Best Practices: Clean House to Avoid Hangovers

Mary F. O. Rosenbloom, Edwards Lifesciences LLC, Irvine, CA
Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

ABSTRACT

In a production environment, where dozens of programs are run in sequence, often monthly or quarterly, and where logs can span thousands of lines, it's easy to overlook the small stuff. Maybe a data statement fails to execute, but one already exists in the temp library from a previous program. Maybe a global macro assignment is missed or fails to execute, but a global macro of the same name already exists from a previous program. This can also happen with macros. The list goes on. This paper offers some suggestions for housekeeping steps that can be taken at the end of each SAS program to minimize the chance of a hangover.

INTRODUCTION

It's time again to run that quarterly report! Oh joy, there are 30 programs to run, and the last time that you looked at them was last quarter. Maybe you did yourself a favor and wrote a program that uses %include to call them all in order. You check your folders, clear out any unwanted files, reboot SAS, and hit RUN. Good news, the log is clean and the output looks good. But are you sure that you didn't miss something?

There are a multitude of 'little things' that can sneak in and cause trouble. Temporary datasets can find their way into subsequent programs, so can macros programs, and global macros. Or, what if you ran all 30 of your programs, reviewed the log, and then forgot to save it! Maybe you ran your programs and just overwrote the output from last quarter! Or, what if you send the wrong output to your customer? Head starting to hurt? This paper offers several suggestions for housekeeping and file keeping that may help prevent these types of SAS hangovers. So drink up, and read on.

CLEANING HOUSE WITH "GOOD" PROGRAMMING STYLE

Most professional SAS users know that adhering to a "good", and accepted, programming style is an important and necessary step for the success of a program. Style contributors include using meaningful variable and routine names, good program structure, straightforward and easily understandable coding techniques, clear code layout (or visual structure), and a minimization of control-flow. Good programming style not only makes a program more readable and serviceable, but serves as a critical form of documentation known as code-level documentation. Unlike many types of external documentation (e.g., program specifications and user documentation) code-level documentation is self-documenting and typically is the type of documentation to remain correct as the code is modified.

USING PROC DATASETS TO CLEAR TEMP DATASETS AT THE END OF A PROGRAM

Let's face it, there are only so many names that one can assign to a temporary dataset. When the same person writes 20 or 30 SAS programs, chances are that some of the programs will generate temporary datasets with the same names. In the event that an error occurs and one of these temporary datasets is not generated, if the previous temporary datasets were not cleared, one of them may be substituted. This occurrence might not always be obvious by looking at the SAS log, either. So, why not, at a minimum, place a PROC DATASETS call at the end of every program?

```
***delete temporary datasets;
proc datasets lib=work ❶ memtype=data nolist;❷
       delete _PVALUES ⓿;
quit;
```

In this example, the dataset _PVALUES ⓿ is deleted from the WORK library ❶ at the end of the program. The NOLIST option is specified. ❷ This is similar to the NOPRINT option available for many procedures, and ensures that nothing is sent to the OUTPUT window. Alternatively, one can place the underscore '_' as the first character in the name of every temporary dataset, and then use PROC DATASETS to delete all datasets beginning with an underscore.

```
***delete temporary datasets;
proc datasets lib=work memtype=data nolist;
       delete _: ❸;
quit;
```

The colon ':' ❸ is used as a wildcard here and when paired with the underscore it represents any and all datasets beginning with an underscore.

USING PROC CATALOG TO DELETE UNWANTED MACRO PROGRAMS

Macro programs should be deleted as soon as they are no longer needed. This way, the user ensures that only the desired macro is used. Macros used throughout the entire program suite should be placed into one or more macro-only programs. Every time a new one is written, the programmer should confirm that a macro of the same name does not already exist. For macros written and used exclusively in one SAS program, PROC CATALOG should be used to delete the macro program once it is no longer needed.

```
***delete temp macros from the macro catalog;
proc catalog catalog=work.sasmacr; ❻
delete continuous ❺/ et=macro ❹;
quit;
```

This code deletes the temporary macro ❹ %CONTINUOUS ❺ from the WORK.SASMACR ❻ library.

