Paper 047-2012

Working the System: Our Best SAS® Options

Patrick Thornton, SRI International, Menlo Park, CA

Iuliana Barbalau, Adecco, Pleasanton, CA

ABSTRACT

This paper provides an overview of SAS system options and discusses the best options. These are options that are beneficial, interesting, or both. There are many ways to work the SAS system by changing the settings of one or more of hundreds of system options. These settings can have sweeping effects that influence DATA steps (REPLACE and MERGENOBY=), the Log (MSGLEVEL=), data sets (VALIDVARNAME), listing output (NODATE, NONUMBER, ORIENTATION=, and FORMDLIM=), and other destinations (PDFSECURITY=). Option settings are invaluable for working with format catalogs (FMTSEARCH=), and they are essential for using, developing, and debugging macro programs (MPRINT, MLOGIC, MPRINT, SYMBOLGEN, and, new to SAS 9.2, MCOMPILE and MCOMPILENOTE).

INTRODUCTION

SAS system options may be used to effect sweeping session-level change across a wide variety of domains (Lin, 2009; Poll 2011), and there are several different ways to explore the options by classification (Heaton, 2003; Thornton, 2011). This paper demonstrates several ways to learn about system options and their settings, and discusses our best options—those we have found most beneficial, interesting or both (Table 1).

Table 1 A Summary of Some of Our Favorite System Options

Classification*	Option Name	General Form**
SAS FILES	REPLACE MERGENOBY VALIDVARNAME OBS	NOREPLACE =NOWARN | WARN | ERROR =V7 | UPCASE | ANY = MAX | [n]
Log Control	MSGLEVEL	= I | N
List Control	DATE NUMBER CENTER FORMDLIM=	NODATE NONUMBER NOCENTER ="" | " "
ODS PRINT & Security	ORIENTATION PDFSECURITY	=PORTRAIT | LANDSCAPE =NONE| LOW| HIGH
Environment	FMTSEARCH	= (catalog-specification-1... catalog-specification-n)
Macro	MPRINT MFILE MLOGIC SYMBOLGEN MCOMPILE MCOMPILENOTE	NOMPRINT NOMFILE MLOGIC | NOMLOGIC SYMBOLGEN | NOSYMBOLGEN MCOMPILE | NOMCOMPILE MCOMPILENOTE=NONE | AUTOCALL | ALL

*Classifications were taken from DICTIONARY.OPTIONS; **Not all values of the options are discussed in the paper. General forms were found in SAS Institute, Inc (2011).

EXPLORE YOUR OPTIONS

There are a number of tools available to help you learn about system options (Poll, 2011). First, you can use the function GETOPTION to see the current value of an option. This is particularly useful before you have changed an option setting because you can easily see the system default. It's convenient to use GETOPTION with %SYSFUNC in order to see the option setting in the LOG. For example, the following syntax saves the current value of the PS (page size) option to the macro variable &MYOPTION and %PUT shows it in the LOG:

```
%let myoption=%sysfunc(getoption(ps,keyword));
%put Original Option Setting &myoption;
Option ps=50;
```

The text in the LOG is “Original Option Setting ps=56.” Since we used the argument KEYWORD with the GETOPTION function we obtained “ps=” in addition to value of 56, so the &MYOPTION macro variable can later be used to return PS to the default value

```
Option &myoption;
```

There are also a number of ways to learn the names, values and descriptions of system options. PROC OPTIONS provides extensive access to system options (Heaton, 2003; Poll, 2011). For example, we used PROC OPTIONS LISTGROUPS to list all the option groups in our LOG. We then copied the option group name of interest from the LOG and resubmit PROC OPTIONS.

```
Proc options listgroups;
Run;
Proc options group=macro value;
Run;
```

The second PROC OPTIONS created a list in the LOG of 26 macro-related options. The current setting, scope, and how the option was set was also listed in the LOG because we used procedure option VALUE.

