PharmaSUG-2012 - Paper TA11-SAS

Why Does SAS® Say That? What Common DATA Step and Macro Messages Are Trying to Tell You

Charley Mullin and Kevin Russell, SAS Institute, Cary, NC, USA

ABSTRACT

SAS notes, warnings, and errors are written to the log to help SAS programmers understand what SAS is expecting to find. Some messages are for information, some signal potential problems, some require you to make changes in your SAS code, and some might seem obscure. This paper explores some of these notes, warnings, and errors that come from DATA step and macro programs. This paper deciphers them into easily understood explanations that enable you to answer many of your questions.

INTRODUCTION

Information messages in any programming language have to be precise about the condition found. Their goal is not to describe what caused the condition, but to convey accurate information about the current or potential problems. These technically accurate messages might, however, seem somewhat cryptic to new SAS programmers or those who are expanding their skills by trying new and more advanced tasks. The purpose of this paper is to interpret those messages that occur often and that might be a challenge to understand. The paper focuses on SAS® 9.3.

DATA step examples are presented first, followed by macro examples. For each section, we show the SAS code that produces the message, an explanation, and one or more ways to eliminate the message from the log.

Tips are also provided to help make the SAS log easier to understand, to explain why you see what you do in the SAS log, or to add emphasis.

DATA STEP EXAMPLES

DATA step is the primary programming language in Base SAS® software. It can be used for many tasks, including reading external files, analyzing and manipulating data, and combining SAS data sets.

This section presents DATA step examples grouped by type of processing: those that pertain to reading an external file, character and numeric variables, and arrays.

Some messages require you to take a different approach in your programming, add error traps, or process the data in a different manner. Some issues require an additional step to solve the problem or remove the messages from the SAS log.

TIP: In some cases, you can set an option to prevent the occurrence of messages that do not indicate a problem.

EXAMPLE 1: USING PROC IMPORT TO READ A DELIMITED EXTERNAL FILE

The IMPORT procedure (PROC), as shown below, is used to read delimited external files. The process that creates the external files frequently puts 1 or more blank lines at the top. The blank lines cause PROC IMPORT to stop processing.

```
PROC IMPORT DATAFILE="C:\data\test.dat" dbms=dlm out=data replace;

DELIMITER="|";
RUN;
```

As a result, the following message is likely to appear in your SAS log:

```
Unable to sample external file, no data in first 5 records.
NOTE: Import Cancelled.
NOTE: The SAS System stopped processing this step because of errors.
```

The problem might occur when you save a table or an entire page from an Excel file as a delimited external file.

What You Can Do

Here are 3 possible workarounds:

- One solution is to simply delete the blank rows from the text file.
- A second solution is to add the DATAROW= option to PROC IMPORT to indicate where the data starts.
- A third solution is to go back to the source file and save only the desired data.

EXAMPLE 2: READING DATA FROM A TEXT FILE

Invalid data in your input file might be unavoidable. The "Invalid data" message shows where the mismatched data appears in your file.

This example uses the following INPUT statement:

```
Input aaa $ bbb c d var5 var6;
```

The SAS log contains a note, but no errors or warnings. However, the note does indicate a problem—it shows "Invalid data" for a variable.

```
NOTE: Invalid data for BBB in line 26 6-15.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7
26        fred,flintstone,1,2,3,4 23
AAA=fred BBB=. C=1 D=2 VAR5=3 VAR6=4 _ERROR_=1 _N_=25
```

What Does This Note Mean?

Gleaning the useful information from the log can be a little tricky, but you can understand it if you take it a line at a time, starting with the first line:

```
NOTE: Invalid data for BBB in line 26 6-15.
```

This line identifies 3 key points:

- The variable whose data type did not match the data read is `BBB`
- The record number in the text file where the problem was found is `26`
- The starting and ending columns of the invalid data are `6-15`

The next line, the RULE line, helps you locate the invalid data. The digits in the line indicate column numbers by multiples of 10, so 1 is column 10, 2 is column 20, and so on. The plus signs mark the midway point between the numbers. The first + indicates column 5, the next + indicates column 15, then 25, and so on.

TIP: If you set the system option LineSize=110, the input records will be 100 characters on each line in the log. The ruler digits and plus signs line up with 100 bytes on each line of the log, making it easier to find columns beyond 100.

Another 3 points are shown in this third line of the log:

```
26        fred,flintstone,1,2,3,4 23
```

- The record number of the problem record is `26`
- The actual record is `fred,flintstone,1,2,3,4`
- The number of characters read from that record is `23`

The final line in the log is a list of the variables and the values of those variables after the current iteration of the DATA step ended:

```
AAA=fred BBB=. C=1 D=2 VAR5=3 VAR6=4 _ERROR_=1 _N_=25
```

TIP: When you modify variables after the INPUT statement has executed, the values shown have the modified data values, not the original data that was read from the input of the external file.

