CHAPTER
10

Macro Facility Error Messages and Debugging

General Debugging Information

Because the macro facility is such a powerful tool, it is also complex, and debugging large macro applications can be extremely time-consuming and frustrating. Therefore, it makes sense to develop your macro application in a way that minimizes the errors and that makes the errors that do occur as easy as possible to find and fix. The first step is to understand what kind of errors can occur and when they manifest themselves. Then, develop your macros using a modular, layered approach. Finally, use some built-in tools such as system options, automatic macro variables, and the %PUT statement to diagnose errors.

Note: To receive certain important warning messages about unresolved macro names and macro variables, be sure the system options SERROR and MERROR are in effect. See Chapter 13, “Macro Language Dictionary,” for more information on these system options. △

Understanding What Kind of Errors You Can Encounter

When the word scanner processes a program and finds a token in the form of **&** or **%**, it triggers the macro processor to examine the name token that follows the **&** or **%**. Depending on the token, the macro processor initiates one of the following activities:

- □ macro variable resolution
- □ macro open code processing
- □ macro compilation
- □ macro execution.

An error can occur during any one of these stages. For example, if you misspell a macro function name or omit a necessary semicolon, that is a *syntax error* during compilation. Syntax errors occur when program statements do not conform to the rules of the macro language. Or, you may refer to a variable out of scope, causing a *macro variable resolution error*. *Execution errors* (also called *semantic errors*) are usually errors in program logic. They can occur, for example, when the text generated by the macro has faulty logic (statements not executed in the right order or in the way you expect).

Of course, your macro code can be perfect–that does not guarantee that you will not encounter errors caused by plain SAS code, such as a libref not defined, or a syntax error in open code (that is, outside of a macro definition), or a typo in the code your macro generates. Typically, error messages with numbers are plain SAS code error messages; error messages generated by the macro processor do not have numbers.

Developing Bug-free Macros

When programming in any language, it is good technique to develop your code in modules. That is, instead of writing one massive program, develop it piece by piece, test each piece separately, then put the pieces together. This technique is especially useful when developing macro applications because of the two-part nature of SAS macros: macro code and the SAS code generated by the macro code.

Another good idea is to proofread your macro code for common mistakes before you submit it.

The following list gives some simple things whose omission can cause errors–by proofreading your code, you can catch these problems before compiling your code:

- □ the names in the %MACRO and %MEND statements match, and there is a %MEND for each %MACRO.
- □ the number of %DO statements matches the number of %END statements.
- □ %TO values for iterative %DO statements exist and are appropriate.
- □ all statements end with semicolons.
- □ comments begin and end correctly and do not contain unmatched single quotation marks.
- □ macro variable references begin with **&** and macro statements begin with **%**.
- □ macro variables created by CALL SYMPUT are not referenced in the same DATA step in which they are created.
- □ statements that execute immediately (such as %LET) are not part of conditional DATA step logic.
- □ single quotation marks are not used around macro variable references (such as in TITLE or FILENAME statements). When used in quoted strings, macro variable references resolve only in strings marked with double quotation marks.

- □ macro variable values do not contain any keywords or characters that could be interpreted as mathematical operators. (If they do contain such characters, use the appropriate macro quoting function.)
- □ macro variables, %GOTO labels, and macro names do not conflict with reserved SAS System and host environment keywords.

Troubleshooting Your Macros

Table 10.1 on page 109 lists some problems you may encounter when working with the macro facility. Because for many of these situations error messages aren't written to the SAS log, solving them can be hard. For each problem, the table gives some possible causes and some solutions.

Table 10.1 Commonly Encountered Problems

Problem	Cause(s)	Explanation
Display manager session hangs after you submit a macro definition. You type and submit code but nothing happens.	□ Syntax error in %MEND statement □ Missing semicolon, parenthesis, or quotation mark □ Missing %MEND statement □ Unclosed comment	The %MEND statement is not recognized and all text is becoming part of the macro definition.
Display manager session hangs after you call a macro	An error in invocation, such as forgetting to provide one or more parameters, or forgetting to use parentheses when invoking a macro that is defined with parameters.	The macro facility is waiting for you to finish the invocation.
The macro does not compile when you submit it.	A syntax error exists somewhere in the macro definition.	Only syntactically correct macros are compiled.
The macro does not execute when you call it or partially executes and stops.	□ A bad value was passed to the macro (for example, as a parameter). □ A syntax error exists somewhere into the macro definition.	A macro successfully executes only when it receives the correct number of parameters that are of the correct type.
The macro executes but the SAS code gives bad results or no results.	Incorrect logic in the macro os SAS code.	Computers do what you tell them, not what you intended to tell them.

