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 itself 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 conjunction 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 concerning 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.

3 OPTIONS UPDATE LICENSED; -
4 PROC OPTIONS; /* DISPLAY SAS SYSTEM OPTIONS */
5 RUN;

PORTABLE OPTIONS:

- MACRO: Perform macro processing?
- MAPSOURCE: Allow SAS macro automatic call from source library
- NUMERIC: Treat apparent undefined macro references as an error?
- NUMERIC: Trace macro execution?
- NUMERIC: Print macro facility results in a synthetic compressed form?
- NUMERIC: Attempt to look up macro which were not found previously?
- NUMERIC: Consider undefined macro variable references as an error?
- NUMERIC: 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 entities 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 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 $ 16-47
     INCOME $ 48-49 CITY $ 50-52
     SALARIES 46-70;
TITLE "HYPOTHETICAL DATA CONCERNING CLIENT COMPANIES";
CARDS;
```

An Example of a SAS Code Construction Error

```
DATA SLCTFRMS;
DATA SLCTFAAS;
IF CITY=SLCTCTY;
INCOME < 40000;  
NOTE: Variable CITY is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
NOTE: The PROCEDURE PRINT printed page 1.
```

We notice that the macro facility was unable to resolve the macro variable SLCTCTY 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:

```
DATA SLCTFRMS;
DATA SLCTFAAS;
IF CITY="SLCTCTY";
NOTE: Variable CITY is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
NOTE: The PROCEDURE PRINT printed page 1.
```

The programming statement should have been:

```
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
```

An Example of a Macro Variable Resolution Error

```
DATA SLCTFRMS;
DATA SLCTFAAS;
IF CITY=SLCTCTY;
NOTE: Variable CITY is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
NOTE: The PROCEDURE PRINT printed page 1.
```

The following is an excerpt from a SAS Log:

```
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
```

The programming statement should have been:

```
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
```

An Example of a Macro Syntax Error

(Misspelled Macro Keyword)

```
DATA SLCTFRMS;
DATA SLCTFAAS;
IF CITY="SLCTCTY";
NOTE: Variable CITY is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
NOTE: The PROCEDURE PRINT printed page 1.
```

The following is an excerpt from a SAS Log:

```
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 variables.
```

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:

```
DATA SLCTFRMS;
DATA SLCTFAAS;
IF CITY="SLCTCTY";
NOTE: Variable CITY is uninitialized.
NOTE: The data set WORK.SLCTFRMS has 2 observations and 5 variables.
NOTE: The data set WORK.SLCTFAAS has 2 observations and 5 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";
Evidently, the programmer misspelled a macro keyword.

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

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
ERROR 100-322: Statement not valid or it is used out of proper order.
```

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

```
ERROR 100-322: Statement is not valid or it is used out of proper order.
```

Following this, there are several instances of warnings

```
WARNING: Apparent symbolic reference SASDSN not resolved.
WARNING: Apparent symbolic reference SASVAR 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?

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

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 ,****;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

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:

```
OPTIONS WRITEMacro SLOGIC, NOAUTO;
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
```

```
MACRO AVGVAL(SASDSN, SASVAR);
PROC MEANS DATA=SASDSN N MEAN;
VAR SASVAR;
TITLE "AVERAGE VALUE OF SASVAR IN DATASET SASDSN";
RUN;
```

```
BEGIN module AVGVAL:
PARAMETERS SASDSN in WORK;
PARAMETERS SASVAR in WORK;
```

```
END AVGVAL;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

```
NOTE: The PROCEDURE MEANS used 0.01 CPU seconds and 2040K.
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:

```
MLOGIC(AVGVAL): Beginnings execution.
MLOGIC(AVGVAL): Parameter SASVAR has value .
MLOGIC(AVGVAL): Parameter SASDSN has value .
MLOGIC(AVGVAL): Ending execution.
```

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?

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

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 all the installation name-style macro invocations are the rule. Instead of AVGVAL(FIRMS, SALES), the programmer should have used N 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:

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

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.

```
MLOGIC(AVGVAL): Parameter SASDSN has value .
MLOGIC(AVGVAL): Parameter SASVAR has value .
MLOGIC(AVGVAL): Ending execution.
```

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=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 N AVGVAL(FIRMS, SALES).

350
Here is a SAS Log for a program which failed to execute as the programmer intended:

```sas
PROC SORT DATA=COUNTS;
RUN;

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

NOTE: The data set WORK.COUNTS has 6 observations and 4 variables.
NOTE: The DATA step executed 0.03 CPU seconds and 3125K.

PROC PRINT;
RUN;

NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 3222K.

NOTE: The PROCEDURE PRINT printed page 2.

NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 3222K.

NOTE: The PROCEDURE PRINT printed page 2.

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

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 from a job which utilizes a method which correctly achieves the desired result:

```sas
PROC PRINT DATA=FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

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:

```sas
PROC MEANS DATA=FIRMS NWAY;
CLASS CITY;
OUTPUT OUT=COUNTRIES;
NOTE: The data set WORK.COUNTRIES has 6 observations and 4 variables.
NOTE: The PROCEDURE MEANS printed page 1.
NOTE: The PROCEDURE MEANS used 0.02 CPU seconds and 2586K.

PROC SORT DATA=COUNTRIES;
BY DESCENDING NUM;
NOTE: The data set WORK.COUNTRIES has 6 observations and 4 variables.
NOTE: The PROCEDURE SORT used 0.01 CPU seconds and 3071K.
NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 3125K.
NOTE: The PROCEDURE PRINT printed page 2.
```

Here is 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:

```sas
MPRINT(SEEK), WHERE COMPANY='THE WINN COMPANY'
MLOGIC(SEEK), Parameter NAME has value THE WINN COMPANY
MLOGIC (SEEK), Beginning execution.
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

And finally, let's look at the code which produced the error message:

```sas
MACRO SEEK1NAME);
MENO SEEK;
MACRO SEEK (NAME);
```

where the invocation

**ERROR: Parameter NAME has value THE WINN COMPANY.**

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:

```sas
MPRINT(SEEK), WHERE COMPANY='THE WINN COMPANY'
MLOGIC(SEEK), Parameter NAME has value THE WINN COMPANY
MLOGIC (SEEK), Beginning execution.
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

An Example of a DATA Step Interface Error

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

```sas
PROC MEANS DATA=FIRMS NWAY;
CLASS CITY;
OUTPUT OUT=COUNTRIES;
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 2586K.
```

Here is 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:

```sas
MACRO SEEK1NAME);
MENO SEEK;
MACRO SEEK (NAME);
```

where the error message

**ERROR: Parameter NAME has value THE WINN COMPANY.**

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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```

where 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:

```sas
DATA NULL;
SET FIRMS;
WHERE COMPANY='THE WINN COMPANY';
RUN;
```
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.

References


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

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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.