What You Can Do

In the scenario illustrated here, the first problem is that SAS is trying to read character data into a numeric variable. The solution depends on whether the data are correct or whether the data should be numeric. If the data should be character, change the variable type to match the data type. As shown here, add a character informat for variable BBB in the INPUT statement in a DATA step.

```
Input aaa $ bbb :$20. c d var5 var6;
```

If numeric is the correct type for this variable, and the data are incorrect, place 2 question marks after the variable name for which there are invalid data.

```
Input aaa $ bbb ?? c d var5 var6;
```

A second problem occurs when the informat specified for a variable does not match the data. An example of this would be if SAS tries to read the date value “24APR2012” with the informat MMDDYY10. The solution is to change to the correct informat, in this case, DATE9.

This problem can also occur when DATA step code is generated by PROC IMPORT. The PROC IMPORT option GUESSINGROWS specifies the number of rows of text that will be scanned in order to determine the type and length of all variables.

The GUESSINGROWS default value is stored in the SAS registry. To override the default value, add the GUESSINGROWS data source statement to the PROC IMPORT code. If you specify a value larger than the default, PROC IMPORT samples more records of the data to determine the variable type and maximum length. If character data for a variable appear only deep in the file being read, the result is more accurate than if a small number of rows had been sampled. The tradeoff is that sampling more rows requires more system resources.

EXAMPLE 3: APPLYING A NUMERIC FORMAT TO A CHARACTER VARIABLE

When the format differs from the data type of the variable it is assigned to, as shown here, then an “unknown format” error might occur.

```
Input var1 var2 var3 var4 $ var5 $;
Format var4 mmddyy8.;
```

At compile time, SAS reads through the DATA step program from left to right and from top to bottom. Statements such as LENGTH and RETAIN define variables and their data type. A FORMAT statement defines the appearance of the value when displayed. The first reference to a variable determines its type and length. When the first occurrence of a variable is in an INPUT statement, by default it becomes a numeric variable with a length of 8 bytes. If a dollar sign ($) follows the variable, the variable is defined as a character variable with a length of 8 bytes. If a variable is defined as character, and a subsequent FORMAT statement attempts to assign a numeric format to this variable, the ERROR message shown above is written to the SAS log.

```
2681  format var4 mmddyy8.;
                  --------
                  48
ERROR 48-59: The format $MMDDYY was not found or could not be loaded.
```

If you try to assign a numeric format to a character variable, SAS tries to correct the situation by adding a $ to the beginning of the format name and then searches for the new format. In this example, SAS searches for a $MMDDYY*w*. format to apply to the VAR4 variable. None is found, so the ERROR message above is written to the log. SAS also underscores the statement that is causing the error.

What You Can Do

The solution is to use character formats and informats for character variables and numeric formats and informats for numeric variables. All formats and informats for character variables begin with a $. Numeric formats and informats do not.

EXAMPLE 4: VARIABLES THAT HAVE MULTIPLE DEFINITIONS

If you have a single DATA step that contains 2 SAS data sets that have variables of the same name but which are defined differently, confusion about the data type might result. PROC CONTENTS shows that VAR1 is defined as character in data set WORK.ONE and as numeric in data set WORK.TWO.

```
Data Set Name        WORK.ONE
                        Alphabetic List of Variables and Attributes

                                #    Variable    Type    Len
                                1    var1        Char      1
                                2    var2        Num       8
Data Set Name        WORK.TWO
                        Alphabetic List of Variables and Attributes

                                #    Variable    Type    Len
                                1    var1        Num       8
                                2    var2        Num       8
```

Output 1. PROC CONTENTS Listing for Example 4 Data Sets

SAS code submitted:

```
Data three;
Merge one two;
By var1 var2;
Run;
```

When the DATA step program finds an input data set, the data set opens and the data set attributes (header) are read. The header contains information about each variable: type, length, informat, format, and label. Variables are read in the order in which they are stored in the data set and are added to the program data vector (PDV) in the same order.

SAS processes the data set variables in the sequence in which they are found in the header. As subsequent data sets are opened, each variable is examined to determine whether it is already in the PDV. If it is, SAS checks the variable's definition in the PDV against the variable definition in the data set header that was just opened.

If a variable is defined as character in one data set and as numeric in another data set, this message occurs:

```
ERROR: Variable VAR1 has been defined as both character and numeric.
```

What You Can Do

Look at the definitions of VAR1 on the data sets being read by the DATA step. The variable is character in WORK.ONE and numeric in WORK.TWO.

From the PROC CONTENTS output shown above, we see that the variable VAR1 is character in WORK.ONE and it is numeric in WORK.TWO.

