX. The paper demonstrates practical applications of: 1) reading the UNIX command line into a SAS program, 2) printing SAS output to the UNIX terminal screen and 3) techniques that allow you to utilize UNIX information and execute UNIX commands from within SAS programs. These techniques can be used to automate many daily tasks, simplify more complex tasks and increase your overall programming productivity.

INTRODUCTION

Many companies have chosen UNIX as the operating platform and working environment of choice for SAS code development. Along with the benefits of using the UNIX system itself, SAS offers many techniques for utilizing UNIX functionality within the SAS language which enable programmers to efficiently transfer useful information between SAS and UNIX systems. This paper discusses a number of these techniques and demonstrates practical applications using them. Topics covered include: 1) Piping UNIX command line information into a SAS data step using the INFILE statement, 2) Using the FILENAME statement with the TERMINAL argument and PROC PRINTTO to route SAS output directly to the UNIX terminal, 3) executing UNIX commands from within a SAS program using the X statement, the CALL SYSTEM routine and the %SYSEXEC MACRO statements, 4) using UNIX environment variables within SAS programs.

Background and Assumptions

1. I assume readers are familiar with basic concepts of the UNIX environment (e.g., UNIX command line, basic UNIX commands, directory structures, environment variables, the keyboard as standard input, the terminal screen as standard output, etc.) or at least have an interest in learning about them. I do not assume readers are power users or shell scripting gurus. You will benefit if you are looking to augment your understanding of how SAS and UNIX can communicate. The focus is on how SAS can utilize UNIX information to facilitate your SAS programming.
2. I assume readers have an intermediate or greater level of understanding of Base SAS and SAS MACRO.
3. Unless otherwise noted, the UNIX command line examples in this paper (denoted w/ the greater than sign ">") are run using tcsh shell syntax to interface with UNIX. Tcsh is a C shell variant. Some UNIX commands may have slightly different syntax in other UNIX shells such as Korn, Bash, etc. although most commands referenced in this paper are basic commands such as "ls -l".

PIPING COMMAND LINE INFORMATION INTO YOUR SAS PROGRAMS AND SENDING OUTPUT TO THE TERMINAL

PROBLEM: How many times have you had to write and run short SAS programs to determine the contents of a SAS data set or determine a simple frequency count of a variable? Over the lifespan of a project you may need to remind yourself of variable names, data types, lengths, labels, etc. numerous times. You are probably not making the best use of your time if you spend much of it opening up tmp.sas and typing something similar to the following:

```
libname mylib '/home/userid/mydata';
run;

proc contents data=mylib.mydsname;
run;
```

You then check that your tmp.log file contains no ERROR: or WARNING: messages, open up tmp.lst and scroll down to search for the variable you are looking for. This seems a small task. But add it up for each data set, perhaps many times over the lifespan of a project, and you probably start thinking there must be a better way to do this.

SOLUTION 1: One way to avoid this repetitive work is to write a simple little macro that does three basic things: 1) reads what you type at the UNIX command line into a SAS program, 2) does the SAS work for you and 3) sends the output to your terminal screen. After the initial code development, all this can be done without having to touch the keyboard again after typing a few words and hitting enter. The example macro contents.sas below performs these operations. In the example, I simply type the following at the UNIX command prompt:

```
> echo mydsname | sas contents
```

and the contents macro does the rest.

```
1  %macro contents;
2
3      data _null_;
4          infile stdin;
5          length ds $ 200;
6          input ds;
7          call symput("ds",compress(ds));
8      run;
9
10     libname tmpcont '.'; run;
11
12     proc contents data=tmpcont.&ds. noprint out=tmpcont;
13     run;
14
15     filename term terminal; run;
16
17     proc format;
18         value charnum 1='Num'
19                     2='Char' ;
20     run;
21
22     proc printto new print=term; run;
23
24     proc print data=tmpcont noobs;
25         var memname nobs name type length label;
26         format type charnum.;
27     run;
28
29     proc printto; run;
30
31 %mend contents;
32 %contents;
```

Line 4 uses the INFILE statement to read in UNIX standard input.

Line 7 uses the CALL SYMPUT routine to create a macro variable containing the name of my data set, in this case mydsname. I can then use this macro variable within the program to refer to the data set of interest.

Line 10 assigns a LIBNAME to the current directory (Note that the code then functions only when run in the same directory as the existing data set. I'll show one way to increase flexibility by using a UNIX shell script later in the paper).

Line 12 uses the CONTENTS procedure to generate a working data set containing the contents information about the permanent data set.

Line 15 uses the FILENAME statement to assign a FILEREf of the terminal screen for use as our output destination later.

