Preventing the Cascading Failure
A Primer to Debug Errors within SAS Programs

Scott D. Brown, University of Central Florida, FL

1. INTRODUCTION

You've been programming for hours, looking at numerous function tutorials to make sure the SAS code for a particular project will compile and run as it should. After doubling checking everything, you compile and run only to find the SAS log screaming gibberish about a coding error. You then realize nothing was compiled. WHAT?! While this may seem like an all too familiar scenario in any programing language, it's important to remember that this isn't a one hit kill for a program. Thanks to all the support that exists for SAS, there's always hope to get a programming up and running.

2. What Should I Do?!?

Programming errors have been around since the dawn of computer programming and machine language. They can and ultimately will happen at a rate which is more frequent than preferred. You may notice different kinds of issues within a program. Did it run properly? If so, is it giving you the correct result? Did it not even produce any kind of data set upon compiling?

Just as there is an infinite amount of ways to code a program, there is also a seemingly infinite way to fix them. This paper will discuss 2 types of errors within a SAS program:

Simple Errors
- Syntax Errors
- Order/Logical Errors
- Understanding the SAS Log

Macro Errors
- Proper Macro Terminology
- Macro Options

2.1 Word of Note: Debugging Can and Will Make You Rage at Times

It's important to distinguish just what you're trying to accomplish when it comes to debugging code. Finding the root of the problem is the key. A cascading failure can still occur if the root of a problem is not fixed or something else is fixed in its place. Think of debugging as a process. The proper order of steps must be taken to determine the proper solution. Though tedious, it will save time and help prevent the enviable rage that comes with a failed program. It's important for code to be correct rather than fortuitous. You want to prevent systemic failure, not encourage it.

3. The Dilemma of Simple Errors

Everyone has heard the term “syntax error” before but what exactly does it mean? To quote Dictionary.com, syntax is “that branch of modern logic that studies the various kinds of signs that occur in a system and the possible arrangements of those signs, complete abstraction being made of the meaning of the signs.” With this in mind, simple errors are often due to improper arrangement of “signs” within SAS code which obstructs the proper meaning of them.

When it comes to debugging, SAS can be your best friend. Why is that? Just look at the log! For many simple errors, looking at the log may be all the debugging you need to do. The SAS log is very informative when something goes wrong (although not very informative when something does do right). It’s important to understand the comments the SAS log can show you.

3.1 ERRORS

Errors are very keen in SAS. Their red text certainly helps them stand out over the dull black text of the log. Errors happen when something was wrong within the code and it could not be executed. Probably the most common cause of getting an error in the log is punctuation errors such as a the lapse of a semicolon. Almost every line in SAS ends in one so forgetting one will prompt something like this:

```
proc print data = all
run;
ERROR 202-322: The option or parameter is not recognized and will be ignored.
```

Here, SAS is told “run” is an option of proc print. The error is printed in the log since SAS cannot properly distinguish it. However, a semi colon may also be seemingly out of place. To make things look a bit cleaner, code can be broken up ove multiple lines. Here, a proc import is used for the 'console' data set:

```
PROC IMPORT out= work.console datafile = "H:\Console Data.xls" dbms = excel
replace;
```

However, the beginning statement for proc import may be broken up over 3 lines to be easier to understand. However, the breaks may be a bit confusing to anyone unfamiliar with proc import. It may cause the curious programmer to create something like this:

```
PROC IMPORT out= work.console;
datafile = "H:\Console Data.xls";
dbms = excel replace;
```

Because of this, an error is produced due to the semicolons placed at the end of lines 1, 2 and 3 of the procedure which prevents SAS found determining where it should get the file from.

```
ERROR: FILE= or TABLE= is required and must be specified.
```