The solution is to either change the character variable to numeric or change the numeric variable to character.

TIP: It is very common for character variables to have invalid numeric data stored in them. Often, the best choice is to convert the numeric variables to character to avoid data loss. When converting a numeric variable to character, make sure that the length of the newly created variables match the existing ones.

Converting a character variable to numeric or numeric to character can be done easily with a few lines of DATA step code and the INPUT or PUT function. If the variable being converted has nonnumeric data, SAS will print invalid data messages in the log. The double question marks (??) that preceded the INFORMAT parameter to the INPUT function suppress invalid data messages.

In this example, a character variable is converted to numeric with the INPUT function:

```
Data one;
Set one(rename=(var1=temp));
Var1=input(temp,??8.);
Drop temp;
Run;
```

To convert numeric values to character values, use the PUT function. The double question marks mentioned above are not needed because all numeric values are valid character data. Here is an example in which numeric values are changed to character values:

```
Data two;
Set two(rename=(var1=temp));
Var1=put(temp,8.);
Drop temp;
Run;
```

EXAMPLE 5: VARIABLES THAT HAVE MULTIPLE LENGTHS

Character variables that are defined on different SAS data sets cause warnings about the length of variables. PROC CONTENTS shows the variable VAR2 defined with different lengths on the input data sets.

```
Data Set Name        WORK.ONE

                        Alphabetic List of Variables and Attributes

                               #    Variable    Type    Len

                               1    var2        Char      3
                               2    var3        Char     20

Data Set Name        WORK.TWO

                        Alphabetic List of Variables and Attributes

                               #    Variable    Type    Len

                               1    var2        Char      7
                               2    var3        Char     50
                               3    var5        Char     35
```

Output 2. PROC CONTENTS Listing for Example 5 Data Sets

SAS code submitted:

```
Data three;
Merge dataset1 dataset2;
By var2;
Var4=cats(var3,var5);
Run;
```

Warnings appear for both the BY variable and a variable from the input data sets in the SAS log:

```
WARNING: Multiple lengths were specified for the BY variable VAR2 by input
data sets. This may cause unexpected results.
WARNING: Multiple lengths were specified for the variable VAR3 by input
data set(s). This may cause truncation of data.
```

Shown above are 2 different messages that say “Multiple lengths were specified...” One of the messages pertains to the BY variable, and the other message is for other variables not named in the BY statement.

The messages are triggered when the compiler finds a variable in a data set that is already in the PDV but the definition in the data set currently being processed has a longer length than the previously defined length. This discrepancy in length is the problem. If the occurrence of a character variable is 20 bytes, but in a subsequent data set the same variable is defined as 50 bytes, any data in bytes 21-50 will be lost when the second data set is read.

What You Can Do

There are several solutions to the problem for both cases.

- One solution is to change the order of the data sets in the program so that the data set with the longer variable definition is opened first.

  Change this:

  ```
  Merge dataset1 dataset2;
  By var2;
  ```

  To this:

  ```
  Merge dataset2 dataset1;
  By var2;
  ```

  Note: Depending on the reasons for combining more than 1 SAS data set, changing the sequence of the data sets in the MERGE statement might not be an option. If you are replacing the value of a variable in the first data set with a new value from the second data set, changing the sequence of the data sets does not accomplish that goal.

- A second solution is to add a LENGTH statement for the variable that immediately follows the DATA statement and before a SET, MERGE, UPDATE, or MODIFY statement. This has the additional effect of moving the variable named in the LENGTH statement to the beginning of the PDV.

  ```
  Data new;
  Length var2 $50;
  Merge dataset1 dataset2;
  By var2;
  ```

- A third solution is to disable the warning by changing the value of the system option VARLENCHK= NOWARN to turn off the message for variables that do not appear in the BY statement. This option has no effect on variables named in the BY statement.

  ```
  Options varlenchk=nowarn;
  ```

  TIP: Setting the VARLENCHK= NOWARN option does not correct the mismatch in length. It only prevents SAS from warning you about it. You can also set the option to ERROR, which causes the DATA step to stop.

EXAMPLE 6: INPUT TO CONCATENATION FUNCTIONS

Concatenating character variables into another variable of similar length, as shown here, might cause truncation.

```
Data final;
merge group1(rename=(results=r1))
      group2(rename=(results=r2))
      group3(rename=(results=r3));
results=cats(r1,r2,r3);
run;
```

The warning in the SAS log shows the total number of characters needed, based on the length of the input variables:

```
NOTE: Argument 1 to function CATS at line 2384 column 6 is invalid.
NOTE: Further warning from this call to CATS will be suppressed.
WARNING: In a call to the CATS function, the buffer allocated for the result
         was not long enough to contain the concatenation of all the arguments.
         The correct result would contain 1040 characters, but the actual result
         may either be truncated to 200 character(s) or be completely blank,
         depending on the calling environment. The following note indicates
         the left-most argument that caused truncation.
NOTE: Argument 1 to function CATS at line 2384 column 6 is invalid.
```

How can an argument to a concatenation function be invalid? After 5 iterations of the warning message, a note appears in the log saying “Further warnings from this function error will be suppressed.”