Lines 17-20 use the FORMAT procedure to create a format through which to view the TYPE variable since it is output from the CONTENTS procedure in numeric codes of 1 and 2.

Line 22 uses the PRINTTO procedure to send all printed output to the "term" FILEREf assigned previously.

Lines 24-27 use the PRINT procedure to display the required information.

Line 29 closes the PRINTTO procedure.

To increase this program's flexibility, a simple UNIX shell script can be used to enable the SAS MACRO to be called from any directory (provided the data set exists in the directory and directory holding the shell script is found in your UNIX \$PATH variable). This ensures that program functionality is no longer dependent on the SAS program and the SAS data set residing in the same directory and allows you to type the following at the UNIX command line:

```
> contents mydsname
```

and receive the requested information printed directly to the UNIX terminal screen. Code for the UNIX shell script named 'contents' above is presented below:

```
1  #! /bin/ksh
2
3  if (( $# != 1 ))
4  then
5      echo
6      echo Please enter the name of a single data set from the current directory\
7      echo
8  else
9      echo $* | sas $HOME/code/contents -log /tmp
10     rm -f /tmp/contents.log
11 fi
```

Line 1 establishes that the shell language to be used is the Korn shell.

Lines 3-7 perform some checking to ensure that only one data set is passed to the script. \$# will resolve to the number of arguments passed from the command line to the shell script (the name of the script itself is not counted, so in the example above \$# resolves to 1).

Line 9 \$* resolves to display all information passed to the script [again, the script itself is not included, so in this example, \$* resolves to the text string "mydsname" (without the double quotes)] and pipes it into the command which executes SAS on the contents.sas program residing in the user's \$HOME/code directory. It also sends the SAS log to the /tmp directory (note that this implies write access to the /tmp directory).

Line 10 cleans up by removing the log file produced by the SAS program. During code development, this is done only after you have verified no further debugging is needed.

Line 11 ends the if loop started on line 3.

SOLUTION 2: To simplify the SAS program using another of SAS's UNIX interface capabilities, the -SYSPARM option can be used when invoking SAS. Using this option populates the automatic macro variable SYSPARM with the text enclosed in quotes (see below). At the command line, type:

```
> sas -sysparm 'mydsname' contents
```

The SYSPARM macro variable is populated with 'mydsname' and we eliminate the need to use the DATA step and CALL SYMPUT to create the macro variable containing the data set name:

```
1  %macro contents;
2
3      libname tmpcont '.'; run;
4
5      proc contents data=tmpcont.&sysparm noprint out=tmpcont;
6      run;
7
8      filename term terminal; run;
9
10     proc format;
11         value charnum 1='Num'
12                     2='Char' ;
13     run;
14
```

```

15      proc printto new print=term; run;
16
17      proc print data=tmpcont noobs;
18          var memname nobs name type length label;
19      run;
20
21      proc printto; run;
22
23  %mend contents;
24  %contents;

```

This solution also requires a slight modification to the UNIX shell script in order to run the 'contents mydsname' command from the UNIX command line. The required changes are highlighted on line 9 below:

```

1  #! /bin/ksh
2
3  if (( $# != 1 ))
4  then
5      echo
6      echo Please enter the name of a single data set from the current directory\
7      echo
8  else
9      sas -sysparm $* $HOME/code/contents -log /tmp
10     rm -f /tmp/contents.log
11 fi

```

Note that while the use of the `-sysparm` technique above is more efficient for passing a single data set to the SAS program, passing more than a single parameter to the SAS program via the UNIX command line may require adding a bit more complexity to your SAS program and/or the use of the DATA step for reading the information into SAS. For example, creating a similar utility program using PROC FREQ to produce a cross-tabulation of multiple variables may require code to parse the following: `"var1\*var2\*var3"`. You must use the escape character `"\"` to prevent UNIX from interpreting the asterisk as a special character on the command line.

With a bit of creativity, you can design utility programs that can be used to simplify many of the everyday tasks used in getting to know our data (e.g., PROC FREQ, PROC UNIVARIATE, etc.). These techniques can reduce the amount of redundant coding required and completely eliminate many common coding errors due to typos or misplaced semicolons.

EXECUTING UNIX COMMANDS WITHIN SAS PROGRAMS

In addition to receiving UNIX information from the command line, SAS can also interface with UNIX by executing UNIX commands directly from within your current SAS session. In this section I will discuss using the X statement, the CALL SYSTEM routine and the %SYSEXEC MACRO statement to run UNIX commands within SAS programs.

PROBLEM: You need to populate a SAS data set with metadata information from the files in a given UNIX directory (e.g., filenames, date/time of last modification, etc.). This can be useful for management of SAS programs and output in the UNIX production environment. The particular business need in the author's case was to create a data set to be used as a driver file for an application archiving SAS output into a document repository.