The Error statement can actually pretty useful. It will often try to show you what exactly is missing or what is not in the proper place.

```
248  proc means data = console med;
                               ---
                               22
                               202
ERROR 22-322: Syntax error, expecting one of the following: ;, (, ALPHA, CHARTYPE, CLASSDATA,
              CLM, COMPLETETYPES, CSS, CV, DATA, DESCEND, DESCENDING, DESCENDTYPES, EXCLNPWGT,
              EXCLNPWGTS, EXCLUSIVE, FW, IDMIN, KURTOSIS, LCLM, MAX, MAXDEC, MEAN, MEDIAN, MIN,
              MISSING, MODE, N, NDEC, NMISS, NOLABELS, NONOBS, NOPRINT, NOTHREADS, NOTRAP, NWAY,
              ORDER, P1, P10, P20, P25, P30, P40, P5, P50, P60, P70, P75, P80, P90, P95, P99,
              PCTLDEF, PRINT, PRINTALL, PRINTALLTYPES, PRINTIDS, PRINTIDVARS, PROBT, Q1, Q3,
              QMARKERS, QMETHOD, QNTLDEF, QRANGE, RANGE, SKEWNESS, STACKODS, STACKODSOUTPUT,
              STDDEV, STDERR, SUM, SUMSIZE, SUMWGT, T, THREADS, UCLM, USS, VAR, VARDEF.

ERROR 202-322: The option or parameter is not recognized and will be ignored.
249  by company;
250  run;
```

With this code, median is mislabeled as 'med'. Not only did SAS not recognize this, it gives a complete list of what it is was expecting. This can be used as a creative way to see what options are available for certain procedures. Whoever said that succeeding through failure never worked!

3.2 WARNINGS

Warnings are much threatening than their error counterparts but they may still show that something did not work properly. Trying to create a regression using the proc import example yields this curious message:

```
proc reg data=console plots(only) = fitplot;
model Total_Sales = Starting_Price Revisions Life_Span Start End Generation;

ERROR: Variable Start in list does not match type prescribed for this list.
ERROR: Variable End in list does not match type prescribed for this list.
ERROR: Variable Generation in list does not match type prescribed for this list.

NOTE: The previous statement has been deleted.
output out=conreg;
run;

WARNING: No variables specified for a SSCP matrix.  Execution Terminating.
```

Since the last 3 variables were character (and somehow unspecified), SAS deletes the entire model statement and thus, produces the warning stating that there are no variables for proc reg and it won't be executed. However, the output of "Conreg" is still created however it has 0 variables and is meaningless to use. Warnings won't really show what the problem is (unlike errors) but it will explain that there was nothing to execute and an empty data set will be created.

3.3 NOTES

Notes are the most common nonblack line that you will see in the log. They simply show the progress of what was compiled and executed. Running the proc import for the 'console' data set yields these notes:

```
NOTE: WORK.CONSOLE data set was successfully created.
NOTE: The data set WORK.CONSOLE has 22 observations and 9 variables.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.55 seconds
      cpu time            0.20 seconds
```

The notes are good to use to make sure all observation and variables are read in correctly during the initial data set creation. The console data set did indeed have 22 observations and 8 variables within in confirming that the data set was accurately established.

4. The Problems with Macros

Knowing the SAS log will help alleviate time that may be spent to fix a program that is not properly working. However, sometimes the log may not have the answer you were looking for. This is the case with macros . Macros often compile without any problems (or any errors, warnings or notes in the log) and only display complications after the macro is invoked. Here, a simple macro for proc print compiles without a problem despite the missing semicolon after ods html.

```
251  %macro prt(r);
252  ods html close; ods html
253  proc print data = &r noobs;
254  run;
255  %mend;
```

4.1 The Simple Way to Make a Macro

Macros are a great way to do set tasks repeatedly over different variables. However, SAS executes them differently through a “macro processor” which is why there's usually no errors after the initial compilation of a macro. However, a macro is nothing more than specific code. Thinking of a macro as just “specific code” will make it easier to think of how a macro should be programmed.

