Debugging SAS® Macros

Thomas J. Winn Jr., Texas State Comptroller's Office, Austin, Texas

Abstract

The SAS® macro facility is a powerful tool which greatly enhances the functionality and flexibility of the SAS programming language. However, the complexity of programming code which utilizes SAS macro language statements sometimes also makes it difficult to debug. The SAS System includes some programming statements and options which can be used in the process of correcting code that is not working properly. This tutorial will describe a strategy for using the MPRINT, MLOGIC and SYMBOLGEN options, as well as %PUT statements, for debugging purposes, including examples of the interpretation of diagnostic messages pertaining to the macro facility.

Introduction

Using the capabilities of the SAS macro language is a wonderful way to handle repetitive coding situations, and to produce flexible code with the ability to accommodate itelf to various changeable details. Macros permit a modularized approach which is particularly useful in applications development. On the other hand, since most of the operations of the macro facility are carried out in the background, and not in the more-readily-visible foreground, this makes it difficult to detect coding errors involving macros and macro variables. The good news is that there are certain programming statements and options which can be used to prompt the macro facility to reveal some of the crucial elements of what is going-on "behind the onstage scenery".

Debugging is a *process* by which coding errors in a computer program are identified and corrected through a sequence of activities, each of which must be adapted according to the outcomes of the preceding steps. The goal of this paper will be to develop a systematic approach for dealing with coding errors involving macros and macro variables.

An Overview of the SAS Macro Language

Macros are stored text which contain entire blocks of SAS code, and which are identified by a name. The stored text can include SAS statements, literals, numbers, macro variables, macro functions, macro expressions, or calls to other macros. *Macro variables* are used to facilitate symbolic substitution of strings of text, whereas *macros* can be used to manipulate SAS source statements.

Macro information can be inserted at any point in a SAS program simply by referring to the macro entity by name, preceded by a special character, which distinguishes macro statements from ordinary SAS code. Macro variables are identified using the ampersand, and macros are identified using the percent sign. The macro facility constructs and edits SAS source statements by substituting the currently-defined values of macro variables, and also by replacing macro names with the stored text which is associated with each of them.

The macro facility follows instructions which are written using a special language. The SAS macro language has variables, statements, functions, expressions, and syntax. It works in conjuction with the SAS programming language. For additional introductory information about the SAS macro language, you should consult one of the standard references on the subject.

Putting First Things First

Okay, so what do you do if you invoke a macro and nothing happens?

If you were going to troubleshoot an inoperative electric appliance, you wouldn't begin by disassembling the article. First, you would check to see whether or not there really is a problem -- before going any further, you would make sure the appliance is plugged-in, and that it is turned-on.

Similarly, before beginning extensive debugging activities conceming macros, first make sure that the macro facility is available to your SAS program. That is, check to ensure that the appropriate system options are in effect. To obtain a list of the current values of all SAS system options on your computer, run the OPTIONS procedure (or in display manager, invoke the OPTIONS window).

```
PROC OPTIONS;
RUN;
```

Following are a few lines excerpted from the SAS Log of a job. This display is *not* a complete list of all of the system options which pertain to the macro facility; and it is only a very small selection of all of the system options.

```
2          OPTIONS NODATE LINESIZE=70;
3
4          PROC OPTIONS;    /*  EXAMINE SAS SYSTEM OPTIONS  */
5          RUN;

PORTABLE OPTIONS:

 MACRO              Perform macro processing?
 MAUTOSOURCE        Allow SAS macro automatic call from source library
 MERROR             Treat apparent undefined macro references as an
                    error?
 NOMLOGIC           Trace macro execution?
 NOMPRINT           Print macro facility results in a synthetic
                    compressed form?
 NOMRECALL          Attempt to look up autocall macros which were not
                    found previously?
 NOMSTORED          Use stored compiled macros?
 SERROR             Consider undefined macro variable references an
                    error?
 NOSYMBOLGEN        Print symbolic replacement text?
```

Make sure that the system option MACRO is in effect, rather than NOMACRO -- this ensures that the macro facility is "turned-on." Also, it is a good idea for the MERROR and SERROR options to be in effect, instead of NOMERROR and NOSERROR, respectively. Otherwise, you might never find out that the reason nothing happened was that the macro processor couldn't find the stored text or program code you thought was associated with a particular name. MERROR and SERROR ensure that you will get the appropriate warning messages when you attempt to refer to macro entitites which do not exist where you think they should. Also, if you need to use a macro which is not defined within your current program, but which exists as a member in an autocall library, then you need to make sure that the MAUTOSOURCE system option is in effect, rather than NOMAUTOSOURCE.