SOLUTION 1: The required file information can be obtained by storing the output from the UNIX `"ls -l"` command into a permanent file and then reading the information in this file into a SAS data set as shown below.

```
> ls -l > myfiles.txt
```

For this example, myfile.txt now contains the following information:

```

total 3588
-rw-r--r--  1 myid9999 mygroup    836333 Jun 15 10:27 file1.lst
-rw-r--r--  1 myid9999 mygroup    70919 Jun 15 10:27 file2.lst
-rw-r--r--  1 myid9999 mygroup    26467 Jun 15 10:27 file3.lst
-rw-r--r--  1 myid9999 mygroup   152463 Jun 15 10:27 file4.lst
-rw-r--r--  1 myid9999 mygroup   556031 Jun 15 10:27 file5.lst

```

```
-rw-r--r-- 1 myid9999 mygroup 192752 Jun 15 10:27 file6.lst
-rw-r--r-- 1 myid9999 mygroup 0 Jun 15 14:03 myfile.txt
```

Both the first line of the file (total 3588, the total block count) and the last line (containing information for the myfiles.txt file) represent unwanted information for our purposes. To eliminate this and make the file more easily readable by SAS, we can manually delete the first and last lines of myfiles.txt. We can then read the remaining information into SAS with the following DATA step :

```
1 data myfiles;
2   infile './myfiles.txt' lrecl=400;
3   length permiss filelink owner group size month day time $20 filename $200;
4   input permiss filelink owner group size month day time filename $;
5   run;
```

Results of the PRINT procedure for the resulting data set are shown below:

Obs	PERMISS	FILELINK	OWNER	GROUP	SIZE	MONTH	DAY	TIME	FILENAME
1	-rw-r--r--	1	myid9999	mygroup	836333	Jun	15	10:27	file1.lst
2	-rw-r--r--	1	myid9999	mygroup	70919	Jun	15	10:27	file2.lst
3	-rw-r--r--	1	myid9999	mygroup	26467	Jun	15	10:27	file3.lst
4	-rw-r--r--	1	myid9999	mygroup	152463	Jun	15	10:27	file4.lst
5	-rw-r--r--	1	myid9999	mygroup	556031	Jun	15	10:27	file5.lst
6	-rw-r--r--	1	myid9999	mygroup	192752	Jun	15	10:27	file6.lst

From this point, we can use the information just like any other SAS data set. Note that two manual steps were used to generate our input file for this task: 1) the UNIX command to create it and 2) file editing to allow easier input to SAS. For a single iteration of this process, this represents two points of human contact where errors may be introduced. If the task is to be repeated as new files are added or the current files are updated, the possibility for error increases. A higher degree of validation and repeatability can be achieved if the process is automated. Solution 2 below presents a more automated solution.

SOLUTION 2: We can automate the process described above by using SAS's ability to execute UNIX commands directly from a SAS session. The X statement, the CALL SYSTEM routine and the %SYSEXEC MACRO statements allow us to do this. Instead of manually creating the myfile.txt file above, we can create it and remove it on the fly using the X statement as shown below.

```
1 x ls -l . | tail +2 > myfiles.txt;
2
3   data myfiles;
4     infile 'myfiles.txt' ;
5     length permiss filelink owner group size month day time $20 filename $200;
6     input permiss filelink owner group size month day time filename $;
7     if not(index(filename,'myfiles')) and not(index(filename,'readfiles'));
8   run;
9
10 x rm -f myfiles.txt;
```

Line 1 uses the X statement to execute the UNIX ls -l command within the SAS session. By piping the output of this command through the "tail +2" UNIX command, we read everything from the "ls -l" command, starting at the second line (which eliminates the total block count), into myfiles.txt.

Lines 3-6 read the file, assign attributes and input the information into the DATA step.

Line 7 subsets the output data set to remove the records for the myfiles.txt file (created by line 1) and this running SAS program (called readfiles in this example)

Line 10 programmatically removes the myfiles.txt file using the X statement to execute the UNIX rm command on the file (the -f option on the rm command eliminates the need to respond to the UNIX prompt asking for confirmation prior to removing the file. Without the -f option, the prompt is sent to the screen and requires user input prior to finishing the SAS session).

The %SYSEXEC MACRO statement allows you to execute these same tasks using a slightly different syntax for lines 1 and 10

above:

```
1 %sysexec(ls -l . | tail +2 > myfiles.txt);  
.  
.  
.  
.  
.  
10 %sysexec(rm myfiles.txt);
```

Both the X statement and the %SYSEXEC MACRO statement cause the UNIX command to execute immediately. Similarly, both result in the assignment of operating environment return codes to the SAS automatic macro variable SYSRC.