Here a macro is needed to do the proc means procedure on the data set 'consoles' but the we wish to do proc means on many aspects of the data. It's best to first start with the basics of what is needed for this procedure:

```
proc sort data=consoles;
by company;
run;
proc means data=consoles;
by company;
run;
```

Proc sort is needed to sort the data by the specified variable. In this example, the variable "company" is chosen but what if this option was left to the user? Also, what if specific statistical options are to be specified? All of this has to be taken into account to establish the macro:

```
%macro mean_proc(data1=,sortvar=,stats1=,stats2=,stats3=);
ods html close; ods html;
proc sort data = &data1;
by &sortvar;
run;
proc means data = &data1 &stats1 &stats2 &stats3;
by &sortvar;
run;
%mend mean_proc;
```

The macro "mean_proc" has been created, allowing the user to pick a data set, a variable to use and 3 meaningful statistical procedures for proc means. It's important to keep all the macro variable names consistent particularly if there are multiple variables within the macro. To invoke the macro (and see if it works correctly):

```
%mean_proc(data1=console,sortvar=company,stats1=n,stats2=median,stats3=mean);

%mean_proc(data1=console,sortvar=company,stats1=t,stats2=median,stats3=mean);
```

The first macro takes in the data set "console", sorts the data set by "company" and will then utilize proc means with the n, median, and mean statistical options. The second macro follows the same process but will complete proc means displaying the t statistic instead of the number of observations.

4.2 Quotation Marks Are Important

When using macro variables in either a title or with the let statement, it's important to specify quotations for SAS. Suppose the only relevant data from the 'console' data set was the "Playstation" data but we want a way to change this to look at other systems. This can be done using the let statement:

```
%let sys = Playstation;
data system1;
set sega sony nintendo microsoft atari;
if system = "&sys";
title "Data for the &sys";
run;
```

With the let statement, we can change the system that is outputted by changing the let statement. Also, the double quotes for "&sys" and the title statement are important to note. Without them, an empty data set would be produced because the macro processor only looks for macro variables that are within double quotes.

5 Macro Options

When all else seems to fail with a macro, you must persist! SAS has 5 options that will ofter further assistance about what is going wrong with a macro.

5.1 The Triggered Options

The first 2 options won't really tell you anything that is unknown. They're purpose is to show that something is either label incorrectly or a macro may have been invoked before it was compiled correctly.

5.2 Merror and Serror

Both of these options will be triggered when something is missing for a macro. Merror will trigger when a macro is invoked before it has been compiled:

```
%cont(all);
WARNING: Apparent invocation of macro CONT not resolved.
```

The macro 'cont' (a macro used to show the contents of a data set) was invoked before it was compiled. Serror is the other triggered error which will flag a symbolic error in macro code:

```
%macro cont(c);
proc contents &data = &c;
run;
%mend;
WARNING: Apparent symbolic reference DATA not resolved
```

Again, the macro 'cont' has 2 ampersand symbols within it, prompting the serror error to warn the user that something is not right within the macro code.

5.3 MLOGIC, MPRINT and SYMBOLGEN

The MLOGIC, MPRINT and SYMBOLGEN macro options will help understand the macro processor when a macro is invoked. Unlike their 2 triggered macro option counterparts, these 3 must be activated with the OPTION statement for them to show up during the macro process:

```
option mprint mlogic symbolgen;
```

These 3 options are very handy for macros and all 3 work together. MLogic will show the start and end of the macro process while specifying the macro parameters. Mprint will show the actual macro code and at what points it's executed. Finally, Symbolgen displays what each set parameter will eventually turn into. Using this macro (named 'combined') as an example to combine 3 separate data sets and print the results:

```
%macro combined (set1=, set2=, set3=);
data &set1&set2&set3;
```

```
set &set1 &set2 &set3;
run;
proc print data = &set1&set2&set3;
run;
%mend combined;
```

The 3 data sets are labeled set1, set2 and set3. While the process of the macro is simple, activating the 3 options will make the entire process even easier to visualize within the log:

```
%combined(set1=atari,set2=sega,set3=nintendo);
MLOGIC(COMBINED):  Beginning execution.
MLOGIC(COMBINED):  Parameter SET1 has value atari
MLOGIC(COMBINED):  Parameter SET2 has value sega
MLOGIC(COMBINED):  Parameter SET3 has value nintendo
```