The warning is saying is that the combined length of the variables listed in the CATS function is longer than the variable to receive the combined result. This also tells you that all data beyond the length of the receiving variable is truncated. Other information, including a listing of all variables and their values on the current iteration of the DATA step, is interspersed with the messages. All of the concatenation functions—CAT, CATS, CATT, CATX—can generate these messages.

What You Can Do

The solution is to declare a character variable long enough to accept the combined length of the values being concatenated.

The warning contains this line:

```
The correct result would contain 1040 characters,
```

This message says that the combined length of the data would be 1040 characters. However, the CATS function was used in this example. The CATS function strips leading and trailing blanks, so this is the total combined length of the stripped data, not the combined length of the variables. To be safe, you need to make the new variable the length of the combined lengths of all contributing variables.

To determine the total length needed, you can use the CAT function, which does not strip leading and trailing spaces, run the program again. The warning message then has the total length of the combined variables. Use that value to add a LENGTH statement before the concatenation function to make the variable long enough to store all data from all contributing variables. Rerun the program including the LENGTH statement and the original concatenation function, as shown here:

```
Data final;
Length results $1040;
merge group1(rename=(results=r1))
      group2(rename=(results=r2))
      group3(rename=(results=r3));
results=cats(r1,r2,r3);
run;
```

EXAMPLE 7: ARGUMENTS TO THE SUBSTR FUNCTION

Using a STRIP or TRIM function to create a shorter temporary field can affect the arguments to the SUBSTR function.

In typical usage, the 3 arguments to the SUBSTR function are as follows:

- The name of the variable from which to extract characters
- The position in the variable to start extracting characters
- The number of characters to extract

Invalid Second Argument

The STRIP function creates a temporary variable that is only the length of the data.

The SUBSTR function, below, has only that number of bytes available.

```
B=substr(strip(a),11,5);
```

From the SAS log:

```
NOTE: Invalid second argument to function SUBSTR at line 3396 column 3.
```

The second argument to the SUBSTR function is larger than the total number of characters in the first argument of the SUBSTR function. A common cause is using the STRIP or TRIM function inside the SUBSTR function on a variable that contains a missing value.

In the case above, the data in variable A is not left-justified, so the STRIP function was used to move the data to column 1 so that the target data would be in a known location. Changing the STRIP or TRIM function to a LEFT function solves the problem. The STRIP and TRIM functions can return missing values to the SUBSTR function. The LEFT function always returns the same number of bytes that it read in to justify.

Invalid Third Argument

In the sample code below, an attempt is made to extract more data than is available.

```
Length chardate $9 charyear $4;
charyear=substr(chardate,6,9);
```

The incorrect parameter generates this note in the SAS log:

```
NOTE: Invalid third argument to function SUBSTR at line 3386 column 3.
```

The third argument to the SUBSTR function is commonly thought to be the ending column number for the characters to be extracted. In fact, the third argument is the length of the extraction wanted.

The programmer's intention is to extract the last 4 characters from the variable CHARDATE value. The variable is 9 characters long. The SUBSTR function starts at the sixth character and attempts to extract the next 9 characters. This number of characters goes beyond the end of the variable length. Because there are not enough characters available to satisfy the request for 9 characters, the "Invalid third argument" message is generated.

What You Can Do

The solution is to specify the number of characters to extract rather than the ending column.

EXAMPLE 8: AUTOMATIC VARIABLE TYPE CONVERSIONS

When SAS encounters an incompatible data type in an operation or expression, such as the following, the data type is converted automatically. This conversion might cause unexpected results.

```
Data one;
A='10';
A=a*1;
Run;
```

The following notes look harmless, but they might lead to unexpected missing values:

```
NOTE: Character values have been converted to numeric values at the places
      given by: (Line):(Column).
      3451:3
NOTE: Numeric values have been converted to character values at the places
      given by: (Line):(Column).
      3451:4
```

The data conversion messages occur if you attempt to use a character variable where a numeric variable is required and vice versa. A common reason for the message about a character being converted to numeric is the use of a character variable in a numeric expression. Another cause is using a SUBSTR or PUT function to write character data into a numeric variable.

The message about a numeric being converted to character occurs when you assign a numeric value to a character variable. A common reason for the message is attempting to convert a character variable to a numeric by adding 0 to it or by multiplying it by 1.