USING %SYMDEL TO DELETE UNNEEDED GLOBAL MACRO VARIABLES

Global macro variables can be just as unwieldy and dangerous as stray temporary datasets and macro programs. Consider the example where we create the macro variable &TESTS, the number of statistical tests run for an analysis. We will use this number to create adjusted p-values.

```
***store the number of statistical tests in a macro variable;
%let tests=40;
```

To see the names and values of all of the existing macro variables that you have generated in your current SAS session, issue the following code:

```
***get a list of global macro variables that exist in this SAS session;
%put _user_;
```

The log shows that currently, the macro variable &TESTS exists and has a value of 40 (stored as a character string).

```
26   ***get a list of global macro variables that exist in this SAS session;
27   %put _user_ ;
GLOBAL TESTS 40
```

We want to delete this macro variable as soon as we are done using it, so that it is not accidentally used later. To do this, we simply use %SYMDEL;

```
***delete temporary global macro variables;
%symdel tests;
```

And now we can see that the macro variable has been deleted from the global symbol table since nothing is listed in the log after line 31.

```
30   ***get a list of global macro variables that exist in this SAS session;
31   %put _user_ ;
```

USING DM STATEMENTS TO SAVE THE LOG

We are often required to save the SAS log each time analysis is run. It is easy to forget to do this; but, when DM statements are built into the program, this is done automatically.

```
***Save the log;
DM 'log; file "C:\prgs\prod\log &sysdate9..log" replace'; ❼
***Clear the log window;
DM "log; clear; "; ❽
```

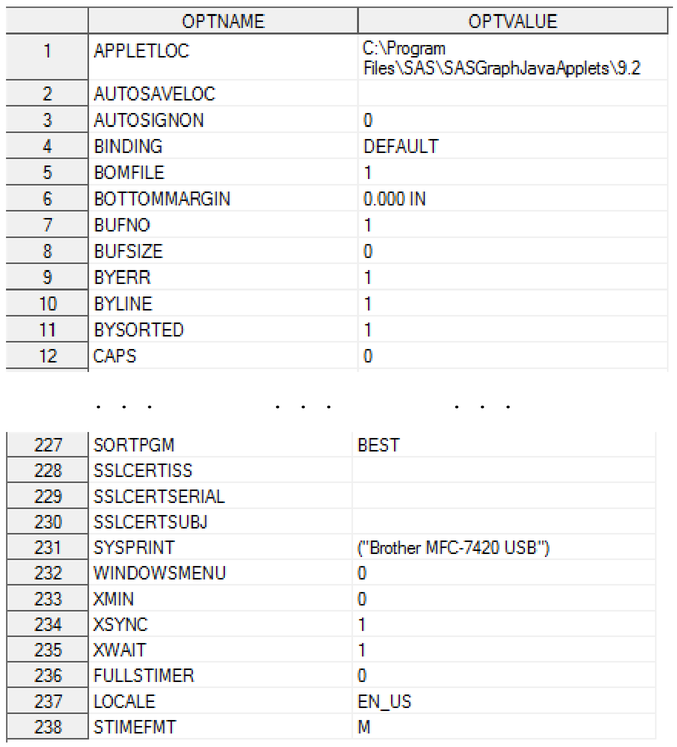
This code issues a command to save the log ❼ with the name “log &sysdate9.log” (where &sysdate9 is a global system variable containing the system date). Then the log window is cleared. ❽ This can be done multiple times throughout a programming suite, if needed. Remember to give each log a unique name. In this example the system date is used to name the log. Alternatively, the data extract date could be used.

SAVING AND RESTORING STARTUP (INITIALIZED) SYSTEM OPTIONS

Before starting a process, determine whether the values assigned to the startup SAS system options need to be preserved (or saved) for later reinitializing. If option settings do need to be saved, then you can run PROC OPTSAVE, a specially-designed procedure to save current SAS system option settings, before processing and option changes occur. In the next example, we illustrate the process of saving the startup SAS system options to a SAS dataset using the OPTSAVE procedure.

```
***Save the startup SAS System Options with PROC OPTSAVE;
proc optsave out=sasuser.startup_system_options; ❾
run;
```

The OPTSAVE procedure code ❾ saves the startup SAS system options to the user-assigned dataset startup_system_options in the SASUSER library. If the output dataset already exists with the same name, then it is automatically replaced. A partial snapshot of the saved options appears below.