You may also use the metadata in SAS DICTIONARY to explore categories of system options and their current settings (Thornton, 2011). PROC SQL provides direct real-time access to the options via DICTIONARY.OPTIONS, or SASHELP.VOPTION may be used with other procedures to return metadata from DICTIONARY.OPTIONS.

```
proc tabulate data=sashelp.voption;
    class group level opttype;
    table group, level*opttype*n;
run;
```

In SAS 9.2, the PROC TABLUATE syntax listed 29 categories of options classified on the GROUP variable. After looking at the categories we chose the category “PDF” and listed the details (Figure 1).

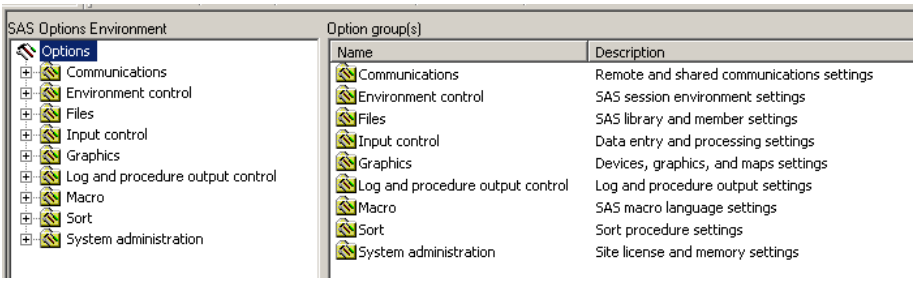
```
proc print data=sashelp.voption;
    var optname setting group optdesc;
    where group= 'PDF';
run;
```

Figure 1 List of PDF System Options Returned by VOPTION View

optname	setting	group	optdesc
PDFACCESS	PDFACCESS	PDF	Allow access to PDF documents
PDFASSEMBLY	NOPDFASSEMBLY	PDF	Allow PDF document assembly
PDFCOMMENT	NOPDFCOMMENT	PDF	Allow modification of PDF document comments
PDFCONTENT	NOPDFCONTENT	PDF	Allow modification of PDF document content
PDFCOPY	PDFCOPY	PDF	Allow copying of PDF document text and graphics
PDFFILLIN	PDFFILLIN	PDF	Allow fields in a PDF document to be filled
PDFPAGELAYOUT	DEFAULT	PDF	Identify the page layout for PDF documents
PDFPAGEVIEW	DEFAULT	PDF	Identify page viewing mode for PDF documents
PDFPRINT	HRES	PDF	Identify PDF document printing permissions

The SAS graphic user interface may also be used to explore the system options. As an example from a Windows OS, choose from the TOOLS menu, OPTIONS, and SYSTEM. Figure 2 shows the SAS Option Environment menu that will appear on the left of the screen. Choosing a category on the left causes the right panel to list the options in the category.

Figure 2 Groups of Options Available through the Menu



The rest of the paper discusses the options listed on the first page in Table 1.

SAS FILES AND LOG CONTROL

REPLACE

There may be times, especially when working with someone else's syntax, when it is informative to see the log of the submitted syntax without overwriting data sets. The NOREPLACE option will prevent the overwriting of permanent data sets. We saved SASHELP.SHOES to the permanent library MYDATA. Then in the following syntax we set the ❶ NOREPLACE option and try to overwrite that data set. The data set was not replaced and a warning was written in the log.

```
%let option = %sysfunc(getoption(replace,keyword));
%put Original Option &option;

*Prevent all permanent data sets from being replaced;
❶option noreplace;
data mydata.sasshoes;
    set sashelp.shoes;
run;
option &option;
```

Also shown in this example is the use of %SYSFUNC and GETOPTION. The first two lines of syntax obtain the original option setting in the macro variable OPTION, and show it in the log, and the last line resets the option to the original setting.