SAS tries to handle the data type mismatch by implicitly converting data during execution. The conversion messages indicate the potential for missing or incorrect results. Implicit data conversion (that is, conversion done automatically by SAS rather than by programming statements) consumes far more CPU resources than are consumed when using the correct data type or when adding an explicit conversion with INPUT or PUT functions.

If you assign a numeric variable to a character variable, SAS does the conversion with the BEST*n*. format where *n* is the length of the character variable. If the character variable is shorter than the number of significant digits in the numeric variable, the character variable contains the number in E notation with loss of significant digits. Here is an example:

```
3484  data _null_;
3485  length a $6;
3486  b=12345678;
3487  a=b;
3488  put _all_;
3489  run;
NOTE: Numeric values have been converted to character values at the places
      given by:(Line):(Column).
      3487:3
a=1.23E7 b=12345678 _ERROR_=0 _N_=1
```

What You Can Do

For information about how to convert a variable from one type to the other, see Example 4.

The data conversion messages provide valuable clues for finding the code that caused the problem.

TIP: The note above refers to log line number 3487 and column number 3 on that line. The first 6 columns in the log are not counted in determining the column number, allowing for log line numbers up to 6 digits. If the SAS statement that generates the conversion message starts at the left-most column of the editor, as in the code shown above, the column where the conversion occurs can be determined by counting the characters in the statement on the log line number specified in the message.

If your SAS program has indented lines, the number of columns skipped by indention also needs to be counted. You might want to go back to the program editor and count the columns on the referenced line.

EXAMPLE 9: UNINITIALIZED AND UNREFERENCED VARIABLES

When a variable that does not exist in the PDV is used to assign a value to another variable, unexpected missing values might result.

```
data one;
length abc $10;
abc=xyz;
keep def;
run;
This note is generated when a new variable is defined in a DATA step but is never
used on the left side of an assignment statement.
NOTE: Variable XYZ is uninitialized.
```

An example of this is when the only reference to a new variable is on the right side of an equal sign, or the variable is listed in a definition such as a LENGTH or FORMAT statement and never assigned a value. This message is not generated when the variable is read from a data set or from an external file. The most common cause of this problem is a misspelled variable name.

The following warning results from referencing a variable only in a DROP, KEEP, or RENAME statement or option. As with the uninitialized note, the most common reason for this warning is when a variable name is misspelled.

```
WARNING: The variable def in the DROP, KEEP, or RENAME list has never been
referenced.
```

There are situations in which referencing a nonexistent variable in a DROP, KEEP, or RENAME is expected. You might not want a warning to occur for the problem or you might want to stop the program execution. SAS has provided 2 system options to give you the ability to control the type of notification that is generated. The options are DKRICOND and DKROCOND. DKRICOND applies to reading data from a data set using the DROP, KEEP, or RENAME data set options. The option DKROCOND controls messages when writing to a SAS data set. Here are valid values:

- ERROR sets error flag and writes an error message to the SAS log
- WARNING writes a warning message to the SAS log
- NOWARNING does not write any message to the SAS log

DKROCOND applies to using a data set option on data set(s) named in the DATA statement or in a DROP, KEEP, or RENAME statement. DKROCOND applies to the statement version because the DROP, KEEP, or RENAME is applied to the output data set(s) named in the DATA statement.

EXAMPLE 10: USING A FUNCTION NAME AS AN ARRAY NAME

Using a function name supplied by SAS as an array name prevents of the use of the function.

```
3522  array abs(5);
```

The note below alerts you that you cannot use the SAS function in this DATA step:

```
NOTE: The array abs has the same name as a SAS-supplied or user-defined
      function.  Parentheses following this name are treated as array
      references and not function references.
```

There are more than 600 different functions in the Base SAS 9.3. More functions are added with each new release. At some point, you might choose a name for an array that matches the name of a function defined by SAS. The note is written to the SAS log to alert you that further references to that array/function name will reference the array, not the function. In the sample shown above, for the duration of the current DATA step, the ABS function is not available. User-written functions created with PROC FCMP are not examined for matching names.

EXAMPLE 11: MISSPELLED ARRAY REFERENCE

Different messages occur depending on which side of the assignment statement contains the array reference.

This is an example of a misspelled array reference being assigned a value:

```
array abd(5) (1,2,3,4,5);
abc(5)=5;
```

The following shows errors that result from the assignment statement not matching the name in the ARRAY statement:

```
ERROR: Undeclared array referenced: abc.
ERROR: Variable abc has not been declared as an array.
```

What You Can Do

When an array name is used on the left side of an assignment statement, the error above is caused by a misspelling in the assignment statement using the array name or in the definition of the array. The solution is to examine the SAS code and correct the error.

