SCSUG2011-S02

The Top 10 Head-Scratchers: SAS® Log Messages That Prompt a Call to SAS Technical Support

Kim Wilson, SAS Institute Inc., Cary, NC, USA

ABSTRACT

As a DATA step programmer, you know the sigh of relief that comes when your job finishes and your SAS log is clear of any errors or warnings. When these messages do occur, most of the time they are intuitive enough so that you can move directly to the offending code, make the necessary changes, and complete the job successfully. However, what do you do about those perplexing messages that sometimes appear, making you scratch your head in puzzlement?

This paper examines the top 10 errors, notes, and warnings that prompt DATA step programmers to call SAS Technical Support. The discussion addresses the common causes of the messages and provides solutions so that you can reduce your troubleshooting time and effort.

INTRODUCTION

The DATA step is a powerful method for creating and manipulating SAS data sets. Customers with different backgrounds and levels of programming experience use the DATA step to reshape data, summarize data within BY groups, and perform many other logic-related tasks. Sometimes, though, errors, notes, and warnings can occur in your SAS log even if your programs seem syntactically correct. Often, the messages are intuitive and you know immediately how to fix the problem to obtain a clean log. However, in some instances, it might not be clear how to troubleshoot the message.

This paper describes some of the most common log messages that have been reported to the DATA step team within the Technical Support Division of SAS. Most of these messages are formally documented in SAS notes, which are searchable on the SAS Customer Support Web site (support.sas.com). This paper groups the top 10 head-scratchers by errors, notes, and warnings. Each message is described, often with code examples. Tips for avoiding the messages are also offered.

Hopefully, understanding these common pitfalls can help you write code that produces clean logs more often.

ERRORS

The errors that are described in this section are some of the most common errors that prompt a call to the DATA step group within Technical Support. The following scenarios provide additional information about analyzing programming issues and fixing potentially damaged data sets.

ERROR: AN INTERNAL ERROR HAS OCCURRED WHILE READING A COMPRESSED FILE. PLEASE CALL YOUR SAS SITE REPRESENTATIVE AND REPORT THE FOLLOWING…

You might encounter this error when you are reading a compressed SAS data set. This error often indicates that the compressed data set you are reading is corrupt. This error can result from a system failure that occurs while the data set is being written, from network errors, or other issues related to I/O.

You can attempt to repair the corrupt data set by running the DATASETS procedure with the REPAIR statement as shown in this example:

```
proc datasets library=your_lib;
   repair a;
quit;
```

This code might repair the damage, but not in all instances. For example, if the data set resides on a network drive, problems with the network card, cable, or driver could be the culprit. In this case, move the data set to a local drive and reference the data set in a SET statement to see whether it can be read without errors. If errors still occur, run the preceding PROC DATASETS code again. If this fails, restore the data set from your backup copy, if one exists. If one does not exist, then you need to re-create the data set.

ERROR: ARRAY SUBSCRIPT OUT OF RANGE AT LINE *N* AND COLUMN *N*.

An *array* is a temporary grouping of variables that are arranged in a particular order and identified by an array name. Using an array can save many lines of code because one action can easily be performed on multiple variables. Variables in an array can be referenced by their variable names or by the array name (*n*), where *n* is the location of the variable in the array.

When an array reference is made to a nonexistent variable, this error occurs. The following code shows how the error can occur:

```
data b;
   set a;
   array test(3) x y z;
   do i=1 to 3;
     if test(i) > test(i+1) then newvar='yes';
   end;

run;
```

In this sample code, the array contains three variables. The IF statement compares the current array variable to the adjacent variable. When I=3, the IF statement tries to compare Z to the fourth variable in the array, which does not exist, and an error occurs. This error message can surface in many different scenarios, but it always relates to the array variables. After the error message occurs, SAS transfers all of the variable data to the log, which helps tremendously with debugging. Here is the log for the previous sample code:

```
x=1 y=2 z=3 i=3 newvar=  _ERROR_=1 _N_=1
```

As you begin debugging, the value for the automatic variable _N_ tells you that the error occurred during the first iteration of the DATA step. The variable I has a value of **3**, so you know that there was a problem on the third instance of the DO loop. If you add these values to the IF statement, you see that a fourth variable does not exist.