MERGNOBY

It is probably not very often that you would merge data sets without specifying a BY statement, so the ❷ MERGNOBY option is useful in controlling the behavior of the program when MERGE appears without a BY statement. When MERGNOBY=WARN a warning appears in the LOG, or when MERGNOBY=ERROR an error appears. A portion of the LOG generated by the syntax is shown in Figure 3. A warning message is shown in green.

```
option ❷ MERGENOBY=warn ❸ MSGLEVEL=i;
data mydata.sasshoes;
   set sashelp.shoes;
   run;
data mydata.sasshoes2;
   set sashelp.shoes;
   run;
data sashelpshoes;
      merge mydata.sasshoes mydata.sasshoes2;
run;
```

❸ MSGLEVEL=I (i.e. information) is also set in order to see the a listing of the variables that will be overwritten as a result of merging the two data sets. If you do not want the variables overwritten, you may find it convenient to copy these variables out of the LOG in order to get a jump on creating DROP or RENAME statements.

Figure 3 Log with MERGENOBY=WARN and MSGLEVEL=I

```
161      data sashelpshoes;
162         merge mydata.sasshoes mydata.sasshoes2;
163      run;

WARNING: No BY statement was specified for a MERGE statement.
INFO: The variable Region on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Product on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Subsidiary on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Stores on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Sales on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Inventory on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
INFO: The variable Returns on data set MYDATA.SASSHOES will be overwritten by data set
      MYDATA.SASSHOES2.
NOTE: There were 395 observations read from the data set MYDATA.SASSHOES.
NOTE: There were 395 observations read from the data set MYDATA.SASSHOES2.
NOTE: The data set WORK.SASHELPSHOES has 395 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

VALIDVARNAMES

The option VALIDVARNAMES is useful when working with data sources that have variables names that do not conform to the default SAS standard. For example, it is typical for the field names in databases to have more than one word in the name, for example “Date of Birth.” Under the default, if you were to import such a data set the variable would be renamed to DATE_OF_BIRTH. Renaming is often desired, but not always. For example, you may need to export the data back to the database and create a table with the original names. Setting the option VALIDVARNAMES=ANY allows SAS to import the data set without renaming the variables.

OBS

The option OBS is very useful for creating empty data sets or for efficiency in testing syntax on only a subset of the observations in large data sets. Empty data sets may serve as templates for data receive from outside of SAS. The following syntax shows how to use OBS to create an empty copy of the SASHELP.SHOES data set. OBS=MAX returns the option to its default.

```
option obs=0;
data sasshoes;
    set sashelp.shoes;
run;
option obs=max;
```

LIST CONTROL AND ODS PRINT

CENTER, DATE, NUMBER AND PDFSECURITY

By default the options CENTER, DATE, and NUMBER cause the output to be centered and to have a date, time and page number; however, these may be changed to allow further customization. The following syntax removes the default ❶ date, time and page number and adds a custom date, time and page number to the ❷TITLE1 of a PDF destination. Note that ODS ESCAPECHAR is necessary for the page numbering (see Thornton, 2010 for much more on ODS PFD) and we are using %SYSFUNC and PATHNAME to obtain the path for the permanent library MYDATA.

```
options ❶nonumber nodate nocenter ❸PDFSECURITY=high PDFPW=(OPEN=mypassword);
ods pdf file="%sysfunc(pathname(mydata)).\Mypdf.pdf";
ODS escapechar='^';
❷ title1 height=8pt j=right f=arial  "%sysfunc(date(),worddate.) Page ^{thispage}
of ^{lastpage} ";
proc print data=sashelp.shoes;
run;
ods pdf close;
options number date center PDFSECURITY=none;
```

❸The addition of the system options PDFSECURITY=high and PDFPW=(OPEN=mypassword) encrypts the document and prevents it from being viewed until the password (i.e. mypassword) is entered.

FORMDLIM

The option FORMDLIM saves space when running procedures to the list window. By setting FORMDLM=’ ‘(a blank space), procedures stack their output one after the other (Figure 4).