The above tasks can also be performed by using the CALL SYSTEM routine to execute the UNIX commands within SAS. The significant difference between using CALL SYSTEM and using the X or %SYSEXEC MACRO statements is that the CALL SYSTEM routine must be run within a DATA step. One of the benefits of this is that it implies the UNIX commands can be run conditionally if desired (using familiar SAS syntax as opposed to shell scripting language). An example of using the CALL SYSTEM routine to perform one of the example tasks is shown below:

```
1 data _null_;  
2   call system('ls -l . | tail +2 > myfiles.txt');  
3 run;
```

SOLUTION 3: We can also eliminate the need to create a permanent file by streaming the output from the “ls -l” UNIX command directly into a SAS DATA step using the FILENAME statement with the pipe option. The DATA step looks similar to the above examples, with the exception that instead of reading data from a physical file, we read the information into the DATA step from a data stream that never produces a hard file. So there is no need to create it, subset the output data set for the myfiles.txt file (as we did above) or remove any files from the UNIX environment.

```
1   filename mylist pipe "ls -l . | tail +2"; run;  
2  
3   data myfiles;  
4     infile mylist lrecl=400;  
5     length permiss filelink owner group size month day time $20 filename $200;  
6     input permiss filelink owner group size month day time filename $;  
7     if not(index(filename,'readfiles'));  
8     run;
```

Solutions one through three all produce the same final working MYFILES data set using differing levels of complexity and having different degrees of flexibility. Each may be better suited to certain specific tasks than the others depending on your needs and preferences.

USING UNIX ENVIRONMENT VARIABLES WITHIN SAS PROGRAMS

In your UNIX production environment, you probably have many system environment variables that can be utilized to make your SAS code more efficient and flexible. You can use the %SYSGET MACRO function to make use of the values of UNIX environment variables.

PROBLEM 1: You need to assign a SAS library reference to work with data in a directory with a long fully-qualified path name.

SOLUTION: You can use SAS’s ability to retrieve the values of environment variables to populate LIBREFs for use in data retrieval.

For example, you may have data which reside in the following UNIX directory:

```
/prod/projid/lots/of/directories/to/get/to/my/data
```

A UNIX environment variable may exist containing the name of this directory. For example, if you have an environment variable named DATAPATH that refers to the above directory, you can use the %SYSGET MACRO function to retrieve this information and assign it to a SAS LIBREF as shown below.

```
1 libname mydata "%sysget(DATAPATH)";
```

```
2
3 data work.mydataset;
4     set mydata.mydataset;
5 run;
```

This simple use of %SYSGET to retrieve environment variable values can help you eliminate the need to make numerous libname assignments. MACRO code can then be developed that refers to this environment variable. The SAS MACRO will then function identically for various projects with a simple reassignment of the UNIX environment variable, eliminating the need to reassign your LIBNAMEs for new projects.

The uses of environment variables through SAS are far-reaching. In addition to populating LIBNAMEs with the values of environment variables, you can use their values to execute code conditionally. For example, your UNIX environment may contain a MODE environment variable indicating whether your login is in user mode or production mode. Your SAS macros can be crafted such that specific sections of code switch on or off depending on whether code is being executed in user mode or production mode. Additionally, your UNIX system probably has an environment variable indicating the user id of the user logged into the current session. The USER environment variable may be used in the creation of system log files to aid in creation of an audit trail to track production activity.

CONCLUSION

With a little creativity and some basic knowledge of UNIX and SAS, you can develop some simple SAS MACROs to help eliminate, or at least minimize the time you spend performing, several of the more mundane tasks of programming. By standardizing some of the techniques presented here in macro code libraries, small improvements in efficiency can multiply through use by many programmers over the course of large-scale projects to produce large-scale benefits. Even at the individual level, small incremental improvements multiplied, improved upon and expanded over the course of a programming career can result in significant impact on your ability to produce high quality code and contribute to team efforts.

REFERENCES

Gleick, James (1987), *Chaos: Making a New Science*, Penguin Books

Peek, Jerry, O'Reilly, Tim and Loukides, Mike (1997), *UNIX Power Tools*, Sebastopol, CA: O'Reilly & Associates, Inc.

SAS Institute Inc. (1999), SAS OnlineDoc® documentation, Version 8, Cary NC

CONTACT INFORMATION

Joe Novotny
GlaxoSmithKline
1250 South Collegeville Rd.
Collegeville, PA 19468
Phone: (610) 917 – 6939
Fax: (610) 917 - 4701
Email: joe.2.novotny@gsk.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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