	OPTNAME	OPTVALUE
1	APPLETLOC	C:\Program Files\SAS\SASGraphJavaApplets\9.2
2	AUTOSAVELOC	
3	AUTOSIGNON	0
4	BINDING	DEFAULT
5	BOMFILE	1
6	BOTTOMMARGIN	0.000 IN
7	BUFNO	1
8	BUFSIZE	0
9	BYERR	1
10	BYLINE	1
11	BYSORTED	1
12	CAPS	0
	. . .	. . .
227	SORTPGM	BEST
228	SSLCERTISS	
229	SSLCERTSERIAL	
230	SSLCERTSUBJ	
231	SYSPRINT	("Brother MFC-7420 USB")
232	WINDOWSMENU	0
233	XMIN	0
234	XSYNC	1
235	XWAIT	1
236	FULLSTIMER	0
237	LOCALE	EN_US
238	STIMEFMT	M

SAS provides another method of saving the current SAS system option settings. In the next example the DMOPTSAVE Display Manager command is specified (from any command line in the SAS windowing environment).

```
***Save the Startup SAS System Option Settings with DMOPTSAVE;
dmoptsave sasuser.startup_system_options; ❿
```

The DMOPTSAVE command ❿ saves the current SAS system option settings to the user-assigned SAS dataset startup_system_options in the SASUSER library. As with the OPTSAVE procedure, the output dataset is automatically replaced if it already exists. **Note:** The OUT= keyword is not specified with the DMOPTSAVE command as it is with the OPTSAVE procedure.

After all desired processing is completed, the SAS system option settings can be restored (loaded) back to their initial startup values from the “saved” option settings using the OPTLOAD procedure, as follows.

```
***Restore the Startup SAS System Option Settings with PROC OPTLOAD;
proc optload data=sasuser.startup_system_options; ⓫
run;
```

The OPTLOAD procedure ⓫ restores the “saved” SAS system option settings from the user-assigned SAS dataset startup_system_options in the SASUSER library. When run, the OPTLOAD procedure automatically replaces the current option settings with the “saved” settings created with the OPTSAVE procedure or DMOPTSAVE command.

The next example shows code containing a number of proc steps and options statements to illustrate the entire process of first saving the initialized SAS system option values, then the modification of the MSGLEVEL= option value, and finally restoring (or loading) the options back to their default values.

```
***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;  ⓬
run;

***Save the Startup SAS System Options with PROC OPTSAVE;
proc optsave out=sasuser.myoptions;  ⓭
run;

***Change the Value of the MSGLEVEL= SAS System Option;
options msglevel=I;  ⓮

***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;  ⓯
run;

***Load the Saved Startup SAS System Options with PROC OPTLOAD;
proc optload data=sasuser.myoptions;  ⓰
run;

***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;  ⓱
run;
```

The resulting SAS log after executing the previous code appears below.

```
***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;
run;

MSGLEVEL=N        Level of messages displayed  ⓬
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0:00.00
      cpu time            0:00.01

(Continued on next page)
```

```
(Continued from previous page)

***Save the Startup SAS System Options with PROC OPTSAVE;
proc optsave out=sasuser.myoptions;
run;

NOTE: The data set SASUSER.MYOPTIONS has 238 observations and 2 variables. ⓭
NOTE: PROCEDURE OPTSAVE used (Total process time):
      real time           0:00.01
      cpu time            0:00.00

***Change the Value of the MSGLEVEL= SAS System Option;
options msglevel=I;   ⓮

***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;
run;

MSGLEVEL=I        Level of messages displayed   ⓯

***Load the Saved Startup SAS System Options with PROC OPTLOAD;
proc optload data=sasuser.myoptions; ⓰
run;

NOTE: PROCEDURE OPTLOAD used (Total process time):
      real time           0:00.40
      cpu time            0:00.23

***Display the Value of the MSGLEVEL= SAS System Option;
proc options option=msglevel;
run;

MSGLEVEL=N        Level of messages displayed   ⓱
```

USING X COMMANDS TO ZIP YOUR OUTPUT

How many times have you accidentally overwritten old output, either because you forgot to update the output folder location, or perhaps you batch submitted the wrong program? One way to reduce the chance of making this mistake is to place data and output into .zip folders.

```
options noxwait noxsync mprint ;  ⓲
***zip data;
x " ""D:\Program Files\WinZip\WINZIP32.EXE"" -a
       C:\data\data &sysdate9..zip""
       ""C:\data\*.sas7bdat"" "; ⓳
```

We start by setting some options to ensure that dialogue windows are automatically closed, and don't cause the SAS session to pause ⓲. In this example ⓳, WinZip is executed, and all SAS datasets with the .sas7bdat extension are zipped into the folder “data &sysdate9.zip”. This can also be done for output files such as Excel documents, and even for SAS programs.