To resolve this error, modify your code to reference only the variables in your array. In this example, use a value of **2** as the upper boundary of the DO loop, as shown below:

```
do i=1 to 2;
```

ERROR 48-59: THE FORMAT $*NAME* WAS NOT FOUND OR COULD NOT BE LOADED.

This error occurs when you are attempting to format a character variable with a numeric format. Consider the following code:

```
data a;
   x='12345';
   format x mmddyy10.;
run;
```

The variable X was created as a character variable but was given a numeric format. The compiler assumed that you intended to use the format specified with a $ prepended to it. This results in an unknown format name. The problem is easy to correct: Just remove the quotation marks (') from the value for X, and the problem is resolved.

This error can also occur if the variable type is changed by another statement in the program. Consider this example:

```
data a;
   x='12345';
   y=input(x,$5.);
   format y mmddyy10.;
run;
```

In this sample code, Y was created with the INPUT function, which is used to convert a character variable to a numeric variable. However, the informat that is specified determines whether the result is character or numeric. In this case, a numeric informat should have been supplied because it tells SAS how to read the variable that is listed as the first argument to the function. Because the informat is a character informat, the variable Y is created as a character variable. The numeric format is specified for Y, which results in this error message:

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

You can also receive a similar message if you specify a user-created format and SAS is not aware of where that format is stored. If you receive a SAS data set and a format catalog containing custom formats that apply to variables therein, you must store the catalog in a location and then notify SAS of that location. You specify the location by issuing a LIBNAME statement with the reserved libref of LIBRARY. SAS knows to look in that location when a custom format name is called. You might use a different libref, but you must specify that name in the FMTSEARCH= system option. If you fail to issue the LIBNAME statement, you receive this error message. The custom format name, in this case MYDATE, is shown in this sample SAS log:

```
data b;
   set a;
   format x mydate.;
                -------
           48
ERROR 48-59: The format MYDATE was not found or could not be loaded.
run;
```

NOTES

Notes in the log often mean that something is not correct programmatically. You cannot always tell what code changes you need to make simply by reading the note in the log. This section presents some of the most common notes that require extra thought in order to make the necessary changes to the code.

NOTE: THE MEANING OF AN IDENTIFIER AFTER A QUOTED STRING MAY CHANGE IN A FUTURE SAS RELEASE. INSERTING WHITE SPACE BETWEEN A QUOTED STRING AND THE SUCCEEDING IDENTIFIER IS RECOMMENDED.

Beginning in SAS®9, you might receive this note when you place an alphabetic character immediately after a closing single (') or double (") quotation mark.

SAS *constants*, which are also called *literals*, are number or character strings inside quotation marks and followed by a lowercase letter that indicates the type of constant. For example, a character hexadecimal constant is a string of an even number of hexadecimal characters enclosed in single or double quotation marks, followed immediately by an X (that is, '20'x is the hexadecimal representation for a space on an ASCII machine). If you want SAS to convert a constant for a specific date and time to a SAS datetime value, then you should enclose the constant in quotation marks followed by DT, as shown in the following example:

```
'01jan2011:12:30:22'dt
```

As a result, SAS converts this datetime value to a SAS datetime of 1609504222.

This note informs the programmer that a letter was specified immediately following a closing quotation mark, but the letter is not currently a SAS literal.

To remove the note, find the offending character that follows the closing quotation mark and add a space between the quotation mark and the character, as shown in this example:

```
data a;
   x=99;
   put 'this'a x;
run;
```

NOTE: INVALID ARGUMENT TO FUNCTION INPUT AT LINE *N* COLUMN *N*.

When you want to convert a character variable to a numeric variable, use the INPUT function to create a new variable. SAS uses the specified informat, which is supplied as the second argument to the function, as the instruction to read the character value and make the conversion. When some portion of the value is not valid for the given informat, this note is written to the log.

Consider this example:

```
data a;
   x='13/01/2011';
   y=input(x,mmddyy10.);
run;
```

In this example, the informat MMDDYY10**.** describes the character value that will be converted to a numeric value. A value for month cannot be greater than 12, so this note is generated. The same note also occurs if the day value exceeds the maximum number of days for the specified month.

