Abstract

The interaction between SAS macro code and SAS program statements makes SAS macros particularly difficult to debug. If an error occurs in a SAS macro, it could be in the macro code, in the SAS code generated by the macro, or even in SAS code preceding the macro.

This tutorial will present a step-by-step, "cookbook" approach to debugging a SAS macro. Topics to be covered include:

* A very brief review of SAS macro processing ("what happens when")
* First checks to resolve common problems
* Guidelines for determining where the problem lies: in macro code or SAS code
* Useful SAS system options and statements for debugging, with explanations of output generated by options
* Other resources, to learn more about writing SAS code and avoiding errors

This tutorial is intended for anyone who writes (and therefore has to debug) SAS macros.

Introduction

SAS macro programming is not for the faint of heart. The macro programming language is very powerful, but with its power comes complexity. Add the fact that the syntax of the macro language is almost identical to DATA step programming language (the key word here is "almost"), and errors are bound to occur - errors which are difficult to detect.

The purpose of this tutorial is to provide a process to follow to track down errors in macro code. We will follow this outline:

* Brief overview of macro processing: what happens when
* Avoiding problems: suggestions on how to avoid errors
* First checks: some common problems that can occur that you can check quickly
* Problem determination: when it's not obvious, how to first determine where the problem lies, then what to do about it
* Tools for debugging: SAS options and other statements you can use to determine the cause of the error
* Resources: manuals you can refer to and courses you can attend to learn more about the macro language

Brief overview of macro processing

If you are writing macros, you should already know this. But, it never hurts to review. The most important thing to remember about macro programming is that macro code gets executed first, then SAS code. This is important to remember because many errors occur because people forget this and program as if macro code and SAS code execute simultaneously. The diagram gives a simplified picture of how the SAS system runs. Whenever the SAS word scanner encounters a percent sign (%) or ampersand (&), it passes control to the macro processor to see if it can be resolved into SAS statements, or portions of SAS statements. It can help to think of the macro processor as a SAS code generator.

Avoiding Problems (or, An Ounce of Structured Programming Is Worth a Pound of Debugging)

Like many wise people have undoubtedly told you throughout your life, the best way to get out of trouble is not to get into it in the first place (or something like that). This is also true with macro programming. Here are a couple of simple ways to help yourself avoid trouble:

* Use structured programming. If you can minimize the number of flows into our out of a block of code, you can reduce the number of places an error can occur. You can also minimize the number of paths to trace if an error does occur.

* Document thoroughly (At least use lots of comments). As we have said, the macro language can be confusing to write in. It can be nearly impossible to figure out what somebody else is trying to do, or what you were trying to do six months ago. Documenting your intentions should reduce the number of bugs introduced in maintenance as a result of not knowing or remembering what's going on.
• Use visual cues. Because the syntax of macro code is so similar to SAS data step code, it can become difficult to distinguish between the two in a program. Use some sort of visual cue to help you tell them apart. I use lowercase for macro code and uppercase for SAS code. Different means of indentation may help distinguish between the two.

First Checks (or, Is the Appliance Plugged In?)

Obviously, best laid plans sometimes go awry, and the best written programs will sometimes have bugs in them. There are a few quick checks you can do before you plunge head first into debugging.

Problem: SAS doesn't recognize your macro

• Check that the MACRO and DQUOTE options are on.
• If your macro is in an autocall library, make sure that the MAUTOSOURCE option is on.

Problem: Macro variable not recognized

• Check to see whether the macro variable is local or global. A macro variable is only defined for the macro it was created in and any macros that it calls, unless the %GLOBAL statement is used.
• Check to see if the value of the macro variable is uppercase or mixed case. For example:

  %put Which option?;
  %input opt;
  %if %upcase(&opt) = 'YES' then PROC PRINT;

  If the user entered a lowercase "a" this comparison will not be true. Also note that the CAPS option does not affect macro data.
• Make sure that you are not trying to use CALL SYMPUT to create a macro variable, then use the macro variable in the same DATA step. For example:

  CALL SYMPUT('N',NVAR);
  DO I = 1 TO &nvar;

• If you are using SAS/AF(r), remember that a SAS/AF field is not a macro variable. Also, remember that if the associated SAS macro variable is used to initialize the SAS/AF field when the screen is first displayed, and the value of the SAS/AF field is stored in the macro variable when the screen is exited, the macro variable does not change value during display of the screen, and cannot be used in the program that way.