Problem	Cause(s)	Explanation
Code runs fine if submitted as open code, but when generated by a macro, the code doesn't work and issues strange error messages.	□ Tokenization is not as you intended. □ A syntax error exists somewhere in the macro defintion.	Rarely, macro quoting functions alter the tokenization of text enclosed in them. Use the %UNQUOTE function. See "%UNQUOTE" in Chapter 13.
A %MACRO statement generates "invalid statement" error.	□ The MACRO system option is turned off. □ A syntax error exists somewhere in the macro defintion.	For the macro facility to work, the MACRO system option must be one. Edit your SAS configuration file accordingly.

Table 10.2 on page 111 lists some common macro error and warning messages. For each message, some probable causes are listed, and pointers to more information are provided.

Table 10.2 Common Macro Error Messages and Causes

Error Message	Possible Causes	For More Information
`Apparent invocation of macro xxx not resolved.`	☐ You have misspelled the macro name. ☐ MAUTOSOURCE system option is turned off. ☐ MAUTOSOURCE is on, but you have specified an incorrect path in the SASAUTOS=system option. ☐ You are using the autocall fackility but have given the macro and file different names. ☐ You are using the autocall facility but didn't give the file the .SAS extension ☐ There is a syntax error within the macro defintion.	☐ Check the spelling of the macro name ☐ "Solving Problems with the Autocall Facility" on page 121. ☐ "Developing Bug-free Macros" on page 112.
`Apparent symbolic reference xxx not resolved.`	☐ You are trying to resolve a macro variable in the same DATA step as the CALL SYMPUT that created it. ☐ You have misspelled the macro variable name. ☐ You are referencing a macro variable that is not in scope. ☐ You have omitted the period delimiter when adding text to the end of the macro variable.	☐ "Resolving Timing Issues" on page 118 ☐ Check the spelling of the macro variable ☐ "Solving Problems with Macro Variable Scope" on page 115 ☐ "Solving Macro Variable Resolution Problems" on page 114 ☐ "Generating a Suffix for a Macro Variable Reference" in Chapter 1.

Solving Macro Variable Resolution Problems

When the macro processor examines a name token that follows an **&**, it searches the macro symbol tables for a matching macro variable entry. If it finds a matching entry, it pulls the associated text from the symbol table and replaces **&name** on the input stack. When a macro variable name is passed to the macro processor but the processor does not find a matching entry in the symbol tables, it leaves the token on the input stack and generates this message:

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

The unresolved token is transferred to the input stack for use by other parts of the SAS System.

Note: You receive the WARNING only if the SERROR system option is on. △

To solve these problems, check that you've spelled the macro variable name right and that you are referencing it in an appropriate scope.

When a macro variable resolves but does not resolve to the correct value, you can check several things. First, if the variable is a result of a calculation, make sure the correct values were passed into the calculation. And, make sure you have not inadvertently changed the value of a global variable. (See "Solving Problems with Macro Variable Scope" on page 112 for more details on variable scope problems.)

Another common problem is adding text to the end of a macro variable but forgetting to add a delimiter that shows where the macro variable name ends and the added text begins. For example, suppose you want to write a TITLE statement with a reference to WEEK1, WEEK2, and so on. You set a macro variable equal to the first part of the string, then supply the week's number in the TITLE statement:

```
%let wk=week;

title "This is data for &wk1";   /* INCORRECT */
```

When these statements compile, the macro processor looks for a macro variable named WK1, not WK. To fix the problem, add a period (the macro delimiter) between the end of the macro variable name and the added text, as in the following statements:

```
%let wk=week;

title "This is data for &wk.1";
```

CAUTION:

Do not prefix macro variable names with AF, DMS, or SYS. The letters AF, DMS, and SYS are frequently used by the SAS System as prefixes for automatic variables. SAS does not prevent you from using AF, DMS, or SYS as a prefix for macro variable names. However, using these strings as prefixes may create a conflict between the names you specify and the name of an automatic macro variable (including automatic macro variables in later SAS releases).

If a name conflict occurs, SAS may not issue a warning or error message, depending on the details of the conflict. Therefore, the best practice is to avoid using the strings AF, DMS, or SYS as the beginning characters of macro names and macro variable names. △

Solving Problems with Macro Variable Scope

A common mistake that occurs with macro variables concerns referencing local macro variables outside their scope. As described in Chapter 5, "Scope of Macro Variables," macro variables are either global or local. Referencing a variable outside its scope prevents the macro processor from resolving the variable reference. For example, consider the following program:

```
%macro totinv(var);
   data inv;
      retain total 0;
      set sasuser.houses end=final;
      total=total+&var;
      if final then call symput("macvar",put(total,dollar14.2));
   run;
   %put **** TOTAL=&macvar ****;
%mend totinv;
```

```
%totinv(price)
%put **** TOTAL=&macvar ****;   /* ERROR */
```

When you submit these statements, the %PUT statement in the macro TOTINV writes the value of TOTAL to the log, but the %PUT statement that follows the macro call generates a warning message and writes the text **`TOTAL=&macvar`** to the log, as follows:

```
TOTAL= $1,240,800.00
WARNING: Apparent symbolic reference MACVAR not resolved.
**** TOTAL=&macvar ****
```

The second %PUT statement fails because the macro variable MACVAR is local to the TOTINV macro. To correct the error, you must use a %GLOBAL statement to declare the macro variable MACVAR.