Okay, now let us suppose that when the macro was invoked, *something* did happen, but *not* what was anticipated. Was an error message sent to the SAS Log? If so, then examine it to determine if the error message was generated by the macro facility, or from somewhere else in the SAS System. Distinguishing between different types of problems helps the troubleshooter decide what to do next.

Error messages concerning macro language statements may not appear to point to coding problems as specifically as error messages which pertain to ordinary SAS statements. It is helpful if the troubleshooter understands the source of the error. Most macro activity takes place during the compilation of SAS code. The SAS compiler does not recognize the existence of macro entities, and does not perceive when a symbolic substitution has been made.

Types of Macro Programming Errors

Here are some major categories of errors in macro program code (also see Phillips, Walgamotte & Drummond, p. 361):

* macro syntax errors,
* SAS code construction errors,
* macro invocation errors,
* macro variable resolution errors,
* DATA Step Interface errors,
* problems with special characters.

Macro Syntax Errors

As with ordinary SAS code, one of the most common errors in the use of SAS macro language statements is incorrect syntax. This includes such things as, for example, misspelled keywords, omitting the percent sign from macro language keywords, and omitting the ampersand from references to macro variables.

SAS Code Construction Errors

This type of error can occur whenever macro-generated text is not used appropriately in conjunction with ordinary SAS code. This isn't a macro problem, per se, but a problem with ordinary SAS involving information substituted by the macro facility.

Macro Invocation Errors

There are several situations which can result in invocation errors. Macros must be defined before they can be invoked. Furthermore, even when macros are correctly defined, they still can fail if they are not called properly. Some programmers occasionally forget to invoke the macro at all -- for some reason, many SAS users need to be reminded that just defining a macro doesn't cause it to execute. Invocation errors also can occur whenever essential parameter values are missing. Another common mistake is using a style of macro invocation which may be wrong for the particular installation (there are three alternative styles of macro invocations, each having its own system option). Attempted invocation of a macro which has not been defined will result in the appearance in the SAS Log of a message similar to the following:

```
WARNING: Apparent invocation of macro XXXXX not resolved.
```

Macro Variable Resolution Errors

Macro variable references can fail to resolve for a variety of reasons, such as: misspelling the name of the macro variable, enclosing the macro variable reference in single quotes, or attempting to reference the macro variable outside of its own referencing environment (that is, outside of the macro within which it was created). Often, this type of error will result in the appearance of a message in the SAS Log which is similar to the following:

```
WARNING: Apparent symbolic reference XXXXX not resolved.
```

DATA Step Interface Errors

A frequent error in trying to pass information from a SAS DATA step to the macro facility is attempting to use the %LET statement to define a macro variable, instead of using the SYMPUT function. Another common DATA step interface problem is attempting to reference a macro variable before the execution of the DATA step.

Problems With Special Characters

Error messages result whenever there are unmatched quotation marks (apostrophes) and parentheses, or incomplete SAS statements because a semicolon was unintentionally interpreted as the end of a SAS statement, instead of an element in a character string. The SAS language uses several special characters for syntactical purposes, such as semicolons or other delimiters, logical operators, quotes, and parentheses. Sometimes, however, it is desirable to have the macro facility ignore the syntactical meaning of certain special characters, and to treat them as nothing more than plain text in a character string. Moreover, errors also can result whenever SAS interprets "IN", "OR", or "NE" as logical operators, instead of as state abbreviations. There are ways to avoid these problems. The SAS literature refers to these solutions as "quoting functions".

Macro Debugging Tools

The SAS System includes some system options and programming statements which can be used to display information in the SAS Log, to help the SAS programmer to locate coding errors involving macro information which caused the program to fail to operate correctly.

The MPRINT system option is specified to display all of the SAS code which is generated as a final result by the macro facility. The SYMBOLGEN system option is used to show the result of every macro variable substitution. The MLOGIC system option shows when each statement within a macro executes. The %PUT statement can be used to print messages in the SAS Log regarding the value of a macro variable at a particular location in a macro, or to indicate the attainment of a particular point in the execution of a macro.

