Paper # IN04

What's That Note/Warning/Error Mean and How Do I Fix It?

Deb Cassidy

Abstract
The SAS System provides lots of notes, warnings and error messages in the log. Some messages are very clear but others can leave you wondering why it appeared. In most cases, you have to resolve the error message to get your program to run. However, your program can run with notes and warnings. What many new users - and even a few experienced users - may not realize is that they could be getting wrong results. This presentation will cover some common notes, warnings and error messages by showing examples of what could have caused the message. You'll also learn some ways to help you determine if you are getting wrong results.

A Simple Example

The following code shows a simple error and some helpful notes.

```
data one;
   item1=4;
   item1=5;
run;

proc print data=one;
   var item1 item2 item3;
run;
```

The log shows:

```
1    data one;
2       item1=4;
3       item1=5;
4    run;

NOTE: The data set WORK.ONE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.24 seconds
      cpu time            0.04 seconds

5
6    proc print data=one;
7       var Item1 Item2 Item3;
ERROR: Variable ITEM2 not found.
ERROR: Variable ITEM3 not found.
8    run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.18 seconds
      cpu time            0.00 seconds
```

The notes after the first data step tell you the data set has been created. This may look like a note that can be ignored but if you look at the code you'll notice you should have 2 observations but file ONE only has 1. Why? You didn't include an output statement after each record you were trying to create. If you usually read in existing data sets then you may have found the implied output statement sufficient.

However, when creating your own data using statements, you need to provide the output statement or only one record will be output.

This was a very simple example but the number of observations should always be checked to see if it makes sense given your data. The number of variables can also vary from what you expected. This could be a sign that you didn't create something correctly or that you are bringing along extra variables that are no longer needed. The latter case can have a major impact on how long a program runs and can be a clue to writing a more efficient program.

The error messages after the PROC PRINT will actually stop the program. Most errors but not all will stop the program but the rest of the code will usually be checked for syntax errors. There are some differences in error handling on different operating systems.

These errors are very easy to understand. It is obvious that you are requesting a variable that doesn't exist. Of course, sometimes you have a typo that may not be obvious when you try to fix it. You may think you have the variable in the data set only to discover than you incorrectly named it much earlier in the program.

SAS Makes Assumptions

SAS makes some assumptions about errors for you and keeps processing. The following example with a misspelled word just gives you a warning.

```
porc print data=one;
run;
```

results in:

```
WARNING 14-169: Assuming the symbol PROC was misspelled as porc.
```

However, SAS doesn't always know what to assume. If you misspelled the procedure name you could get an ERROR message stating there was a misspelling but SAS couldn't assume what you did want so processing stopped.

```
 proc printx data=one; run;
      ------
      181
ERROR 181-322: Procedure name misspelled.
```

Now you might think you would get the same message no matter how you misspelled the procedure. This isn't true as shown below.

```
proc prnt data=one; run;
```

results in the following:

```
ERROR: Procedure PRNT not found.
```

Notes Due to Data

The following code is perfectly correct.

```
proc freq data=check;
     tables a*c/chisq;
```

```
     run;
```

However, you look at your output and don't see the CHISQ results. When you check the log, you'll discover the following note:

```
NOTE: No statistics are computed for a * c since a has less than 2 nonmissing
levels.
```

The CHISQ computation had requirements that weren't met but SAS was able to give you the rest of your requested output that didn't have to meet those requirements.

A Merge Problem

I worked with someone who had seen the note "MERGE statement has more than one data set with repeats of BY values." He didn't think it was a problem and had never looked at why he received the message. At the time, I had never had a case where it wasn't a problem!

```
data one;
 a=3; b=4; output;
 a=3; b=5; output;

data two;
 a=3; c=6; output;
 a=3; c=7; output;
 run;

** assumes data is already sorted;
 data check;
 merge one two;
  by a;
  run;

NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 2 observations read from the data set WORK.ONE.
NOTE: There were 2 observations read from the data set WORK.TWO.
NOTE: The data set WORK.CHECK has 2 observations and 3 variables.
```

The results are:

```
 3  4  6
 3  5  7
```

Is this correct or not? That depends, what did you want it to be?

Let's look at another case where the number of observations was not the same.

```
data one;
 a=3; b=4; output;
 a=3; b=5; output;
 a=3; b=9; output;
run;

data two;
 a=3; c=6; output;
```

```
 a=3; c=7; output;
run;

data check;
   merge one two;
   by a;
run;
```

These results are :

1	3	4	6
2	3	5	7
3	3	9	7

In this case, it is more likely that these are not the results you wanted since the last value of C was assigned to the “extra” records from the first data set.

Any time you get this message, you should first ask yourself if it is acceptable to have more than one record with the same BY values. Let’s assume you were not expecting any duplicates. How can you find them to investigate the data? The following code deletes the records that had a unique value for A. The duplicate records will be left.

```
*review duplicates;
data one_dup;
  set one;
  by a;
 if first.a and last.a then delete;
run;
```

First. and Last. are automatic variables created by SAS. They will not be in your output data set unless you assign them to a variable. As shown below, FIRST gets assigned a 1 if it was the first occurrence of a value. Otherwise, it is assigned a 0. Similarly, LAST gets assigned a 1 if it was the last occurrence of a value. Unique values of you BY variable will have both FIRST and LAST set to 1.

A	First	Last
1	1	1
2	1	0
2	0	1
3	1	0
3	0	0
3	0	1

Once you’ve looked at your duplicates, you’ll need to determine how to eliminate them. You may just be able to use PROC SORT with the NODUP option. In other cases, you may need to write more complex logic to keep the value you want. You might have other cases where you need to summarize the duplicate records into one record by using a calculation such as sum, minimum, or maximum.