Here, a variable is being assigned a value from an array reference:

```
array abd(5) (1,2,3,4,5);
x=abc(5);
```

Shown below is an example of an error resulting from an assignment statement that does not match the name in the ARRAY statement:

```
3575  array abd(5) (1,2,3,4,5);
3576  x=abc(5);
        ---
        68
ERROR 68-185: The function ABC is unknown, or cannot be accessed.
```

When an array name is used on the right side of an assignment statement, the error is caused by a misspelling in the assignment statement or in the definition of the array. In this example, SAS attempts to find a function named ABC. When the function is not located, the error message is printed in the SAS log. As in the example shown above, the solution is to make the names agree by correcting the typographical error.

EXAMPLE 12: ARRAY SUBSCRIPT VALUES

Using a subscript value that is not within the defined elements in the array might require revising the program logic.

In the code below, a logical array named ABD is created with 5 numeric variables. The variables are ABD1–ABD5. The numbers that are valid inside the array's parentheses, known as subscripts or pointer values, are 1–5 for this array. Any subscript value <1 or >5 is not a valid pointer value for this array. This code makes reference to an array element that does not exist:

```
array abd(5) (1,2,3,4,5);
x=abd(6);
```

This error results from using a subscript value that is outside the values defined for the array:

```
ERROR: Array subscript out of range at line 3584 column 1.
```

What You Can Do

If you use a numeric variable inside the parentheses, its values must be between 1 and 5, inclusive. If the pointer is <1, >5, or missing, SAS generates the error message shown above.

In addition, SAS writes all variables and their values to the SAS log when this error occurs. To determine the value of the subscript when the error occurred, find the pointer variable in the list and compare it to the valid values. This will help you determine whether the array definition needs to be changed or the value of the pointer variable is incorrect or missing.

MACRO EXAMPLES

The SAS macro facility enables you to extend and customize your SAS code. It is a great tool for reducing the amount of text that you must enter to complete common tasks.

The macro facility has 2 components: The *macro processor* is the portion of SAS that does the work, and the *macro language* is the syntax that you use to communicate with the macro processor.

EXAMPLE 13: INCORRECT MACRO INVOCATION SYNTAX

One of the most common causes of errors is the typographical errors that we all make when programming in SAS. Typical mistakes include missing semicolons, missing or unmatched quotation marks, and misspelling of SAS keywords.

In the example below, there is a simple error.

```
%macro test(a=,b=,c=,);
%put &=a;
%put &=b;
%put &=c;
%mend;
%test((a=One,b=Two,c=Three);
```

Instead of 1 opening parenthesis, there are 2. This syntax causes the following error messages:

```
ERROR: The macro TEST is still executing and cannot be redefined.
ERROR: A dummy macro will be compiled.
```

SAS macro code above does not produce an error when submitted the first time, but it also does not produce any results. When submitting code does not produce results, your likely reaction is to resubmit the code. However, when code like that shown above is submitted multiple times, the error is generated.

In the sample program, the first left parenthesis is read. The second left parenthesis triggers the SAS macro facility to begin collecting the arguments for a positional parameter. When no macro variable name is found, SAS defines positional, not keyword parameters. The SAS macro facility continues to collect text as the value of a positional parameter argument until the matching right parenthesis for the second left parenthesis is found.

When the macro is submitted a second time, the keyword %MACRO is found first. This terminates the macro definition. SAS begins to compile the macro. The SAS macro facility checks to see whether a macro by that name is currently executing. If it is, an ERROR message is recorded in the SAS log and a “dummy” macro definition is compiled. The SAS macro facility continues collecting the argument list until the matching right parenthesis is found for the first left parenthesis.

What You Can Do

The majority of the time, this error message indicates a syntax error in the macro’s invocation. Even though the syntax error is simple, there is no conclusive way to recover from the error. The circumvention is to find the syntax error, correct it, save the code, and then resubmit the code in a new SAS session.

EXAMPLE 14: MASKING SPECIAL CHARACTERS

Macro quoting issues can be some of the most confusing aspects of the SAS macro language. The following example illustrates a very common problem and the solution.

```
ERROR: Macro function %substr has too many arguments.  The excess arguments will be
ignored.
```

This common error is produced for a variety of macro functions, including %SUBSTR, %SCAN, and %INDEX. The error occurs when the value of an argument to the function contains commas.