Debugging Examples

Several examples of debugging are presented for illustrative purposes. All of the examples use the following SAS data set:

```
DATA FIRMS;
   INPUT IDNUM    $  7-13    COMPANY $ 16-41
         INDCODE $ 44-47    CITY    $ 50-62
         SALES     65-70;
   TITLE "HYPOTHETICAL DATA CONCERNING CLIENT COMPANIES";
   CARDS;
      0123456  ABCD CORPORATION            5441  DALLAS          92589
      1234567  XYZ INVESTMENTS CO.         6282  HOUSTON         13217
      2345678  THE WINN COMPANY            5399  AUSTIN         585036
      3456789  AMAZING RESOURCES CORP.     1311  FORT WORTH     209724
      4567890  BOGUS PRODUCTS CO.          3089  DALLAS         194473
      5678901  PRETEND MANUFACTURING CO.   3931  SAN ANTONIO     65902
      6789012  IMAGINARY SERVICES INC.     7389  HOUSTON        136758
      7890123  TOM'S FINANCIAL CORP.       6141  AUSTIN         442106
      8901234  T-W INTERNATIONAL LTD.      5046  EL PASO        288290
      9012345  TEXAS DELUSIONS, INC.       7311  HOUSTON         58995
;;;;
```

Unless specified otherwise, the example programs were run using the default system options:

```
OPTIONS NOMPRINT NOSYMBOLGEN NOMLOGIC;
```

An Example of a SAS Code Construction Error

Here is a small portion of a SAS Log for a program which did not function as the programmer intended:

```
21          %LET SLCTCTY=AUSTIN;
22          DATA SLCTFRMS;
23             SET FIRMS;
24             IF CITY=&SLCTCTY;
25

NOTE: Variable AUSTIN is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 0 observations and 6 variables.
```

We presume that the value of the macro variable SLCTCTY was supposed to be used as the criterion for subsetting the FIRMS data set. We expected that the data set SLCTFRMS would contain the two Austin observations but, instead, there are *no* observations in it. And what is the significance of the message, "Variable AUSTIN is uninitialized"?

Think about it. The macro variable SLCTCTY correctly resolved to the value AUSTIN but, because this character string isn't enclosed in quotation marks, the SAS Compiler interpreted the substituted value as a missing *variable* appearing as the right-hand side of an assignment statement.

If we were uncertain about the value to which the macro variable SLCTCTY resolved, we could run the program again, this time using the system option SYMBOLGEN. Here is what we would get:

```
21          OPTIONS SYMBOLGEN;
22
23          %LET SLCTCTY=AUSTIN;
24          DATA SLCTFRMS;
25             SET FIRMS;
26             IF CITY=&SLCTCTY;
SYMBOLGEN:  Macro variable SLCTCTY resolves to AUSTIN
27

NOTE: Variable AUSTIN is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 0 observations and 6 variables.
```

This is a simple example of a SAS code construction error. The macro facility is doing its job correctly, the problem is with ordinary SAS. The correct programming statement would have been: `IF CITY="&SLCTCTY";`

An Example of a Macro Variable Resolution Error

Here is another snippet from a SAS Log:

```
21          %LET SLCTCTY=AUSTIN;
22          DATA SLCTFRMS;
23             SET FIRMS;
24             IF CITY="&SLCTCTY";
25

NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The DATA statement  used 0.01 CPU seconds and 2916K.

26          PROC PRINT DATA=SLCTFRMS;
27             TITLE "OBSERVATIONS FROM DATASET WITH CITY=&SELCITY";
WARNING: Apparent symbolic reference SELCITY not resolved.
28
NOTE: The PROCEDURE PRINT printed page 1.
```

We notice that the macro facility was unable to resolve the macro variable SELCITY in the TITLE statement. If we examine the program carefully, we should conclude that the reason this variable did not resolve is that no macro variable with this name was ever defined. The macro variable name was misspelled.

If we ran the program using the system option SYMBOLGEN, we would obtain the following result:

```
21          OPTIONS SYMBOLGEN;
22
23          %LET SLCTCTY=AUSTIN;
24          DATA SLCTFRMS;
25             SET FIRMS;
26             IF CITY="&SLCTCTY";
SYMBOLGEN:  Macro variable SLCTCTY resolves to AUSTIN
27

NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The DATA statement  used 0.01 CPU seconds and 2916K.

28          PROC PRINT DATA=SLCTFRMS;
29             TITLE "OBSERVATIONS FROM DATASET WITH CITY=&SELCITY";
WARNING: Apparent symbolic reference SELCITY not resolved.
30
NOTE: The PROCEDURE PRINT printed page 1.
```

The programming statement should have been:

```
TITLE "OBSERVATIONS FROM DATASET WITH CITY=&SLCTCTY";
```