Another common mistake that occurs with macro variables concerns overlapping macro variable names. If, within a macro definition, you refer to a macro variable with the same name as a global macro variable, you affect the global variable, which may not be what you intended. Either give your macro variables distinct names or use a %LOCAL statement to explicitly define a local macro variable. See "Forcing a Macro Variable to Be Local" in Chapter 5 for an example of this technique.

Solving Open Code Statement Recursion Problems

Recursion is something calling itself. *Open code recursion* is when your open code erroneously causes a macro statement to call another macro statement. The most common error that causes open code recursion is a missing semicolon. In the following example, the %LET statement is not terminated by a semicolon:

```
%let a=b   /* ERROR */
%put **** &a ****;
```

When the macro processor encounters the %PUT statement within the %LET statement, it generates this error message:

```
ERROR: Open code statement recursion detected.
```

Open code recursion errors usually occur because the macro processor is not reading your macro statements as you intended. Careful proofreading can usually solve open code recursion errors, because this type of error is mostly the result of typos in your code, not errors in execution logic.

To recover from an open code recursion error, first try submitting a single semicolon. If that does not work, try submitting the following string:

```
*'; *"; *); */; %mend; run;
```

Continue submitting this string until the following message appears in the SAS log:

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

If the above method does not work, close your SAS session and restart SAS. Of course, this causes you to lose any unsaved data, so be sure to save often while you are developing your macros, and proofread them carefully before you submit them.

Solving Problems with Macro Functions

Some common causes of problems with macro functions include

- misspelling the function name

- □ omitting the opening or closing parenthesis
- □ omitting an argument or specifying an extra argument.

If you encounter an error related to a macro function, you may also see other error messages, generated by the invalid tokens left on the input stack by the macro processor.

Consider the following example. The user wants to use the %SUBSTR function to assign a portion of the value of the macro variable LINCOLN to the macro variable SECONDWD. But a typo exists in the second %LET statement, where %SUBSTR is misspelled as %SUBSRT:

```
%macro test;
%let lincoln=Four score and seven;
%let secondwd=%subsrt(&lincoln,6,5);   /* ERROR */
%put *** &secondwd ***;
%mend test;

%test
```

When the erroneous program is submitted, the following appears in the SAS log:

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

The error messages clearly point to the function name, which is misspelled.

Solving “Apparent Invocation of Macro Not Resolved” Problems

When a macro name is passed to the macro processor but the processor does not find a matching macro definition, it generates the following message:

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

This error could be caused by the misspelling of the name of a macro or a macro function, or it could be caused by an error in a macro definition that caused the macro to be compiled as a dummy macro. A *dummy macro* is a macro that the macro processor partially compiles but does not store.

Note: You receive this warning only if the MERROR system option is on. △

Solving the “Black Hole” Macro Problem

One error that is most frustrating to new users of the macro language is one that is not accompanied by any error message (and is therefore hard to solve if you are a novice). When the macro processor begins compiling a macro definition, it reads and compiles tokens until it finds a matching %MEND statement. If you omit a %MEND statement or cause it to be unrecognized by omitting a semicolon in the preceding statement, the macro processor does not stop compiling tokens. Every line of code you submit becomes part of the macro.

Resubmitting the macro definition and adding the %MEND statement does not correct the error. When you submit the corrected definition, the macro processor treats it as a nested definition in the original macro definition. The macro processor must find a matching %MEND statement to stop compilation.

Note: It is a good practice to use the %MEND statement with the macro name, so you can easily match %MACRO and %MEND statements. △

If you recognize that SAS is not processing submitted statements and you are not sure how to recover, submit %MEND statements one at a time until the following message appears in the SAS log:

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

Then recall the original erroneous macro definition, correct the error in the %MEND statement, and submit the definition for compilation.

There are other syntax errors that can create similar problems, such as unmatched quotation marks and unclosed parentheses. Often, one of these syntax errors leads to others. Consider the following example:

```
%macro rooms;
   /* other macro statements */
   %put **** %str(John's office) ****;   /* ERROR */
%mend rooms;

%rooms
```

When you submit these statements, the macro processor begins to compile the macro definition ROOMS. However, the single quotation mark in the %PUT statement is not marked by a percent sign. Therefore, during compilation the macro processor interprets the single quote as the beginning of a literal token. It does not recognize the closing parenthesis, the semicolon at the end of the statement, or the %MEND statement at the end of the macro definition.