Mlogic shows the macro 'combined' has been executed with the 3 'set' parameters each set to a value of 'atari', 'sega' and 'nintendo'.

```
SYMBOLGEN:  Macro variable SET1 resolves to atari
SYMBOLGEN:  Macro variable SET2 resolves to sega
SYMBOLGEN:  Macro variable SET3 resolves to nintendo
MPRINT(COMBINED):   data atariseganintendo;
SYMBOLGEN:  Macro variable SET1 resolves to atari
SYMBOLGEN:  Macro variable SET2 resolves to sega
SYMBOLGEN:  Macro variable SET3 resolves to nintendo
MPRINT(COMBINED):   set atari sega nintendo;
MPRINT(COMBINED):   run;
```

Symbolgen then shows what each 'set' parameter resolves for the 'atariseganintendo' data set that is created which is verified with Mprint. Mprint also confirms the 3 sets are then set properly.

```
NOTE: There were 3 observations read from the data set WORK.ATARI.
NOTE: There were 4 observations read from the data set WORK.SEGA.
NOTE: There were 5 observations read from the data set WORK.NINTENDO.
NOTE: The data set WORK.ATARISEGANINTENDO has 12 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.00 seconds
```

```
SYMBOLGEN:  Macro variable SET1 resolves to atari
SYMBOLGEN:  Macro variable SET2 resolves to sega
SYMBOLGEN:  Macro variable SET3 resolves to nintendo
MPRINT(COMBINED):   proc print data = atariseganintendo;
MPRINT(COMBINED):   run;

NOTE: There were 12 observations read from the data set WORK.ATARISEGANINTENDO.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.15 seconds
      cpu time            0.04 seconds


MLOGIC(COMBINED):  Ending execution.
```

The notes show that all 3 data sets are created and Symbolgen shows that all 3 data set variables resolved correctly. Mprint then shows the proc print in the macro and the notes confirm that

'atarisegenintento' will be printed with 12 observations (the correct number). Finally, Mlogic then concludes that the macro execution has been completed.

While these 3 options will help with the macro process and errors, they can further slow down the macro processor when engaged. It's important that when a macro is working correctly after using any of the 3 options to turn them off to improve performance. This can again be done with the option statement:

```
option nomprint nomlogic nosymbolgen;
```

CONCLUSION

This paper discussed debugging techniques with the SAS system. Its focal point was on common SAS errors, how to read and interpret messages within the SAS log window, proper macro construction and macro debugging options. It should provide a good starting point to get a erroneous program working properly.

For more information, contact

Scott Brown
12829 Sharp Shined Street
Orlando, FL, 32837
phone: 407-690-3967 e-mail: scott_brown_jp@yahoo.com

REFERENCES

Susan J. Slaughter, Lora D. Delwiche, (2013). Errors, Warnings, and Notes (Oh My) A Practical Guide to Debugging SAS Programs . *SAS Global Forum 2013*. (127), pp.1-15. Retrieved November 29, 2013, from http://support.sas.com/resources/papers/proceedings13/127-2013.pdf

Andrew Ratcliffe, (2013). Visual Techniques for Problem Solving and Debugging. *SAS Global Forum 2013*. (373), pp.1-10. Retrieved November 30, 2013, http://support.sas.com/resources/papers/proceedings13/373-2013.pdf

Frank C. BiIorio, (2001). The SAS Debugging Primer. *SAS Global Forum 2001*. (68), pp.1-5. Retrieved November 29, 2013, from http://www2.sas.com/proceedings/sugi26/p054-26.pdf

Neil Howard, (2003). Beyond Debugging: Program Validation. *SAS Global Forum 2003*. (28), pp.1-9. Retrieved November 29, 2013, from http://www2.sas.com/proceedings/sugi28/058-28.pdf

"Syntax." *Dictionary.com*. Dictionary.com, n.d. Web. 03Dec.2013. <http://dictionary.reference.com/browse/syntax?s=t>.

TRADEMARK INFORMATION