An Example of a Macro Syntax Error
(Misspelled Macro Keyword)

The following is an excerpt from a SAS Log:

```
23          %MACRO AVGVAL(SASDSN,SASVAR);
24             PROC MEANS DATA=&SASDSN N MEAN;
25                VAR &SASVAR;
26                TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27             RUN;
```

```
28         %END AVGVAL;
ERROR: There is no matching %DO statement for the %END.  This
       statement will be ignored.
NOTE: Extraneous information on %END statement ignored.
29
30         %AVGVAL(FIRMS,SALES)
31
WARNING: Missing %MEND statement.

ERROR: Errors printed on page 1.
```

Evidently, the programmer misspelled a macro keyword. Instead of `%END AVGVAL;`, the offending line of code should have been `%MEND AVGVAL;`. How could a would-be troubleshooter figure this out? The hint is that %DO and %END must always appear together in pairs. Therefore, *something* isn't quite right about `%END AVGVAL;`. Moreover, in this case, invoking any or all of the macro debugging system options would provide *no* additional assistance toward discovering this mistake. Sometimes there is no substitute for a very careful examination of the code.

Another Example of a Macro Syntax Error
(Omitted Percent Sign)

The following lines were taken from a SAS Log for another program:

```
23         MACRO AVGVAL(SASDSN,SASVAR);
24            PROC MEANS DATA=&SASDSN N MEAN;
25               VAR &SASVAR;
26               TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27            RUN;
28         MEND AVGVAL;
29
30         %AVGVAL(FIRMS,SALES)
30       +(SASDSN,SASVAR);
                 -
                 180

ERROR 180-322: Statement is not valid or it is used out of proper
               order.

31       +   PROC MEANS DATA=&SASDSN N MEAN;
                           -
                           200
WARNING: Apparent symbolic reference SASDSN not resolved.
ERROR: File WORK.SASDSN.DATA does not exist.
32       +      VAR &SASVAR;
                  -
                  200
WARNING: Apparent symbolic reference SASVAR not resolved.
ERROR: No data set open to look up variables.
WARNING: Apparent symbolic reference SASVAR not resolved.
WARNING: Apparent symbolic reference SASDSN not resolved.

ERROR 200-322: The symbol is not recognized.

NOTE: The SAS System stopped processing this step because of errors.
NOTE: SAS set option OBS=0 and will continue to check statements.
      This may cause NOTE: No observations in data set.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2802K.
37

ERROR: Errors printed on page 1
```

The first indication we have that something is wrong is the numbered error message,

`ERROR 180-322: Statement is not valid or it is used out of proper order.`

Following this, there are several instances of warnings

`WARNING: Apparent symbolic reference XXXXX not resolved.`

Actually, it appears as though *none* of the macro-related statements were successful. Closer inspection reveals the reason: the programmer omitted the percent signs in the %MACRO and %MEND statements. Therefore, the intended macro never made it into the macro facility in the first place.

An Example of a Macro Invocation Error
(Specification of Positional Parameters)

What is wrong with the code which resulted in the following portion of a SAS Log?

```
23         %MACRO AVGVAL(SASDSN,SASVAR);
24            PROC MEANS DATA=&SASDSN N MEAN;
25               VAR &SASVAR;
26               TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27            RUN;
28         %MEND AVGVAL;
29
30         %AVGVAL(SALES,FIRMS)
ERROR: File WORK.SALES.DATA does not exist.

NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 3004K.
31

ERROR: Errors printed on page 1.
```

The error message indicates that there is a problem with the SAS data set named SALES -- it doesn't exist! Hmmm. Wasn't the *dataset* called FIRMS, and the *variable* named SALES? When the macro AVGVAL was defined, it had two positional parameters. Positional parameters are dependent upon the order in which they are defined. It seems that the programmer reversed the order of the values of the parameters when the macro was invoked. Instead of `%AVGVAL(SALES,FIRMS)`, the invocation should have been `%AVGVAL(FIRMS,SALES)`.