Here is an example:

```
%let x=a,b,c;
%let y=%substr(&x,1,1);
%put &=y;
```

After the code above is executed, the SAS log contains the following set of error messages:

```
14   %let x=a,b,c;
15   %let y=%substr(&x,1,1);
ERROR: Macro function %SUBSTR has too many arguments.  The excess arguments will
       be ignored.
ERROR: A character operand was found in the %EVAL function or %IF condition where
       a numeric operand is required. The condition was: b
ERROR: Argument 2 to macro function %SUBSTR is not a number.
ERROR: A character operand was found in the %EVAL function or %IF condition where
a numeric operand is required. The condition was: c
ERROR: Argument 3 to macro function %SUBSTR is not a number.
16   %put &=y;
Y=
```

In the case of the first error message, the %SUBSTR function expects 2 or 3 arguments, but in our example, when &X resolves, there are 4 commas, which cause the error message about too many arguments.

The second error message, “A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: b,” is generated because the second and third arguments to the %SUBSTR function are expressions. Whenever an expression is encountered in a macro statement, there is an implied %EVAL. When the perceived second argument is not a number, but actually the letter *b*, an error message is issued.

The third error message, “Argument <n> to the macro function %SUBSTR is not a number,” is generated when the %SUBSTR function expects the second and third arguments to be numeric. The second argument to the function indicates the starting position of the substring to be extracted**.** The optional third argument indicates the numbers of characters extracted in the substring**.**

What You Can Do

In this situation, the solution is to use the macro quoting function %BQUOTE to mask the commas in the resolved value of macro variable X so that they are seen as text by the macro processor. The %SUBSTR function now correctly recognizes the 2 commas that follow the &X as delimiters for the %SUBSTR function**.**

```
%let x=a,b,c;
%let y=%substr(%bquote(&x),1,1);
%put &=y;
```

EXAMPLE 15: AUTOCALL MACROS USED AS MACRO FUNCTIONS

Many macro functions are autocall macros supplied by SAS. As with all autocall macros, SAS must know where to locate them or warnings and errors can occur. Here is a list of these autocall macros:

- %CMPRES and %QCMPRES
- %COMPSTOR
- %DATATYP
- %KVERIFY
- %LEFT and %QLEFT
- %LOWCASE and %QLOWCASE
- %SYSRC
- %TRIM and %QTRIM
- %VERIFY

Here is an example that illustrates the problem:

```
options mautosource sasautos=('c:\mymacs');

%let x=%str(abc
%let y=%trim(&x);
```

The warning occurs because the SASAUTOS= system option has been redefined:

```
35   options mautosource sasautos=('c:\mymacs');
36
37   %let x=%str(abc   );
38   %let y=%trim(&x);
WARNING: Apparent invocation of macro TRIM not resolved.
```

What You Can Do

The SASAUTOS= system option is redefined without listing the automatically generated fileref SASAUTOS. The fileref SASAUTOS points to the SAS macro library that contains the SAS supplied autocall macros**.** To prevent this behavior, list the automatically generated fileref of SASAUTOS in the SASAUTOS system option.

The following is the modified SAS code that prevents the warning:

```
options mautosource sasautos=(sasautos,'c:\mymacs');

%let x=%str(abc   );
%let y=%trim(&x);
```

EXAMPLE 16: USING MACRO VARIABLES IN FILENAMES

Macro variables are the most commonly used component of the macro language. They can be used virtually anywhere within SAS to perform text substitution. The following is an example of the use of a macro variable, but with a syntax error:

```
%let year=2011;
proc import datafile="C:\class&year.csv"
  dbms=csv
  out=myclass
  replace;
  getnames=yes;
run;
```

This error occurs when the specified file does not exist:

```
ERROR: Physical file does not exist, C:\class2011csv.
ERROR: Import unsuccessful.  See SAS Log for details.
```

This PROC IMPORT code produces the error shown above. The problem is the single period following the macro variable reference. A single period after a macro variable reference is seen as a delimiter for the macro variable reference and is resolved with the macro variable**.** As shown, the SAS code above attempts to read in the file which has no file extension and does not exist.

```
datafile="C:\class2011csv"
```

What You Can Do

You need an additional period after the text resolved by the macro processor, so use 2 periods instead of 1. The first period is seen as the macro variable delimiter and the second period is seen as text. Here is the correct syntax.

```
%let year=2011;
proc import datafile="C:\class&year..csv"
  dbms=csv
  out=myclass
  replace;
  getnames=yes;
run;
```

EXAMPLE 17: REFERENCING STORED COMPILED MACROS

One way to permanently store a macro definition is to use the stored compiled macro facility. The stored compiled macro facility compiles and saves compiled macros in a SAS catalog named SASMACR. The SASMACR catalog is in a permanent SAS library. The macro compilation occurs only once as the macro is stored. The stored compiled macro facility is intended for large production jobs that are updated infrequently.