If the character variable contains any character that cannot be converted to a numeric value, the same note is generated. Often, a character variable contains unprintable characters that you cannot see when you look at the value. In the following example, variable X is created by concatenating a tab character, the literal string '123', and what is often referred to as a *null character*, which is '00' in hexadecimal representation. The null character is not the same as a blank, which is '20'x.

```
data a;
  x='09'x || '123' || '00'x;

run;
```

This is how the value for X appears when it is written to the log with a PUT statement:

```
put x=;

x=123
```

As you can see, the value appears to be a numeric value of 123. In order to convert this variable to numeric, use the INPUT function with an appropriate informat that describes how to read the character value. For example, if other observations contain values up to five digits, then you use the informat of 5**.** as shown below:

```
y=input(x,5.);
```

Here is the relevant log message:

```
NOTE: Invalid argument to function INPUT at line 900 column 3.
x=   123  y=. _ERROR_=1 _N_=1
```

You cannot see any other characters besides '123', so you need to more closely examine the value by writing the value of X using the PUT statement and the $HEXw**.** format. This special format shows the hexadecimal representation of the value, as shown below:

```
put x $hex10.;
0931323300
```

When SAS writes a hexadecimal representation, each set of two characters represents one character in the ASCII or EBCDIC collating sequence. In the sample output, '09'x is a tab, '31'x is a 1, '32'x is a 2, '33'x is a 3, and '00'x is a null character. Because the tab and null characters cannot be stored in a numeric variable, the note that is output to the log is correct. You need to decide whether to read the variable as a character, possibly specify a delimiter in the INFILE statement that you were not aware of until now, or make other adjustments as needed in order to read the file.

NOTE: MERGE STATEMENT HAS MORE THAN ONE DATA SET WITH REPEATS OF BY VALUES.

When SAS merges data sets and more than one of them has duplicate observations for the same BY value, this courtesy note is issued to alert you of possible unexpected results.

When a MERGE statement executes, the observations from each data set are read into the Program Data Vector (PDV) in the order in which the data sets are listed in the MERGE statement.

Consider these sample data sets:

```
data a;
   input x y z;
datalines;
1 2 9
1 3 99
1 4 999
;
run;
```

```
data b;
   input x y zz;
datalines;
1 20 100
1 30 200
;
run;

data final;
   merge a b;
   by x;
run;
proc print;
run;
```

Output 1shows the result from the merge.

Obs	x	y	z	zz
1	1	20	9	100
2	1	30	99	200
3	1	4	999	200

Output 1. Output from Merging Data Sets A and B

Some SAS programmers believe that the values from the last data set that is listed in the MERGE statement always overwrite the common variable values that are supplied by other data sets. This does occur when the number of observations for a BY group is the same in all data sets. When that number is unbalanced, the values from the extra observations are written to the output data set because this was the last data set to populate the PDV.

In the preceding output, the Y value for the third observation came from data set A because, after the values from data set A were placed into the PDV, there were no observations from data set B to contribute to the merged output. Those variables that are unique to a given data set retain their values until the last observation in the BY group processes.

NOTE: SAS WENT TO A NEW LINE WHEN INPUT STATEMENT REACHED PAST THE END OF A LINE.

When SAS reads an external file, options that are specified in the INFILE statement control whether the INPUT statement tries to read past the end of the current input data record. FLOWOVER is the default behavior of the INPUT statement. This example demonstrates this option:

```
   /* Contents of the input file. */
a
ab
abc
```

This file is read with the following DATA step:

```
data a;
   infile 'path to file';
   input x $3.;
run;
```

When the file is read, the preceding note is output to the log. Because the first record did not have 3 bytes of data, the pointer flowed over to the second record and output the data therein. Then the pointer read the third record and output it to the data set. Output 2 shows output from the PRINT procedure.

Obs	x
1	ab
2	abc

Output 2. Output from PROC PRINT

The MISSOVER and TRUNCOVER options do not enable the input pointer to move to the next record when the INPUT statement does not find all of the data that it expects. The SCANOVER and STOPOVER options are also valid but are not used nearly as often.