If the programmer isn't sure just what the value of a macro variable is at a particular point in a program, a %PUT statement can be used, within the appropriate referencing environment, to display the currently defined value in that environment. For example, by inserting the following lines within the macro,

```
%PUT ***** THE CURRENT VALUE OF SASDSN IS &SASDSN.*****;
%PUT ***** THE CURRENT VALUE OF SASVAR IS &SASVAR.*****;
```

we would obtain:

```
23         %MACRO AVGVAL(SASDSN,SASVAR);
24            PROC MEANS DATA=&SASDSN N MEAN;
25               VAR &SASVAR;
26               TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27            %PUT ***** THE CURRENT VALUE OF SASDSN IS &SASDSN.*****;
28            %PUT ***** THE CURRENT VALUE OF SASVAR IS &SASVAR.*****;
29            RUN;
30         %MEND AVGVAL;
31
32         %AVGVAL(SALES,FIRMS)
ERROR: File WORK.SALES.DATA does not exist.
***** THE CURRENT VALUE OF SASDSN IS SALES.*****
***** THE CURRENT VALUE OF SASVAR IS FIRMS.*****

NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 3004K.
33
```

Here is what would have been generated if the example program was run using *all three* of the macro debugging system options, and without the %PUT statements:

```
23         OPTIONS MPRINT SYMBOLGEN MLOGIC;
24
25         %MACRO AVGVAL(SASDSN,SASVAR);
26            PROC MEANS DATA=&SASDSN N MEAN;
27               VAR &SASVAR;
28               TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
29            RUN;
30         %MEND AVGVAL;
31
32         %AVGVAL(SALES,FIRMS)
MLOGIC(AVGVAL):  Beginning execution.
MLOGIC(AVGVAL):  Parameter SASDSN has value SALES
MLOGIC(AVGVAL):  Parameter SASVAR has value FIRMS
SYMBOLGEN:  Macro variable SASDSN resolves to SALES
ERROR: File WORK.SALES.DATA does not exist.
MPRINT(AVGVAL):   PROC MEANS DATA=SALES N MEAN;
SYMBOLGEN:  Macro variable SASVAR resolves to FIRMS
MPRINT(AVGVAL):   VAR FIRMS;
SYMBOLGEN:  Macro variable SASVAR resolves to FIRMS
SYMBOLGEN:  Macro variable SASDSN resolves to SALES
MPRINT(AVGVAL):   TITLE "AVERAGE VALUE OF FIRMS IN DATASET SALES";
MPRINT(AVGVAL):   RUN;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 3004K.
MLOGIC(AVGVAL):  Ending execution.
33

ERROR: Errors printed on page 1.
```

Now, a careful examination of the preceding SAS Log shows the step-by step result of each action taken by the macro facility. MLOGIC traces the execution of macro programming statements. SYMBOLGEN shows the result of each macro variable substitution. MPRINT shows the SAS code which is generated as a final result by the macro facility. If we keep in mind that the original DATA step has a *variable* named SALES in a *data set* called FIRMS while we are reviewing the SAS Log, then we should be able to identify the problem.

Well, you might say, why not just invoke *all* of the macro debugging system options *whenever* you have a macro problem? That approach would be okay for very simple programs but, most of the time, invoking all three options would result in much more printout than you probably would want to look at, particularly if your macro involves iterative execution (looping statements – %DO ... %TO ... %BY ...; and %END;) or nested referencing environments (when one macro calls another macro).

Another Example of a Macro Invocation Error

(Incorrect Macro Name)

Here is a piece of another SAS Log:

```
23          %MACRO AVGVAL(SASDSN,SASVAR);
24             PROC MEANS DATA=&SASDSN N MEAN;
25                VAR &SASVAR;
26                TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27             RUN;
28          %MEND AVGVAL;
29
30          %MEANVAL(FIRMS,SALES)
            _
            180
WARNING: Apparent invocation of macro MEANVAL not resolved.
31

ERROR 180-322: Statement is not valid or it is used out of proper
               order.

ERROR: Errors printed on page 1.
```

The warning message indicates that the job attempted to invoke a macro which was not known to the macro facility. Apparently, the programmer failed to correctly remember the name of the macro. Notice that when the macro facility returned the incorrect code to the wordscanner, the SAS System generated a numbered error message. This type of error message definitely originates from outside of the macro facility. In this case, all that it indicates is that the SAS System didn't know what to do with the incorrect code.

Another Example of a Macro Invocation Error

(Disallowed Style of Invocation)

What is the matter with this SAS Log?

```
23          %MACRO AVGVAL(SASDSN,SASVAR);
24             PROC MEANS DATA=&SASDSN N MEAN;
25                VAR &SASVAR;
26                TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27             RUN;
28          %MEND AVGVAL;
29
30          AVGVAL(FIRMS,SALES);
30          AVGVAL(FIRMS,SALES);
            _____
            180

ERROR 180-322: Statement is not valid or it is used out of proper
               order.

31

ERROR: Errors printed on page 1.
```

In this case, the programmer attempted to use a statement-style of macro invocation which, although it may appear to be consistent with SAS usage, is not allowed at an installation where name-style macro invocations are the rule. Instead of `AVGVAL(FIRMS,SALES);` ,the programmer should have used `%AVGVAL(FIRMS,SALES)` .