```
title "Stack my Output";
option FORMDLIM=' ' obs=2 date number ls=64;
proc print data=sashelp.shoes;
run;
title;
options nodate nonumber;
proc print data=sashelp.class;
run;
option FORMDLIM=''; obs=max date number ls=100;
```

Figure 4 Example of Stacking Procedure Output using FORMDLM

```
Stack my Output                    12:47 Saturday, August 6, 2011   34

Obs    Region    Product         Subsidiary     Stores

  1    Africa    Boot            Addis Ababa      12
  2    Africa    Men's Casual    Addis Ababa       4

Obs          Sales       Inventory        Returns

  1       $29,761        $191,821            $769
  2       $67,242        $118,036          $2,284


Obs     Name     Sex    Age    Height    Weight

  1    Alfred     M      14      69.0      112.5
  2    Alice      F      13      56.5       84.0
```

FMTSEARCH

By default SAS looks for formats in the catalogs WORK.FORMATS and LIBRARY.FORMATS (Lund, 2003). Changing the FMTSEARCH option causes SAS to look for a format in other format catalogs. The following example shows that FMSEARCH may be exploited by saving different formats with the same name each in different format catalogs, for example, WORK.FORMATS and MYDADA.FORMATS.

```
title; options nodate nonumber;
❶proc format lib=work.formats;
value salescut 0 - 38911 = 'Below the Median' 38912 - high = 'Median and Above';
run;
❷proc format lib=mydata.formats ;
value salescut 0 - 85700.16 = 'Below the Mean' 85700.17 - high = 'Mean and Above';
run;
data mydata.sasshoes;
  set sashelp.shoes;
❸ format sales salescut.;
run;
options formdlim=' ';
❹proc freq data = mydata.sasshoes notitle;
   table sales;
   title1 "Format from WORK.FORMATS";
run;
❺options   fmtsearch=(mydata.formats   work
library);
 proc freq data = mydata.sasshoes notitle;
   table sales;
   title1 "Format from MYDATA.FORMATS";
   run;
❻options      fmtsearch=(work      library)
formdlim='';
```

Figure 5 Changing the Format with FMTSEARCH

Format from WORK.FORMATS

Total Sales

Sales	Frequency	Percent
Below the Median	197	49.87
Median and Above	198	50.13

Format from MYDATA.FORMATS

Total Sales

Sales	Frequency	Percent
Below the Mean	277	70.13
Mean and Above	118	29.87

❶ The format SALESCUT was saved to the default catalog WORK.FORMATS, and ❷ a different format having the same name was saved to the catalog MYDATA.FORMATS. The format in the default catalog classifies sales below the median and median and above, and the format in the other catalog classifies sales below the mean and mean and above.❸ SALESCUT was assigned to the variable SALES, and because the default catalog is

WORK.FORMATS ❹ the first PROC FREQ counts the total sales classified on median. ❺ FMTSEARCH was then used to direct SAS to first search MYDATA.FORMATS, thus the second PROC FREQ classifies SALES on the mean. ❻ FMTSEARCH was returned to the default.

MPRINT

The general form of MPRINT is, MPRINT | NOMPRINT with a default of NOMPRINT. When syntax is generated by a macro program, setting the option to MPRINT allows the syntax to be viewed and debugged in the LOG. For example, the following macro program generates a PROC PRINT with a WHERE statement if the parameter to the macro is not missing, otherwise it prints all observations of SASHELP.CLASS:

```
%macro printclass(name);
  title1 "Print SASHELP.CLASS";
  proc print data=sashelp.class;
    %if &name ne %then %do;
    where name="&name";
    title2 "Name=&name";
     %end;
  run;
%mend printclass;
Options mprint; %printclass(Janet); options nomprint;
```

Submitting the syntax generates the LOG shown in Figure 6. Each line of syntax generated by the macro program is shown in the LOG where the lines begin with an identification of the macro program “MPRINT(PRINTCLASS).”

Figure 6 Option MPRINT Showing Syntax Generated by the Macro Program