Some syntax checks:

Here are some quick syntax checks you can make:

• Not enough or too many semicolons:

  %if %upcase(&print) = 'YES' then PROC PRINT;

  In the above example, the semicolon closes the macro statement. There is no semicolon generated for the PROC PRINT statement. The statement should read:

  %if %upcase(&print) = 'YES' then PROC PRINT %str();

• Not enough or too many parentheses:

  %let sub = %scan(&string,%eval(&i+1);

  There should be two right parentheses before the semicolon; one to close the %SCAN function and one to close the %EVAL function.

• Not enough ampersands and/or percent signs:

  %do i = 1 to 5;
  to should be %to.

• Too many ampersands and/or percent signs:

  %let &i = %eval(&i+1);

  There should not be an ampersand before the first "i" in the expression. This tells the macro processor that the macro variable &i needs to be resolved to the name of a macro variable before the expression can be evaluated.

• Ampersands where percent signs should be, and vice versa:

  %let i = %eval(&i+1);

  The ampersand before eval should be a percent sign. EVAL is a macro function, not a macro variable.

• An incorrect or missing %MEND statement. If you are using VM/CMS, the name of the macro must appear on the %MEND statement as well as the %MACRO statement.

Problem Determination

If the simple checks don't take care of the problem, some investigation is necessary to figure out what's wrong. One of the things that makes macros difficult to debug is that, because the macro language is essentially a SAS code generator, the error could be in either the macro itself or in the SAS code that the macro is generating.

The first place to look to determine the source of the error is at the number of the error messages that are produced. The errors are coded by the number of digits in the error number:

• An error message with no number or one or three digits indicates that the macro executed without error, but the SAS code that it generated is in error.

• An error message with four or five digits indicates that the macro executed without error, but the SAS code that it generated is in error.

It's best to go after the macro errors first. They occurred first, and they are often the cause of the SAS errors.

• Check errors in the macro definition first - those errors between the %MACRO and %MEND statements. They are usually syntax errors, and are easier to spot:
ERROR 1116: EXPECTED %THEN NOT FOUND.

* Next, check macro errors that occur when the macro is invoked. Often, there is a macro variable which is resolving to a null string and causing an unexpected resolution of a macro statement. The %PUT statement and the MLOGIC option are useful to spot these. (See "Tools for Debugging" below.)

* Some macro "errors" are not errors at all. Remember that the word scanner passes all words beginning with an ampersand or percent sign to the macro processor. Some times these are simply part of the text:

```
TITLE "A&P REVENUES";
```

If the error is in the SAS code (no number, or one or three digits), here are some things you can do to track it down:

* Check the macro definition itself to make sure there are no syntax errors or misspellings in the macro definition:

```
%if %upcase(&opt) = MEANS
   %then PORC MEANS DATA=&dsn %str(·); 
%else PROC PRINT DATA=&dsn %str();
```

The above will produce an error when the MEANS option is specified, because "PROC" is misspelled.

* Use the MPRINT option to display the SAS code that this run of the macro has generated.

* Use the SYMBOLGEN option to show the resolution of macro variables in open code, to make sure that they are being resolved correctly.

* If the SAS code which the macro is generating looks correct, but you are getting errors anyway, the problem could be one of quoting or unquoting. Refer to Chapter 9 of the "SAS(r) Guide to Macro Processing" for more information about quoting functions.

Tools for Debugging

When debugging macros, or any language for that matter, the key rule I go by is that you can't debug what you can't see - or at least, it's a lot harder. When trying to figure out what is causing the errors, don't treat the problem like an academic test or a puzzle, where it's cheating if you look up the answer. Print everything out that you can!

Here are some useful SAS options for debugging macros. They provide extra information in the SAS log.

* MPRINT: shows generated SAS code, including resolution of macro variables

* SYMBOLGEN: shows resolution of all macro variables, including those used in open code

* MACROGEN: shows generated SAS code, except for the resolution of macro variables

* MLOGIC: shows the branching logic of macro execution. This is hard to read and produces a lot of output. It should be used only as a last resort.

Examples of the output appear in the Appendix.