Yet Another Example of a Macro Invocation Error

(Missing Parameter Values)

Here is a look at another portion of a SAS Log:

```
23          %MACRO AVGVAL(SASDSN,SASVAR);
24             PROC MEANS DATA=&SASDSN N MEAN;
25                VAR &SASVAR;
26                TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
27             RUN;
28          %MEND AVGVAL;
29
30          %AVGVAL
31
32
ERROR: File WORK.N.DATA does not exist.

NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 3004K.
```

The error message indicates that the SAS System is looking for a SAS data file named N. How can this be? Let's run the example program again, this time we will specify *all three* of the macro debugging system options.

```
23          OPTIONS MPRINT SYMBOLGEN MLOGIC;
24
25          %MACRO AVGVAL(SASDSN,SASVAR);
26             PROC MEANS DATA=&SASDSN N MEAN;
27                VAR &SASVAR;
28                TITLE "AVERAGE VALUE OF &SASVAR IN DATASET &SASDSN";
29             RUN;
30          %MEND AVGVAL;
31
32          %AVGVAL
MLOGIC(AVGVAL):  Beginning execution.
33
34
MLOGIC(AVGVAL):  Parameter SASDSN has value
MLOGIC(AVGVAL):  Parameter SASVAR has value
SYMBOLGEN:  Macro variable SASDSN resolves to
ERROR: File WORK.N.DATA does not exist.
MPRINT(AVGVAL):   PROC MEANS DATA= N MEAN;
SYMBOLGEN:  Macro variable SASVAR resolves to
MPRINT(AVGVAL):   VAR ;
SYMBOLGEN:  Macro variable SASVAR resolves to
SYMBOLGEN:  Macro variable SASDSN resolves to
MPRINT(AVGVAL):   TITLE "AVERAGE VALUE OF  IN DATASET ";
MPRINT(AVGVAL):   RUN;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 3004K.
MLOGIC(AVGVAL):  Ending execution.
35

ERROR: Errors printed on page 1.
```

As we review each step of the execution of the example macro, we notice that the currently assigned values for the macro variables SASDSN and SASVAR are both null. When the macro facility processed the statement,

```
PROC MEANS DATA=&SASDSN N MEAN;
```

then, since the macro variable in this statement resolves to null, the final result which was given to the wordscanner by the macro facility was

```
PROC MEANS DATA=  N MEAN;
```

That explains the error message which we received when the program originally ran, without specifying the debugging options. What was the coding error? The programmer invoked the macro without supplying required values for the positional parameters. The correct invocation would have been something like `%AVGVAL(FIRMS,SALES)`

An Example of a DATA Step Interface Error

Here is a SAS Log for a program which was intended to accomplish a very interesting idea:

```
32          PROC MEANS DATA=FIRMS NWAY;
33             CLASS CITY;
34             OUTPUT OUT=COUNTS N=NUM;

NOTE: The data set WORK.COUNTS has 6 observations and 4 variables.
NOTE: The PROCEDURE MEANS printed page 1
NOTE: The PROCEDURE MEANS used 0.02 CPU seconds and 2982K.

35          PROC SORT DATA=COUNTS;
36             BY DESCENDING NUM;

NOTE: The data set WORK.COUNTS has 6 observations and 4 variables.
NOTE: The PROCEDURE SORT used 0.01 CPU seconds and 3071K.

37          DATA _NULL_;
38             SET COUNTS;
39             IF _N_=1 THEN
40                DO;
41                   IF CITY='AUSTIN' THEN DO;
42                      %LET TOPCITY=AUSTIN;
43                   END;
44                   ELSE IF CITY='DALLAS' THEN DO;
45                      %LET TOPCITY=DALLAS;
46                   END;
47                   ELSE IF CITY='EL PASO' THEN DO;
48                      %LET TOPCITY=EL PASO;
49                   END;
50                   ELSE IF CITY='FORT WORTH' THEN DO;
51                      %LET TOPCITY=FORT WORTH;
52                   END;
53                   ELSE IF CITY='HOUSTON' THEN DO;
54                      %LET TOPCITY=HOUSTON;
55                   END;
56                   ELSE IF CITY='SAN ANTONIO' THEN DO;
57                      %LET TOPCITY=SAN ANTONIO;
58                   END;
59                END;

NOTE: The DATA statement used 0.03 CPU seconds and 3125K.

60          DATA BIGCTY;
61             SET FIRMS;
62             IF CITY="&TOPCITY";

NOTE: The data set WORK.BIGCTY has 1 observations and 5 variables.
NOTE: The DATA statement used 0.01 CPU seconds and 3125K.

63          PROC PRINT;
64             TITLE "THE CITY WITH THE MOST FIRMS IS &TOPCITY";
65          RUN;

NOTE: The PROCEDURE PRINT printed page 2.
NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 3222K.
```