```
97   options mprint;
98   %macro printclass(name);
99     title1 "Print SASHELP.CLASS";
100    proc print data=sashelp.class;
101      %if &name ne %then %do;
102      where name="&name";
103      title2 "Name=&name";
104       %end;
105    run;
106  %mend printclass;
107  %printclass(Janet);
MPRINT(PRINTCLASS):   title1 "Print SASHELP.CLASS";
MPRINT(PRINTCLASS):   proc print data=sashelp.class;
MPRINT(PRINTCLASS):   where name="Janet";
MPRINT(PRINTCLASS):   title2 "Name=Janet";
MPRINT(PRINTCLASS):   run;

NOTE: There were 1 observations read from the data set SASHELP.CLASS.
      WHERE name='Janet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

STORING MPRINT OUTPUT IN AN EXTERNAL FILE

MPRINT used in combination with MFILE allows the syntax generated by a macro program to be directed to an external file. This is particularly helpful for testing and debugging syntax that is generated by a complicated macro program. In order to take advantage of this capability, set MPRINT and MFILE in an OPTIONS statement and then use MPRINT as the FILENAME that references a text file. This example calls the macro %PRINTCLASS featured in the previous example and writes the results produced by MPRINT to a file named TEMPFILE.TXT (Figure 7). The last two lines of syntax removes the MPRINT file reference established in the second line and resets MFILE to NOMFILE.

```
options mprint mfile ls=70;
filename mprint '.\tempfile.txt';
%printclass(name=Janet);
filename mprint clear;
options nomfile;
```

Figure 7 Example LOG of MPRINT Created through the MFILE Option

```
109  options ls=70;
110  options mprint mfile;
111  filename mprint '.\tempfile.txt';
112  %printclass(name=Janet);
MPRINT(PRINTCLASS):   title1 "Print SASHELP.CLASS";
NOTE: The macro generated output from MPRINT will also be written to
      external file C:\Documents and Settings\pthornton\tempfile.txt
      while OPTIONS MPRINT and MFILE are set.
MPRINT(PRINTCLASS):   proc print data=sashelp.class;
MPRINT(PRINTCLASS):   where name="Janet";
MPRINT(PRINTCLASS):   title2 "Name=Janet";
MPRINT(PRINTCLASS):   run;

NOTE: There were 1 observations read from the data set SASHELP.CLASS.
      WHERE name='Janet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.09 seconds
      cpu time            0.00 seconds


113  filename mprint clear;
NOTE: Fileref MPRINT has been deassigned.
114  options nomfile;
```

The file that was created contains the following syntax generated by the macro program.

```
proc print data=sashelp.class;
where name="Janet";
title1 "Report for Janet";
run;
```

MLOGIC

The general form of MLOGIC is MLOGIC | NOMLOGIC with the default being NOMLOGIC. The macro processor traces its execution and writes the trace information to the SAS log when the MLOGIC option is set (SAS Institute Inc., 2005). This option is useful for debugging a macro program. For example, Figure 8 shows the LOG resulting from a call to %PRINTCLASS with the MPRINT and MLOGIC option. The lines created by MLOGIC start with "MLOGIC(PRINTCLASS)" and show the beginning of the execution, the value of the parameter NAME, the result of the conditional %IF statement, and the end of execution.

```
options mprint mlogic;
%printclass(name=Janet);
options nomprint nomlogic;
```

Figure 8 Example of MLOGIC in the LOG

```
118  options mlogic;
119  %printclass(name=Janet);
MLOGIC(PRINTCLASS):  Beginning execution.
MLOGIC(PRINTCLASS):  Parameter NAME has value Janet
MPRINT(PRINTCLASS):   title1 "Print SASHELP.CLASS";
MPRINT(PRINTCLASS):   proc print data=sashelp.class;
MLOGIC(PRINTCLASS):  %IF condition &name ne is TRUE
MPRINT(PRINTCLASS):   where name="Janet";
MPRINT(PRINTCLASS):   title2 "Name=Janet";
MPRINT(PRINTCLASS):   run;

NOTE: There were 1 observations read from the data set SASHELP.CLASS.
      WHERE name='Janet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds


MLOGIC(PRINTCLASS):  Ending execution.
120  options nomlogic;
```

It is recommended that MLOGIC system option is used when there are doubts with respect to the logic of the program. In general, MLOGIC tends to print many lines in the log, so it would be advisable to use it only when debugging the program and not necessarily when the macro is being called within another program.