To recover from this error, you must submit the following:

```
');
%mend;
```

If the above methods do not work, try submitting the following string:

```
*'; *"; *); */; %mend; run;
```

Continue submitting this string until the following message appears in the SAS log:

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

Obviously, it is easier to catch these types errors before they occur. You can avoid subtle syntax errors by carefully checking your macros before submitting them for compilation. Refer to “Developing Bug-free Macros” on page 108 for a syntax checklist.

Note: Another cause of unexplained and unexpected macro behavior is using a reserved word as the name of a macro variable or macro. For example, the SAS System reserves names starting with SYS; you should not create macros and macro variables with names beginning with SYS. Also, most host environments have reserved words too. For example, on PC-based platforms, the word CON is reserved for console input. Check Appendix 1, “Reserved Words in the Macro Facility,” for reserved SAS System keywords; check your SAS companion for host environment reserved words. △

Resolving Timing Issues

Many macro errors occur because a macro variable resolves at a different time than when the user intended or a macro statement executes at an unexpected time. A prime example of the importance of timing is when you use CALL SYMPUT to write a DATA step variable to a macro variable. You cannot use this macro variable in the same DATA step where it is defined; you can use it only in subsequent steps (that is, after the DATA step's RUN statement).

The key to forestalling timing errors is to understand how the macro processor works. In simplest terms, the two major steps are compilation and execution. The compilation step resolves all macro code to compiled code. Then the code is executed. Most timing errors occur because the user expects something to happen during compilation that doesn't actually occur until execution or, conversely, expects something to happen later but is actually executed right away.

Here are two examples to help you understand why the timing of compilation and execution can be important.

Example of a Macro Statement Executing Immediately

In the following program, the user intends to use the %LET statement and the SR_CIT variable to indicate whether a data set contains any data for senior citizens:

```
data senior;
   set census;
   if age > 65 then
   do;
      %let sr_cit = yes;  /* ERROR */
      output;
   end;
run;
```

However, the results differ from the user's expectations. The %LET statement is executed immediately, while the DATA step is only being compiled–*before* the data set is read. Therefore, the %LET statement executes regardless of the results of the IF condition. Even if the data set contains no observations where AGE is greater than 65, SR_CIT is always **yes**.

The solution is to set the macro variable's value by a means that is controlled by the IF logic and does not execute unless the IF statement is true. In this case, the user should use CALL SYMPUT, as in the following correct program:

```
%let sr_cit = no;
data senior;
   set census;
   if age > 65 then
   do;
      call symput ("sr_cit","yes");
     output;
   end;
run;
```

When this program is submitted, only if an observation is found with AGE greater than 65 is the value of SR_CIT set to **yes**. Note that the variable was initialized to **no**. It is generally a good idea to initialize your macro variables.

Macro Resolution Occurs During DATA Step Compilation

In the previous example, you learned you had to use CALL SYMPUT to conditionally assign a macro variable a value in a DATA step. So, you submit the following program:

```
%let sr_age = 0;
data senior;
   set census;
   if age > 65 then
   do;
      call symput("sr_age",age);
      put "This data set contains data about a person";
      put "who is &sr_age years old."; /* ERROR */
   end;
run;
```

If AGE was 67, you'd expect to see a log message like this one:

```
This data set contains data about a person
who is 67 years old.
```

However, no matter what AGE is, the following message is sent to the log:

```
This data set contains data about a person
who is 0 years old.
```

Why is this? Because when the DATA step is being compiled, &SR_AGE is sent to the macro facility for resolution, and the result is passed back *before* the DATA step executes. To achieve the desired result, submit this corrected program instead:

```
%let sr_age = 0;
data senior;
   set census;
   if age > 65 then
   do;
      call symput("sr_age",age);
      stop;
   end;
run;

data _null_;
   put "This data set contains data about a person";
   put "who is &sr_age years old.";
run;
```

Note: Use double quotation marks in statements like PUT, because macro variables do not resolve when enclosed in single quotation marks. △

Here is another example of erroneously referring to a macro variable in the same step that creates it:

```
data _null_;
   retain total 0;
   set mydata end=final;
   total=total+price;
   call symput("macvar",put(total,dollar14.2));
   if final then put "*** total=&macvar ***"; /* ERROR */
run;
```

Submitting these statements writes the following lines to the SAS log:

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

*** total=&macvar ***
```

As this DATA step is tokenized and compiled, the **&** causes the word scanner to trigger the macro processor, which looks for a MACVAR entry in a symbol table. Because such an entry does not exist, the macro processor generates the warning message. Because the tokens remain on the input stack, they are transferred to the DATA step compiler. During DATA step execution, the CALL SYMPUT statement creates the macro variable MACVAR and assigns a value to it. However, the text **&macvar** in the PUT statement occurs because the text has already been processed while the macro was being compiled. If you were to resubmit these statements, then the macro would appear to work correctly, but the value of MACVAR would reflect the value set during the previous execution of the DATA step. This can be misleading.

Remember that in general, the `%` and `&` trigger immediate execution or resolution during the compilation stage of the rest of your SAS code.

For more examples and explanation of how CALL SYMPUT creates macro variables, see “Creating Macro Variables with the CALL SYMPUT Routine” in Chapter 5.

Solving Problems with the Autocall Facility

The autocall facility is an efficient way of storing and using production (debugged) macros. When a call to an autocall macro produces an error, the cause is one of two things:

- an erroneous autocall library specification
- an invalid autocall macro definition.

If the error is the autocall library specification and the MERROR option is set, SAS can generate any or all of the following warnings:

```
WARNING: No logical assign for filename FILENAME.
WARNING: Source level autocall is not found or cannot be opened.
         Autocall has been suspended and OPTION NOMAUTOSOURCE has
         been set. To use the autocall facility again, set OPTION
         MAUTOSOURCE.
WARNING: Apparent invocation of macro MACRO-NAME not resolved.
```

If the error is in the autocall macro definition, SAS generates a message like the following:

```
NOTE: Line generated by the invoked macro "MACRO-NAME".
```

Fixing Autocall Library Specifications

When an autocall library specification causes an error, it is because the macro processor cannot find the member containing the autocall macro definition in the library or libraries specified in the SASAUTOS system option.

To correct this error, follow these steps.

1. If the unresolved macro call created an invalid SAS statement, submit a single semicolon to terminate the invalid statement. This enables the SAS System to correctly recognize subsequent statements.
2. Look at the value of the SASAUTOS system option by printing the output of the OPTIONS procedure or by viewing the OPTIONS window in the SAS Display Manager System. (Or, edit your SAS configuration file or SAS autoexec file.) Verify each fileref or directory name. If you find an error, submit a new OPTIONS statement or change the SASAUTOS setting in the OPTIONS window.
3. Check the MAUTOSOURCE system option. If SAS could not open at least one library, it sets the NOMAUTOSOUCE option. If NOMAUTOSOURCE is present, reset MAUTOSOURCE with a new OPTIONS statement or the OPTIONS window.
4. If the library specifications are correct, check the contents of each directory to verify that the autocall library member exists and that it contains a macro definition of the same name. If the member is missing, then add it.
5. Set the MRECALL option with a new OPTIONS statement or the OPTIONS window. By default, the macro processor only searches once for an undefined macro. Setting this option causes the macro processor to search the autocall libraries for the specification again.
6. Call the autocall macro. This includes and submits the autocall macro source.
7. Reset the NOMRECALL option.

Note: Some host environments have environment variables or system-level logical names assigned to the SASAUTOS library; check your SAS companion for more information on details about how the SASAUTOS library specification is handled in your host environment. △

Fixing Autocall Macro Definition Errors

When the autocall facility locates an autocall library member, the macro processor compiles any macros in that library member and stores the compiled macros in the catalog containing stored compiled macros. For the rest of your SAS session, invoking one of those macros retrieves the compiled macro from the WORK library. Under no circumstances does the autocall facility use an autocall library member when a compiled macro with the same name already exists. Thus, if you invoke an autocall macro and discover you made an error when you defined it, you must correct the autocall library member for future use and compile the corrected version directly in your program or session.

To correct an autocall macro definition in a display manager session, do the following:

1. Use the INCLUDE command to bring the autocall library member into the SAS PROGRAM EDITOR window. If the macro is stored in a catalog SOURCE entry, use the COPY command to bring the program into the PROGRAM EDITOR window.
2. Correct the error.
3. Store a copy of the corrected macro in the autocall library with the FILE command for a macro in an external file or with a SAVE command for a macro in a catalog entry.
4. Submit the macro definition from the PROGRAM EDITOR window.

The macro processor then compiles the corrected version, replacing the incorrect compiled macro. The corrected, compiled macro is now ready to execute at the next invocation.

To correct an autocall macro definition in an interactive line mode session, do the following:

1. Edit the autocall macro source with a text editor.
2. Correct the error.
3. Use a %INCLUDE statement to bring the corrected library member into your SAS session.

The macro processor then compiles the corrected version, replacing the incorrect compiled macro. The corrected, compiled macro is now ready to execute at the next invocation.

File and Macro Names for Autocall

When you want to use a macro as an autocall macro, you must store the macro in a file with the same name as the macro. Also, the file extension must be .SAS (if your operating system uses file extensions). If you experience problems with the autocall facility, be sure the macro and file names match and the file has the right extension when necessary.

Displaying Information about Stored Compiled Macros

To display the list of entries in a catalog containing compiled macros, you can use the display manager CATALOG window or the CATALOG procedure. The following PROC step displays the contents of a macro catalog in a SAS data library identified with the libref MYSASLIB:

```
libname mysaslib 'SAS-data-library';
   proc catalog catalog=mysaslib.sasmacr;
      contents;
   run;
   quit;
```

You can also use PROC CATALOG to display information about autocall library macros stored in SOURCE entries in a catalog. You cannot use PROC CATALOG or the CATALOG window to copy or rename stored compiled macros.

In Release 6.11 or later, you can use PROC SQL to retrieve information about all compiled macros. For example, submitting the following statements produces output similar to Output 10.1 on page 120:

```
proc sql;
   select * from dictionary.catalogs
        where memname in ('SASMACR');
```

Output 10.1 Output from PROC SQL

```
Library   Member    Member    Object    Object
Name      Name      Type      Name      Type
                                      Date      Object
Object Description                    Modified  Alias
------------------------------------------------------------
WORK      SASMACR   CATALOG   FINDAUTO  MACRO
                                        05/28/96

SASDATA   SASMACR   CATALOG   CLAUSE    MACRO
Count words in clause                   05/24/96

SASDATA   SASMACR   CATALOG   CMPRES    MACRO
CMPRES autocall macro                   05/24/96

SASDATA   SASMACR   CATALOG   DATATYP   MACRO
DATATYP autocall macro                  05/24/96

SASDATA   SASMACR   CATALOG   LEFT      MACRO
LEFT autocall macro                     05/24/96
```

To display information about compiled macros when you invoke them, use the SAS system options MLOGIC, MPRINT, and SYMBOLGEN. When you specify the SAS system option MLOGIC, the libref and date of compilation of a stored compiled macro are written to the log along with the usual information displayed during macro execution.

Solving Problems with Expression Evaluation

The following macro statements use an implicit %EVAL function:

%DO	%IF-%THEN	%SCAN
%DO %UNTIL	%QSCAN	%SYSEVALF
%DO %WHILE	%QSUBSTR	%SUBSTR

In addition, you can use the %EVAL function to perform an explicit expression evaluation.

The most common errors that occur while evaluating expressions are the presence of character operands where numeric operands are required or ambiguity about whether a token is a numeric operator or a character value. Chapter 6, “Macro Expressions,” discusses these and other macro expression errors.

Quite often, an error occurs when a special character or a keyword appears in a character string. Consider the following program:

```
%macro conjunct(word= );
   %if &word = and or &word = but or &word = or %then   /* ERROR */
      %do %put *** &word is a conjunction. ***;

   %else
      %do %put *** &word is not a conjunction. ***;
%mend conjunct;
```

In the %IF statement, the values of WORD being tested are ambiguous–they could also be interpreted as the numeric operators AND and OR. Therefore, the SAS System generates the following error messages in the log:

```
ERROR: A character operand was found in the %EVAL function or %IF
       condition where a numeric operand is required. The condition
       was:word = and or      &word = but or       &word = or
ERROR: The macro will stop executing.
```

To fix this problem, use the quoting functions %BQUOTE and %STR, as in the following corrected program:

```
%macro conjunct(word= );
   %if %bquote(&word) = %str(and) or %bquote(&word) = but or
           %bquote(&word) = %str(or) %then
      %do %put *** &word is a conjunction. ***;

   %else
      %do %put *** &word is not a conjunction. ***;
%mend conjunct;
```

In the corrected program, the %BQUOTE function quotes the result of the macro variable resolution (in case the user passes in a word containing an unmatched quotation mark or some other odd value), and the %STR function quotes the comparison values AND and OR at compile-time, so they are not ambiguous. You do not need to use %STR on the value BUT, because it is not ambiguous (not part of the SAS or macro language). See Chapter 7, “Macro Quoting,” for more information on using macro quoting functions.

Debugging Techniques

If you cannot identify your problem in “Troubleshooting Your Macros” on page 109, you can use the techniques described in this section to pinpoint the location of the error.

Using System Options to Track Problems

The SAS system options MLOGIC, MPRINT, and SYMBOLGEN can help you track the macro code and SAS code generated by your macro. Messages generated by these options appear in the SAS log, prefixed by the name of the option responsible for the message.

Note: Whenever you use the macro facility, use the following options: MACRO, MERROR, SERROR, and SOURCE. (While SOURCE is not a macro option, it is helpful to use this option when using the macro facility). In addition, if you are using autocall macros, use the MAUTOSOURCE option. See Table 13.8 for more system options associated with the macro facility. △

Although the following sections discuss each system option separately, you can, of course, combine them. However, each option can produce a significant amount of output, and too much information can be as confusing as too little. So, use only those options you think you might need and turn them off when you are done debugging.

Example of Tracing the Flow of Execution with MLOGIC

The MLOGIC system option traces the flow of execution of your macro, including the resolution of parameters, the scope of variables (global or local), the conditions of macro expressions being evaluated, the number of loop iterations, and the beginning and end of each macro execution. Use the MLOGIC option when you think a bug lies in the program logic (as opposed to simple syntax errors).

Note: MLOGIC can produce a lot of output, so use it only when necessary, and turn it off when debugging is finished. △

In the following example, the macro FIRST calls the macro SECOND to evaluate an expression:

```
%macro second(param);
   %let a = %eval(&param);a
%mend second;

%macro first(exp);
   %if (%second(&exp) ge 0) %then
      %put **** result >= 0 ****;
   %else
      %put **** result < 0 ****;
%mend first;

options mlogic;
%first(1+2)
```

Submitting this example with option MLOGIC shows when each macro starts execution, the values of passed parameters, and the result of the expression evaluation.

```
MLOGIC(FIRST):  Beginning execution.
MLOGIC(FIRST):  Parameter EXP has value 1+2
MLOGIC(SECOND):  Beginning execution.
MLOGIC(SECOND):  Parameter PARAM has value 1+2
MLOGIC(SECOND):  %LET (variable name is A)
MLOGIC(SECOND):  Ending execution.
MLOGIC(FIRST):  %IF condition (%second(&exp) ge 0) is TRUE
MLOGIC(FIRST):  %PUT **** result >= 0 ****
MLOGIC(FIRST):  Ending execution.
```

Example of Examining the Generated SAS Statements with MPRINT

The MPRINT system option writes to the SAS log each SAS statement generated by a macro. Use the MPRINT option when you suspect your bug lies in code that is generated in a manner you did not expect.

For example, the following program generates a simple DATA step:

```
%macro second(param);
   %let a = %eval(&param);a
%mend second;

%macro first(exp);
   data _null_;
      var=%second(&exp);
      put var=;
   run;
%mend first;

options mprint;
%first(1+2)
```

When you submit these statements with option MPRINT, these lines are written to the SAS log:

```
MPRINT(FIRST):   DATA _NULL_;
MPRINT(FIRST):   VAR=
MPRINT(SECOND):  3
MPRINT(FIRST):  ;
MPRINT(FIRST):   PUT VAR=;
MPRINT(FIRST):   RUN;

VAR=3
```

The MPRINT option shows you the generated text and identifies the macro that generated it.

Storing MPRINT Output in an External File

You can store text that is generated by the macro facility during macro execution in an external file. Printing the statements generated during macro execution to a file is useful for debugging macros when you want to test generated text in a later SAS session.

To use this feature, set both the MFILE and MPRINT system options on and also assign MPRINT as the fileref for the file to contain the output generated by the macro facility:

```
options mprint mfile;
filename mprint 'external-file';
```

The external file created by the MPRINT system option remains open until the SAS session terminates. The MPRINT text generated by the macro facility is written to the LOG window during the SAS session and to the external file when the session ends. The text consists of program statements generated during macro execution with macro variable references and macro expressions resolved. Only statements generated by the macro are stored in the external file. Any program statements outside the macro are not written to the external file. Each statement begins on a new line with one space separating words. The text is stored in the external file without the MPRINT(*macroname*): prefix, which is displayed in the LOG window.

If MPRINT is not assigned as a fileref or if the file cannot be accessed, warnings are written to the log and MFILE is turned off. To use the feature again, you must specify MFILE again.

By default, the MPRINT and MFILE options are off.

The following example uses the MPRINT and MFILE options to store generated text in the external file named TEMPOUT:

```
options mprint mfile;
filename mprint 'TEMPOUT';

%macro temp;
   data one;
      %do i=1 %to 3;
         x&i=&i;
      %end;
   run;
%mend temp;

%temp
```

The macro facility writes the following lines to the SAS log and creates the external file named TEMPOUT:

```
MPRINT(TEMP):   DATA ONE;
NOTE: The macro generated output from MPRINT will also be written
      to external file '/u/local/abcdef/TEMPOUT' while OPTIONS
      MPRINT and MFILE are set.
MPRINT(TEMP):   X1=1;
MPRINT(TEMP):   X2=2;
MPRINT(TEMP):   X3=3;
MPRINT(TEMP):   RUN;
```

When the SAS session ends, the file TEMPOUT contains:

```
DATA ONE;
X1=1;
X2=2;
X3=3;
RUN;
```

Note: Using MPRINT to write code to an external file is a debugging tool only–it should not be used to create SAS code files for purposes other than debugging. △

Examining Macro Variable Resolution with SYMBOLGEN

The SYMBOLGEN system option tells you what each macro variable resolves to by writing messages to the SAS log. This option is especially useful in spotting quoting problems, where the macro variable resolves to something other than what you intended because of a special character.

For example, suppose you submit the following statements:

```
options symbolgen;

%let a1=dog;
%let b2=cat;
%let b=1;
%let c=2;
%let d=a;
%let e=b;
%put **** &&&d&b ****;
%put **** &&&e&c ****;
```

The SYMBOLGEN option writes these lines to the SAS log:

```
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable D resolves to a
SYMBOLGEN:  Macro variable B resolves to 1
SYMBOLGEN:  Macro variable A1 resolves to dog
**** dog ****

SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable E resolves to b
SYMBOLGEN:  Macro variable C resolves to 2
SYMBOLGEN:  Macro variable B2 resolves to cat
**** cat ****
```

Reading the log provided by the SYMBOLGEN option is easier than examining the program statements to trace the indirect resolution. Notice that the SYMBOLGEN option traces each step of the macro variable resolution by the macro processor. When the resolution is complete, the %PUT statement writes the value to the SAS log.

When you use SYMBOLGEN to trace the values of macro variables that have been masked with a macro quoting function, you may see an additional message about the quoting being “stripped for printing.” For example, suppose you submit the following statements, with SYMBOLGEN set to on:

```
%let nickname = %bquote(My name’s O’Malley, but I’m called Bruce);
%put *** &nickname ***;
```

The SAS log contains the following after these statements have executed:

```
SYMBOLGEN:  Macro variable NICKNAME resolves to
                            My name’s O’Malley, but I’m called Bruce
SYMBOLGEN:  Some characters in the above value which were
                           subject to macro quoting have been
                           unquoted for printing.
*** My name’s O’Malley, but I’m called Bruce ***
```

You can ignore the unquoting message.

Using the %PUT Statement to Track Problems

Along with using the SYMBOLGEN system option to write the values of macro variables to the SAS log, you may find it useful to use the %PUT statement while developing and debugging your macros. When the macro is finished, you can delete or comment out the %PUT statements. Table 10.3 gives some occasions where you might find the %PUT statement helpful in debugging, and an example of each:

Table 10.3 Example %PUT Statements Useful when Debugging Macros

Situation	Example
show a macro variable's value	**%PUT ****&variable-name****;**
check leading or trailing blanks in a variable's value	**%PUT ***&variable-name***;**
check double-ampersand resolution, as during a loop	**%PUT ***variable-name&i = &&variable-name***;**
check evaluation of a condition	**%PUT ***This condition was met.***;**

As you recall, macro variables are stored in symbol tables. There is a global symbol table, which contains global macro variables, and a local symbol table, which contains local macro variables. During the debugging process, you may find it helpful on occasion to print these tables to examine the scope and values of a group of macro variables. To do so, use the %PUT statement with one of the following options:

ALL	describes all currently defined macro variables, regardless of scope. This includes user-generated global and local variables as well as automatic macro variables.
AUTOMATIC	describes all automatic macro variables. The scope is listed as AUTOMATIC. All automatic macro variables are global except SYSPBUFF.
GLOBAL	describes all user-generated global macro variables. The scope is listed as GLOBAL. Automatic macro variables are not listed.
LOCAL	describes user-generated local macro variables defined within the currently executing macro. The scope is listed as the name of the macro in which the macro variable is defined.
USER	describes all user-generated macro variables, regardless of scope. For global macro variables, the scope is GLOBAL; for local macro variables, the scope is the name of the macro.

The following example uses the %PUT statement with the argument _USER_ to examine the global and local variables available to the macro TOTINV. Notice the use of the user-generated macro variable TRACE to control when the %PUT statement writes values to the log.

```
%macro totinv(var);
   %global macvar;
   data inv;
      retain total 0;
      set sasuser.houses end=final;
      total=total+&var;
      if final then call symput("macvar",put(total,dollar14.2));
   run;

   %if &trace = ON  %then
      %do;
         %put *** Tracing macro scopes. ***;
         %put _USER_;
      %end;
%mend totinv;

%let trace=ON;
%totinv(price)
%put *** TOTAL=&macvar ***;
```

When you submit these statements, the first %PUT statement in the macro TOTINV writes the message about tracing being on and then writes the scope and value of all user generated macro variables to the SAS log.

```
*** Tracing macro scopes. ***
TOTINV VAR price
GLOBAL TRACE ON
GLOBAL MACVAR  $1,240,800.00
*** TOTAL= $1,240,800.00 ***
```

See Chapter 5 for a more detailed discussion of macro variable scope.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Macro Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 310 pages.

SAS Macro Language: Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

1-58025-522-1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999