The goal of the program was to use a SAS procedure to identify the city which occurs most frequently in our SAS data set of client companies, and then to use a DATA _NULL_ step to create a macro variable whose value is the name of the city with the greatest number of clients. Then, the macro variable is used as the basis for subsetting the original dataset, in order to create a listing of the companies in the city having the largest number of clients.

Now, a visual inspection of our example dataset tells us that Houston is the city with the largest number of companies. However, here is page 2 from the SAS output file:

```
                THE CITY WITH THE MOST FIRMS IS SAN ANTONIO                2

OBS      IDNUM              COMPANY              INDCODE     CITY        SALES

 1     5678901     PRETEND MANUFACTURING CO.      3931     SAN ANTONIO   65902
```

For some reason, instead of resolving to Houston, the macro variable TOPCITY seems to have taken San Antonio as its value. Now, we could run the program again, this time inserting a PROC PRINT after the PROC SORT, to ensure that Houston turns up at the top of the heap, and also specifying the SYMBOLGEN option, to verify the value of TOPCITY. Unfortunately, this approach wouldn't provide us with any additional information. So, what went wrong? Well, the fundamental problem is that *macro statements cannot be controlled by DATA step execution.* Here is a portion of the SAS Log from a job which utilizes a method which correctly achieves the desired result:

```
32          PROC MEANS DATA=FIRMS NWAY;
33             CLASS CITY;
34             OUTPUT OUT=COUNTS N=NUM;

NOTE: The data set WORK.COUNTS has 6 observations and 4 variables.
NOTE: The PROCEDURE MEANS printed page 1.
NOTE: The PROCEDURE MEANS used 0.02 CPU seconds and 2982K.

35          PROC SORT DATA=COUNTS;
36             BY DESCENDING NUM;

NOTE: The data set WORK.COUNTS has 6 observations and 4 variables.
NOTE: The PROCEDURE SORT used 0.01 CPU seconds and 3071K.

37          DATA BIGCTY;
38             IF _N_=1 THEN
39                DO;
40                   SET COUNTS(RENAME=(CITY=TOPCITY))
41                   CALL SYMPUT('TOPCITY',TRIM(TOPCITY));
42                END;
43             SET FIRMS;
44             IF CITY=TOPCITY;

NOTE: The data set WORK.BIGCTY has 3 observations and 9 variables.
NOTE: The DATA statement used 0.03 CPU seconds and 3162K.

45          PROC PRINT;
46             TITLE "THE CITY WITH THE MOST FIRMS IS &TOPCITY";
47          RUN;

NOTE: The PROCEDURE PRINT printed page 2.
NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 3259K.
```

An Example of an Error Involving Special Characters

Here is a portion of a SAS Log for a program which failed to execute as the programmer intended:

```
23          %MACRO SEEK(NAME);
24             PROC PRINT DATA=FIRMS;
25                WHERE COMPANY="&NAME";
26                TITLE "DATA AVAILABLE FOR &NAME IN FIRMS DATASET";
27             RUN;
28          %MEND SEEK;
29
30          %SEEK(THE WINN COMPANY)

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used 0.02 CPU seconds and 3071K.

ERROR: Macro parameter contains syntax error.
31          %SEEK(TOM'S FINANCIAL CORP.)
32

ERROR: Errors printed on page 1.
```

Where is the problem? The macro was invoked two times. Actually, there was no problem the *first* time the macro was called. The problem was with the invocation

```
%SEEK(TOM'S FINANCIAL CORP.)
```