Another example of the MLOGIC system option is presented in the example below (Figure 8).

```
%macro calc;
     %let numerator=2;
     %let denominator=4;
     %let calc=%eval(&numerator/&denominator)*100;
 %mend;
 options mlogic;
 %calc;
 options nomlogic;
```

MLOGIC traces the flow of the programming logic. In this case, the notes in the log indicates the macro variables set by the let statements.

Figure 8 Example of Log Using the Option MLOGIC

```
337  %macro calc;
338       %let numerator=2;
339       %let denominator=4;
340       %let calc=%eval(&numerator/&denominator)*100;
341  %mend;
342  options mlogic;
343  %calc;
MLOGIC(CALC):  Beginning execution.
MLOGIC(CALC):  %LET (variable name is NUMERATOR)
MLOGIC(CALC):  %LET (variable name is DENOMINATOR)
MLOGIC(CALC):  %LET (variable name is CALC)
MLOGIC(CALC):  Ending execution.
```

SYMBOLGEN

The general form of the option SYMBOLGEN is SYMBOLGEN | NOSYMBOLGEN where the default is NOSYMBOLGEN. “The SYMBOLGEN system option tells you what each macro variable resolves to by writing messages to the SAS log. This option is especially useful in spotting quoting problems, where the macro variable resolves to something other than what you intended because of a special character.” (SAS Institute Inc., 2005) Calling the %PRINTCLASS macro with options SYMBOLGEN and MPRINT creates the LOG shown in Figure 9.

```
options mprint symbolgen;
%printclass(name=Janet);
options nomprint nosymbolgen;
```

The lines generated by the option are identified by “SYMBOLGEN:” and state the resolved value of the macro variable NAME at each spot where it is resolved in the macro program: (a) the %IF statement, (b) the WHERE statement, and (c) the TITLE statement.

Figure 9 Example LOG generated using the SYMBOLGEN Option

```
306  options mprint symbolgen;
307  %printclass(name=Janet);
MPRINT(PRINTCLASS):   title1 "Print SASHELP.CLASS";
MPRINT(PRINTCLASS):   proc print data=sashelp.class;
SYMBOLGEN:  Macro variable NAME resolves to Janet
SYMBOLGEN:  Macro variable NAME resolves to Janet
MPRINT(PRINTCLASS):   where name="Janet";
SYMBOLGEN:  Macro variable NAME resolves to Janet
MPRINT(PRINTCLASS):   title2 "Name=Janet";
MPRINT(PRINTCLASS):   run;

NOTE: No observations in data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


308  options nomprint nosymbolgen;
```

MCOMPILE, MCOMPILENOTE

The general form of MCOMPILE is MCOMPILE | NOMCOMPILE where the default is MCOMPILE. The possible values of MCOMPILENOTE are NONE | AUTOCALL | ALL (SAS Institute Inc., 2005). As reported in the SAS® 9.2 Macro Language: Reference (SAS Institute Inc. 2009):

- MCOMPILE allows new macro definitions, while NOMCOMPILE disallows new macro definitions
- MCOMPILENOTE confirms that the compilation of a macro was completed. In the log, a note is describing the size and number of instructions of the macro.

MCOMPILENOTE system option can be useful while debugging a macro program, to make sure the program compiled without errors. MCOMPILENOTE has three additional options, such as NONE (no note will be printed in the log), AUTOCALL (note will be printed in the log), and ALL (upon the compilation of the macro, a note will be printed in the log). The following syntax generated the log with the note shown in Figure 10.

```
options mcompilenote=all;
%macro test;
    %put "WUSS Conference";
%mend test;
```

Figure 10 Example LOG generated using the MCOMPILENOTE Option

```
315  options mcompilenote=all;
316  %macro test;
317      %put "WUSS Conference";
318  %mend test;
NOTE: The macro TEST completed compilation without errors.
      3 instructions 40 bytes.
```

MACRO DEBUGGING PROCESS

Based on previous experience working with system options, there are a couple of things to consider when working with and debugging macros. The first step will be to make sure the macro produces what it is suppose to produce. To check for this, we can use the MPRINT option. The second step is to check for the way the macro variables are being created by using SYMBOLGEN. In the last step, use MLOGIC to check the logic of the program. MLOGIC should be a tool for precise debugging. If all these options are used at once though, the log will be full and hard to 'see' where the problem really is.