If you add the TRUNCOVER option to the INFILE statement in the preceding code, all three records are output correctly in the output.

NOTE: INVALID DATA FOR *VARIABLE-NAME* AT LINE *N*

Invalid data messages are prevalent because some SAS programmers move data into and out of SAS and frequently manipulate data within SAS data sets. This note occurs when informats that are used to read data encounter values that do not match what is expected, columns are read beyond the target data, unprintable characters are embedded in data, and many other instances. It is not feasible to mention and describe each scenario that produces this note, but here are some common scenarios:

- You attempt to read a character value with a numeric informat, for example:

```
data a;
   infile datalines;
   input x 3.;
datalines;
abc
;
run;
```

  In addition to the note that is shown in the log, you will see the value that the informat was trying to read into the variable that is specified in the INPUT statement. You will see that X is a numeric variable but the value that is being read contains non-numeric data.

- You specify an informat that is incongruent with the data, for example:

```
data a;
   infile datalines;
   input x mmddyy10.;
datalines;
2011/12/01
;
run;
```

  In this example, the informat MMDDYY10**.** expects a month value between **1** and **12** to be the first part of the value.

  When you examine the value more closely in the log, you see that the correct informat to use is YYMMDD10**.**, for example:

```
input x yymmdd10.;
```

- You read data past the intended columns, for example:

```
data a;
   infile datalines dlm=',' truncover;
   input x mmddyy10. y;
datalines;
12/1/2011,100
;
run;
```

  In this example, the informat MMDDYY10**.** indicates that 10 bytes of data should be read for variable X. The date value consists of 9 bytes and the comma delimiter is the 10th byte. This comma is not valid for a date and causes this note to display.

  In order to read varying length fields in a delimited file, use the colon format modifier before each informat. This tells the input pointer to read a maximum number of bytes as designated by the informat width but to stop reading the value when a delimiter is read. The newly improved INPUT statement looks like this:

```
input x :mmddyy10. y;
```

- You embed unprintable characters in the data.

  If you are given a variable length file that was created on a PC, the characters that mark the end of each record are a carriage return and line-feed combination, or '0D0A' in hexadecimal representation. If you read the file using the default record format of 'variable', SAS parses the records appropriately. If you mistakenly specify the INFILE options of RECFM=F (fixed-width records) and LRECL= *fixed-length*, you will at some point read the '0D0A'x characters into a variable. If the variable that contains the unprintable characters is typed as character, you will not see this note in the log. However, if the variable type is numeric, you will see the note. You will also see a data transfer that contains the hexadecimal data values. Here is an example:

```
NOTE: Invalid data for a in line 1 1-20.
RULE:     ----+----1----+----2----+----3----+----4----+----5
1   CHAR  11111 222232..111 33
    ZONE  33333233333300333233
    NUMR  111110222232DA111033
```

  This data transfer should be read one column at a time, from top to bottom. The top line represents the characters as they are on the keyboard. The two characters underneath each of these characters are the underlying equivalent hexadecimal representation. For example, column 1, highlighted in yellow, indicates that a '1' is located in column 1 of the first record and the hexadecimal representation is '31'x. Look at columns 13 and 14, which are highlighted in blue, where the '0D0A'x is located. Notice that there are periods on the top row. The periods indicate that the hexadecimal representation contained in that column is for an unprintable character, thereby producing no known keyboard character. Seeing the end-of-record markers alerts you that you probably have a standard variable record file.

  In order to view the contents of one or more records in an external file without having to read the data into a variable and write it out with a PUT statement, you can use the INPUT and LIST statements as shown in this example:

```
data _null_;
   infile 'path to your file' obs=1;
   input;
   list;
run;
```

  If the external file that you are reading has records that are longer than 256 bytes, add the LRECL= option to the INFILE statement. In addition, if you want to limit the number of records that you read, also add the OBS= option. The null INPUT statement reads one record into the input buffer and the LIST statement writes it to the log with a ruled line. If the data contains any unprintable characters, the hexadecimal representation is output as shown in to the preceding example. This is very useful for debugging input issues when the content of the data values is uncertain.

- You are reading a Unicode file.