In this case, the error message

```
ERROR: Macro parameter contains syntax error.
```

refers to the invocation which is reported in the SAS Log *after* the error message. Admittedly, this is somewhat confusing. Let's run the program again, this time we will specify all three of the debugging options. (Note: Since the macro doesn't include any %IF - %THEN - %ELSE logic, we shouldn't be overwhelmed by too much MLOGIC output.) Here is the SAS Log which would be the result:

```
23          OPTIONS MPRINT SYMBOLGEN MLOGIC;
24
25          %MACRO SEEK(NAME);
26             PROC PRINT DATA=FIRMS;
27                WHERE COMPANY="&NAME";
28                TITLE "DATA AVAILABLE FOR &NAME IN FIRMS DATASET";
29             RUN;
30          %MEND SEEK;
31
32          %SEEK(THE WINN COMPANY)

MLOGIC(SEEK):  Beginning execution.
MLOGIC(SEEK):  Parameter NAME has value THE WINN COMPANY
MPRINT(SEEK):   PROC PRINT DATA=FIRMS;
SYMBOLGEN:  Macro variable NAME resolves to THE WINN COMPANY
MPRINT(SEEK):   WHERE COMPANY="THE WINN COMPANY";
SYMBOLGEN:  Macro variable NAME resolves to THE WINN COMPANY
MPRINT(SEEK):   TITLE "DATA AVAILABLE FOR THE WINN COMPANY IN
```

```
DATASET";
MPRINT(SEEK):   RUN;

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used 0.02 CPU seconds and 3071K.

MLOGIC(SEEK):  Ending execution.

MLOGIC(SEEK):  Beginning execution.
ERROR: Macro parameter contains syntax error.
MLOGIC(SEEK):  Ending execution.
33          %SEEK(TOM'S FINANCIAL CORP.)
34

ERROR: Errors printed on page 2.
```

Notice that the error message isn't generated until after the macro begins executing for the second time, and that this error causes the macro to terminate before ever really getting underway. So what is wrong with the macro parameter? Well, the problem is the apostrophe. The SAS System normally expects single quotes to appear in pairs. Single quotes are customarily used in the SAS language for a syntactical purpose which is different from the ordinary uses of an apostrophe in the English language. Here is a form of the invocation which will work correctly:

```
%SEEK(%STR(TOM%'S FINANCIAL CORP.))
```

A Macro Debugging Strategy

Here is some general advice which may be helpful in debugging SAS programs which utilize the macro facility:

* First, check to see which SAS system options are in operation, particularly those which pertain to the macro facility.
* Carefully examine the SAS Log. Try to figure out *where* the error occurred. Don't limit your attention to error messages -- also pay close attention to warning messages and notes.
* If you notice that macro variable references seem to be resolving to unexpected values, then specify the SYMBOLGEN system option.
* If you suspect that a macro variable has a different value than was intended for a particular point in the program, then use a %PUT statement at that point to exhibit the current value.
* If you need to verify that conditional logic within a macro is working correctly, then specify the MLOGIC system option. Be forewarned that this option can generate very many lines to your SAS Log.
* If you suspect that the macro facility is not generating the SAS code which was intended as a result of the execution of a macro, then specify the MPRINT system option.
* Don't assume that the error occurred in the statement which immediately precedes or follows the error message. Error messages concerning macro language statements may not appear to point to coding problems with the same specificity as error messages which pertain to ordinary SAS statements.
* Don't specify more macro debugging system options than you really need for the particular problem you are having.

Conclusion

SAS programs which utilize the macro facility frequently contain complex programming code which may be difficult to debug when not working properly.. The SAS System includes system options and programming statements which can be used to display information in the SAS Log, to locate the cause of the error.

The MPRINT system option is specified to display all of the SAS code which is generated as a final result by the macro facility. The SYMBOLGEN system option is used to show the result of every macro variable substitution. The MLOGIC system option shows when each statement within a macro executes. The %PUT statement can be used to print messages in the SAS Log regarding the value of a macro variable at a particular location in a macro, or to indicate the attainment of a particular point in the execution of a macro.

References

Jeff Phillips, Veronica Walgamotte & Derek Drummond (1994), "Warning: Apparent Macro Invocation Not Resolved . . . Techniques for Debugging Macro Code," SUGI-19 Conference Proceedings, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), SAS Guide to Macro Processing, Version 6, Second Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), "SAS Macro Facility," Chapter 20 of SAS Language: Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.

Thomas J. Winn, Jr. (1991), "The SAS Macro Language: An Overview," Proceedings of the 1991 South-Central SAS Regional Conference, Arlington, Texas: The South-Central SAS Users' Group

The author can be contacted at:

Audit Division Headquarters
Texas Comptroller of Public Accounts
L.B.J. State Office Building, 111 East 17th Street
Austin, TX 78774

Voice: (512) 463-4907
FAX: (512) 475-0349