CONCLUSION

This paper was a sampling of our best and/or most interesting SAS system options. System options often allow for easy and sweeping change across the SAS session, and there are 100s out there to explore. Lin (2009), Sun & Carpenter (2011), and Russell & Tyndall (2011) offer excellent reviews of system options and/or macro related options in particular. When writing macro, it is always a good idea to make use of SAS system options such as: MLOGIC, MPRINT, and MYSMBOLGEN. Other useful system options are MCOMPILE, MCOMPILENOTE. These particular options will help you debug a macro program and point out the issues you might have.

RECOMMENDED READING

Lin, W., & Kang, J. (2009). Don't Despair, You do have an option!. Proceedings of the Northeast SAS User Group, Burlington, VT.

Russell, K. & Tyndall, R. (2010). SAS System Options: The True Heroes of Macro Debugging, Proceedings of the Annual SAS Global Form, Seattle, WA

Sun, E., & Carpenter, A.L. (2011). Protecting Macros and Macro Variables: It is all about control, Proceedings of the Annual Pharmaceutical Industry SAS® Users Group Conference, Nashville, TN

SAS Institute Inc. 2009. SAS® 9.2 Macro Language: Reference. Cary, NC: SAS Institute Inc. http://support.sas.com/documentation/cdl/en/mcrolref/61885/PDF/default/mcrolref.pdf

REFERENCES

Delaney, K.P. & Carpenter, A.L. (2004). SAS Macro: Symbols of Frustration? %Let us help! A guide to debugging Macros. Proceedings of the Twenty Ninth Annual SAS® Users Group International Conference, Montréal, Québec, Canada

Heaton, E. (2003). SAS® Systems Options are your friends. Proceedings of the Twenty Eighth Annual SAS® Users Group International Conference, Seattle, WA

Lund, P. (2003). Keep those formats rolling: A macro to manage the FMTSEARCH= Option. Proceedings of the Twenty Eighth Annual SAS® Users Group International Conference, Seattle, WA

Poll, D. (2011). SAS® Options - Versatile Players in the Game of SAS, Proceedings of the Annual SAS Global Form, Las Vegas, NV

SAS Institute (2011). SAS 9.2 Documentation, SAS(R) 9.2 Macro Language: Reference:

http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a001066200.htm

SAS Institute (2011). SAS 9.2 Documentation, Base SAS(R) 9.2 Procedures Guide, PROC OPTIONS statement:

http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000083867.htm

SAS Institute Inc. 2005. SAS OnlineTutor®: Advanced SAS®. Cary, NC: SAS Institute Inc. http://web.utk.edu/sas/OnlineTutor/1.2/en/60477/m52/m52_11.htm

SAS Institute Inc., (1999) SAS OnlineDoc®, Version 8, Cary, NC: SAS Institute Inc. http://www.cc.kyushu-u.ac.jp/scp/system/manual/sashtml/macro/z1066200.htm

Thornton, S. P. (2010). Essential SAS® ODS PDF. Proceedings of the Eighteenth Annual Western Users of the SAS® Software Conference, San Diego, CA.

Thornton, P. (2011). SAS® DICTIONARY: Step by Step. Proceedings of the Annual SAS Global Form, Las Vegas, NV

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Patrick Thornton, Ph.D.
Principal Scientific Programmer/Analyst, SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
650 859-5583
patrick.thornton@sri.com

Iuliana Barbalau, MS
SAS Programmer, Adecco
4300 Hacienda Drive
Pleasanton, CA 94588
925 730-8518
iuliana.barbalau@contractors.roche.com