EMAILING THE RESULTS USING SAS

Using SAS to issue an email with the results attached is another way to avoid making an error.

```
***email the zipped listings;
filename mymail email to="joe_smith@mycompany.com"
                 subject="Study Ouput"
            content_type="text/plain"
                  attach=("C:\Ouput\listings &sysdate9..zip");
data _null_;
   file mymail;
   put 'Joe, ';
   put /;
   put 'The listings for The Study are attached.';
   put /;
   put 'Enjoy!';
run;

filename mymail clear;
quit;
```

Here, we take advantage of the &SYSDATE9 global macro variable, and send the file that we just created. Using SAS to send an email in this case is not only efficient, but can also prevent costly or embarrassing mistakes.

SUMMARY

In the fast-paced environments that we work in, there are a multitude of things to remember, pitfalls to avoid, and priorities change quickly. By implementing some simple housekeeping techniques we can avoid hangovers, slip-ups and other programming headaches that can happen when you're out all night – at the office.

ACKNOWLEDGMENTS

The authors would like to thank Kathy Valdes, WUSS 2011 Academic Program Chair and Ginger Carey, Operations Chair for accepting our abstract and paper, as well as for a great conference! Mary Rosenbloom would like to thank Kirk Paul Lafler for his mentoring and enthusiasm.

REFERENCES

PROC CATALOG
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002473343.htm

PROC DATASETS
Raithel, Michael A. 2010. "PROC DATASETS; The Swiss Army Knife of SAS® Procedures" SAS institute Inc. Proceedings of the SAS® Global Forum 2010 Conference. Cary, NC: SAS Institute Inc.

DM STATEMENTS
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000167815.htm
http://studysas.blogspot.com/2009/11/sas-display-manager-commands.html
http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a002120744.htm

EMAIL
Hunley, Chuck. 2010. "SMTP E-Mail Access Method: Hints, Tips, and Tricks" SAS Institute Inc. Proceedings of SAS Global Forum 2010 Conference. Cary, NC: SAS Institute Inc.
Whitworth, Ryan. 2010. "Zip and Email Files Using SAS® To Reduce Error and Make Documentation Easy". Proceedings of SAS Global Forum 2010 Conference.

PROC OPTLOAD
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002214501.htm
http://support.sas.com/documentation/cdl/en/proc/63079/HTML/default/viewer.htm#p1ull1aq9699m1n1eghata0jmjcq.htm

PROC OPTSAVE
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002214507.htm
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002214505.htm

%SYMDEL
diTommaso, Dante. 2003. "Taking Control of Macro Variables" SAS institute Inc. Proceedings of SUGI 28 Conference. Cary, NC: SAS Institute Inc.
http://support.sas.com/kb/26/154.html

X COMMANDS
http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#exittemp.htm
Li, Na. 2005. "Applications for Running DOS Commands within SAS" SAS institute Inc.Proceedings of PharmaSUG 2005 Conference. Cary, NC: SAS Institute Inc.

USES OF THE COLON
Luo, Haiping. 2001. "That Mysterious Colon (:)" SAS institute Inc. Proceedings of SUGI 26 Conference. Cary, NC: SAS Institute Inc.

TRADEMARK CITATIONS

ABOUT THE AUTHORS

Mary Rosenbloom is a statistical programmer at Edwards Lifesciences in Irvine, California. She has been using SAS for over 15 years, and is especially interested in using macros to generate data-driven code, DDE, and program validation methods.

Kirk Paul Lafler is consultant and founder of Software Intelligence Corporation and has been using SAS since 1979. He is a SAS Certified Professional, provider of IT consulting services, trainer to SAS users around the world, and sasCommunity.org Advisory Board member. As the author of four books including PROC SQL: Beyond the Basics Using SAS, Kirk has written more than four hundred peer-reviewed papers, been an Invited speaker and trainer at more than three hundred SAS International, regional, local, and special-interest user group conferences and meetings, and is the recipient of 17 "Best" contributed paper awards. His popular SAS Tips column, "Kirk's Korner of Quick and Simple Tips", appears regularly in several SAS User Group newsletters and websites, and his fun-filled SASword Puzzles is featured in SAScommunity.org.

Comments and suggestions can be sent to:

Mary F. O. Rosenbloom
Edwards Lifesciences, LLC
E-mail: Mary_Rosenbloom@Edwards.com
~~~
Kirk Paul Lafler
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
~~~