The %PUT statement can also be very useful to pinpoint problems:

* If the problem is that your macro variables are not being set correctly, you can display their values.

* You can show the flow of execution with %PUT statements, also.

For example:

```
%if %upcase(&opt) = MEANS
   %then %do;
      %put Doing PROC MEANS, dsn= <&dsn>; 
      PROC MEANS DATA=&dsn %str(·);
   %end;
```

It is very useful to use delimiters to verify whether there are excess blanks in a macro variable. Sometimes, it matters whether the value of &dsn is "XYZ" or "XYZ •

If none of the above pinpoints the problem, here are some other more elaborate things you can try:

* If the problem is in the generated SAS code, you can use the MPRINT option to list the SAS code in the SAS log, then edit the SAS log into a file of SAS program statements, then run the SAS program outside the macro. (Watch your file format!)

* Try breaking the macro into smaller macros, then run each smaller macro separately. Work on the one that breaks.

Resources and References

To learn more about macros, and debugging macros, you can check these:

* The "SAS(r) Guide to Macro Processing" is a manual dedicated to explaining the macro language.

* The "SAS(r) Guide to Problem Solving and Error Messages" is a very good description of common problems encountered in SAS. Chapter 3 is dedicated to the macro language.

* To avoid errors resulting from misunderstandings about macros, the SAS(r) Macro Language course is a
very worthwhile use of time. In addition to macro language syntax, the course goes into how the SAS system works. A clear understanding of that takes you a long way to avoiding problems.

Conclusion

The SAS macro language is very useful, but its complexity and the similarity of its syntax to SAS DATA step syntax makes macros particularly difficult to debug. This paper has presented some tools and techniques that can be used to debug SAS macros.

Note

SAS and SAS/AF are registered trademarks of SAS Institute, Inc.

Appendix: Output Generated by Macro Display SAS Options

MLOGIC

3 %macro show(dsn=LAST, opt=PRINT, var=); 4 %if %upcase(&opt)=MEANS 5 %then PROC MEANS DATA=&dsn %str(); 6 %else PROC PRINT DATA=&dsn %str(); 11 if &var= 12 %then VAR &var %str(); 15 RUN; 16 %mend show; 17 8>%upcase(&opt)=MEANS 1-%MLOGIC> 17>&var= 1-%MLOGIC>
24 %show(dsn=TEMP)

SYMBOLGEN

3 %macro show(dsn=LAST, opt=PRINT, var=); 4 %if %upcase(&opt)=MEANS 5 %then PROC MEANS DATA=&dsn %str(); 6 %else PROC PRINT DATA=&dsn %str(); 11 if &var= 12 %then VAR &var %str(); 15 RUN; 16 %mend show; 29+TEMP 3-%SYMBOL
24 %show(dsn=TEMP)

MPRINT

3 %macro show(dsn=LAST, opt=PRINT, var=); 4 %if %upcase(&opt)=MEANS 5 %then PROC MEANS DATA=&dsn %str(); 6 %else PROC PRINT DATA=&dsn %str(); 11 if &var= 12 %then VAR &var %str(); 15 RUN; 16 %mend show;

24 %show(dsn=TEMP) 25+ PROC PRINT DATA=
1-%show 26+ TEMP ; 1-%show 27+ RUN; 1-%show

MACROGEN

3 %macro show(dsn=LAST, opt=PRINT, var=); 4 %if %upcase(&opt)=MEANS 5 %then PROC MEANS DATA=&dsn %str(); 7+ 1-%STR 8 9 %else PROC PRINT DATA=&dsn %str(); 10+ 1-%STR 8 11 if &var= 12 %then VAR &var %str(); 14+ 1-%STR 12 15 RUN; 16 %mend show; 24 %show(dsn=TEMP) 28+ PROC PRINT DATA=&dsn ; 1-%show 31+ RUN; 1-%show

MLOGIC SYMBOLGEN MACROGEN

9>%PRINT 2-%SYMBOL 10=%PRINT 2-%UPCASE 8>%upcase(&opt)=MEANS 1-%MLOGIC> 24 %show(dsn=TEMP) 29+ TEMP 3-%SYMBOL 28+ PROC PRINT DATA=&dsn ; 1-%show 18> 2-%SYMBOL 17>&var= 1-%MLOGIC> 31+ RUN; 1-%show