  If each of the bytes of your file has a '00'x embedded between them, then you are reading a Unicode file. You can specify ENCODING=UNICODE in the INFILE or FILENAME statement. This should resolve this issue, assuming that you are otherwise reading the data correctly.

WARNINGS

The warnings that are discussed in this section provide important information about issues with strings and variables. New system options enable the programmer to better control the types of messages that are produced in the log. Where applicable, these system options are discussed.

WARNING 32-169: THE QUOTED STRING CURRENTLY BEING PROCESSED HAS BECOME MORE THAN 262 CHARACTERS LONG. YOU MAY HAVE UNBALANCED QUOTATION MARKS.

Beginning in SAS 7, the maximum length of a character variable is 32,767 bytes. This warning is copied to the log if your SAS code has a quoted string that exceeds 262 characters in length. This is simply a warning; the data integrity is not compromised and return codes are not set as a result of this warning. This warning was implemented to comply with an operating system's filename restriction.

Beginning in SAS®9, you can specify the NOQUOTELENMAX system option to suppress this warning. This option is valid in a configuration file, at SAS invocation, in an OPTIONS statement, or in a SAS System Options window.

WARNING: MULTIPLE LENGTHS WERE SPECIFIED FOR THE VARIABLE *VARIABLE NAME* BY INPUT DATA SET(S). THIS MAY CAUSE TRUNCATION OF DATA.

A character variable's length is established in the PDV the first time the variable is encountered in the program. When you need to change the length of a variable in an input data set, a common practice is to use a LENGTH statement prior to the SET, MERGE, UPDATE, or MODIFY statement. In the following sample code, the length is changed with the LENGTH statement and the CONTENTS procedure verifies that variable X has a length of 4 in data set B.

```
data a;
   length x $5;
   x='z';
run;

data b;
   length x $4;
   set a;
run;
```

This warning alerts you that different lengths have been specified for a variable. This can be helpful if you were not aware that this occurred; also, it prevents data values from being truncated. Beginning in SAS 9.2, you can control the type of message that is written to the SAS log by specifying the VARLENCHK= system option. Here are valid settings for this option:

- NOWARN
- WARN
- ERROR

You can read more about this system option in the *SAS® 9.2 Language Reference: Dictionary* (SAS Institute Inc. 2011).

By design, this warning message excludes BY variables. When BY variables from different data sets are defined with varying lengths, the resulting warning message is written to the log:

```
WARNING: Multiple lengths were specified for the BY variable x by input data sets. This might
cause unexpected results.
```

In this situation, you need to determine the length of each BY variable in each data set that is listed in the SET, MERGE, UPDATE, or MODIFY statement. If you are not sure how long the variable length should be in order to avoid values being truncated, run PROC PRINT on each data set to examine the values. After you determine the desired length for the BY variable, re-create the data set that contains the shorter-length variable by placing a LENGTH statement before the SET statement.

CONCLUSION

The DATA step is a very powerful and integral part of Base SAS® software. The uses for the DATA step are endless and you are able to accomplish much within a few lines of code. Whether you have been using SAS for many years or are just getting started with this language, you have probably encountered some messages in the log that required changes to the syntax.

The errors, notes, and warnings that are discussed in this paper have been the subject of many calls from SAS customers over the past few years. Hopefully, presenting the discussion about them in one document can serve as a resource to you as you continue using the DATA step.

RESOURCES

SAS Institute Inc. 2010. *SAS® 9.2 Language Reference: Concepts, Second Edition*. Available at support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#titlepage.htm.

SAS Institute Inc. 2010. *SAS® 9.2 Language Reference: Dictionary, Fourth Edition*. Available at support.sas.com/documentation/cdl/en/lrdict/63026/HTML/default/viewer.htm#titlepage.htm.

SAS Institute Inc. 2010. SAS Note 37102. "The VARLENCHK= system option excludes BY variables." Cary, NC: SAS Institute Inc. Available at support.sas.com/kb/37/102.html.

SAS Institute Inc. 2003. SAS Note 3353. "SAS literals in PUT statement generate errors." Cary, NC: SAS Institute Inc. Available at support.sas.com/kb/3/353.html.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kim Wilson
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: support@sas.com
Web: support.sas.com