From the sample log:

```
ERROR: A LOCK IS NOT AVAILABLE FOR LIBREF.SASMACR.CATALOG.
```

What You Can Do

The SAS session locks the SASMACR catalog as soon as you create a stored compiled macro. If another user attempts to invoke a macro in the SASMACR catalog, the above error message is issued. For multiple users to have access to the macro, you need to allocate the catalog as read-only. When changing the catalog to read-only, make sure that none of the users accessing the catalog have open SAS sessions or batch jobs. If running SAS on Windows or UNIX platforms, the easiest way to set the catalog to read-only is to right-click the SASMACR catalog and select **Properties**. Look for **Attributes**, which is located at the bottom of the pop-up menu. Select the **Read-only** check box.

These operating system commands below can also be used to make the SASMACR catalog read-only:

```
WIN : ATTRIB +R SASMACR.SAS7BCAT
UNIX: chmod 444 sasmacr.sct01
z/OS : use RACF, DISP=SHR, or SHR for the SAS data library
```

EXAMPLE 18: INVOKING A MACRO WITH PARAMETERS

This is another error that is caused by commas in the text that are passed to the macro language. When a macro is invoked, a comma is processed as a delimiter between positional parameters. When more commas appear in the macro call than were listed in the %MACRO statement, the error message below is issued to the SAS log.

The value of VALS is x,y,z.

```
%macro test(vals);
  %put &=vals;
  %mend;
%test(x,y,z)
```

The error occurs when the number of perceived parameters does not match the %MACRO statement:

```
ERROR: More positional parameters found than defined.
```

What You Can Do

The commas in the parameter value cause the macro processor to interpret the text as 3 positional parameters. Only 1 parameter is defined. The solution to this problem is the same as when the argument to the macro function contained commas as in Example 14. Mask the commas using a macro quoting function. In this example, instead of using the execution time quoting function %BQUOTE, we need to use the compile time quoting function %STR.

TIP: When differentiating between using a compile time versus an execution time quoting function, follow this rule-of-thumb. If you can see the special characters in the code to be masked, then a compile time quoting function is needed. If you cannot see the special characters (for example, the characters are the result of the resolved value of a macro variable), then an execution time quoting function is needed.

Here is the corrected SAS macro code.

```
%macro test(vals);
  %put &=vals;
%mend;

%test(%str(x,y,z))
```

CONCLUSION

As we have now seen, technically accurate messages can be somewhat challenging to follow. But a methodological approach can help. When looking through the SAS log, use these 2 tasks: First, determine which messages are significant. Second, determine the meaning of the significant messages.

Obviously, errors are significant. Warnings might or might not be important when taken in context. Notes do not directly signal a problem, but do contain information related to an assumption by the SAS System that can be critical to obtaining correct results from your program. Remember, not all problems are printed in red.

We hope that you now have a better understanding of common SAS messages and that you find it easier to determine which messages are important and which ones are simply informational.

We invite you to contact SAS Technical Support with specific questions.

When seeking assistance from SAS Technical Support, please send SAS code that you submitted and the full SAS log including the exact message. Be prepared to send both to us as plain text attachments in an e-mail message to support@sas.com.

REFERENCES

Wilson, Kim. “The Top 10 Head Scratchers: SAS® Log Messages That Prompt a Call to SAS Technical Support.” Proceedings of the SAS Global 2011 Conference. Cary, NC: SAS Institute, Inc. Available from http://support.sas.com/resources/papers/proceedings11/262-2011.pdf

RECOMMENDED READING

- SAS Institute, Inc. SAS® 9.3 Macro Language: Reference
  http://support.sas.com/documentation/cdl/en/mcrolref/62978/HTML/default/viewer.htm#titlepage.htm
- SAS Institute, Inc. Base SAS® 9.3 Procedures Guide
  http://support.sas.com/documentation/cdl/en/proc/63079/HTML/default/viewer.htm#titlepage.htm
- SAS Institute, Inc. SAS® 9.3 Formats and Informats: Reference
  http://support.sas.com/documentation/cdl/en/leforinforref/63324/HTML/default/viewer.htm#titlepage.htm
- SAS Institute, Inc. SAS® 9.3 Functions and CALL Routines: Reference
  http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#titlepage.htm
- SAS Institute, Inc. SAS® 9.3 Statements: Reference
  http://support.sas.com/documentation/cdl/en/lestmtsref/63323/HTML/default/viewer.htm#titlepage.htm

ACKNOWLEDGMENTS

We would like to thank Jan Squillace, Russ Tyndall, and Grace Whiteis. We could not have written this paper without their support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact information for the authors is as follows:

Charley Mullin
SAS Institute Inc.
SAS Campus Dr
Cary, NC 27513
919-677-8008
E-mail: support@sas.com
Web: support.sas.com

Kevin Russell
SAS Institute Inc.
SAS Campus Dr
Cary, NC 27513
919-677-8008
E-mail: support@sas.com
Web: support.sas.com