Now in the case where it is acceptable to have duplicate records, you’ll need to review the data carefully to see if are getting the correct results. In some cases, it will be impossible to get the correct results given the business logic requested. In such cases, changing the definition of “correct results” may be your best option.

More Merge and Data Issues

This simple code produces a warning and perhaps wrong results.

```
data one;
   x='a';
run;

data two;
  x='ab';
run;

data thr;
   merge one two;
    by x;
run;
```

The log state:

```
WARNING: Multiple lengths were specified for the BY variable x by input data
sets. This may cause unexpected results.
NOTE: There were 1 observations read from the data set WORK.ONE.
NOTE: There were 1 observations read from the data set WORK.TWO.
NOTE: The data set WORK.THR has 1 observations and 1 variables.
```

If you look at the results, you see that X='A'. Is that what you expected? Since X was only a length of 1 in the first data set, only one character was read from the second data set when the merge was done.

What happens if you had reversed the data set order when you did your merge as is;

```
 data thr;
   merge two one ;
    by x;
run;
```

In this case, the log shows:

```
NOTE: There were 1 observations read from the data set WORK.TWO.
NOTE: There were 1 observations read from the data set WORK.ONE.
NOTE: The data set WORK.THR has 2 observations and 1 variables.
```

Surprise! You do not get a WARNING note. The results are also different. In this case, you get two observations. The first has a value of "A" and the other has a value of "AB". Why? By changing the order of the data sets, you started with AB and tried to merge it with a record having a value of A. Unlike in the first example where truncation occurred, you started with the longer value. The values do not match so you get one record from each data set.

The moral of the story is to always check that you have the same length on your BY variables.

What happens if you try to do the same thing in PROC SQL?

```
proc sql;
 create table yy as
 select x
 from one a join two b
  on a.x=b.x;
```

If you submit the above code, you'll get

```
ERROR: Ambiguous reference, column x is in more than one table.
```

In SQL, fields from different data sets with the same name must be uniquely identified.

Changing the code to the following may still not give you correct results.

```
proc sql;
 create table yy as
 select a.x
 from one a join two b
  on a.x=b.x;
```

You don't get an ERROR message but you do not get any observations either. The values of X are not identical matches.

The following code attempts to create a data set with records from both data sets even if they don't match.

```
proc sql;
 create table yy as
 select a.x , b.x
 from one a full join two b
 on a.x=b.x;
```

Since I said that SQL forces you to uniquely identify each field, why won't this work? You get the WARNING

```
WARNING: Variable x already exists on file WORK.YY.
```

If you look at the output data set, you'll see

Obs	x
1	A
2	

That is the right number of records but the records that were contributed from the second data set will have a blank for the value of X. The output data set can only have one value of X so the second X cannot be added to the data set. The following code will let you see the different data values.

```
proc sql;
 create table yy as
 select a.x , b.x as new_x
 from one a full join two b
 on a.x=b.x;
```

This code writes the second X as a different variable. Your results show:

Obs	x	new_x
1	A	
2		AB

This is also a useful way to look at your data if you don't think your code is working right. The code could be right and the problem is with unexpected data.

Two Fields, One Name

Notice that in the SQL example you were given a warning that the variable already existed. You don't get a warning with a data step merge.

```
  data one;
    x='a'; z=4; output;
 run;

 data two;
  x='a'; z=5; output;
  run;

  data thr;
   merge one two  ;
   by x;
run;
```

The log shows:

```
NOTE: There were 1 observations read from the data set WORK.ONE.
NOTE: There were 1 observations read from the data set WORK.TWO.
NOTE: The data set WORK.THR has 1 observations and 2 variables.
```

You might think everything is fine. However, depending upon what you wanted for Y, you might have incorrect values. If you put your data sets in a different order, you'll get different results. If you have some of the same variables in the data sets you are merging, you should consider either merging by all the variables that are the same or rename the variables so no data are lost. The following would enable you to look at the values for Z from both data sets.

```
data thr;
 merge one two (rename=(z=two_z));
 by x;
run;
```

A Different Merge Issue

Let's assume you submitted the following code:

```
data thr;
 merge one two;
 by x;
run;
```

This time the log shows

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

However, when you look at the data values, they both look like numeric values. Use PROC CONTENTS or the VAR command and you'll discover than one variable is stored as character even though the values in the field are numeric.

This is a case where you'll need to create a new variable in one data set so the variables are both numeric or both character.

Specifying valid parameters

One of the great features of SAS is the number of built-in functions. However, you need to ensure you know what the function expects as parameters.

```
data one;
      dt='23oct1954:07:20:08.003'dt;
        month=month(dt);
        run;
```

will results in a log with:

```
NOTE: Invalid argument to function MONTH at line 71 column 13.
dt=-163787992 month=. _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following places.
The results of the operations have been set to missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 71:13
```

This code generates 2 NOTEs but you really only have one mistake. This error will also count the number of data errors. Each observation with an error will increase the count. After you reach a certain number of errors, additional errors will not be printed in the log. The maximum number printed can be controlled by an option. If you have the maximum number and clean up the problem, you may resubmit the job and discover there were actually more problems that didn't get cleaned up. This situation can often happen if the errors are being generated when you are reading in data from an external file.

How do you eliminate the NOTEs? In this case, you need to provide a valid parameter. Given that the MONTH function is expecting a date value rather than a date/time value, you just need to specify a date. For this example, using an additional function will solve the problem.

```
data one;
      dt='23oct1954:07:20:08.003'dt;
     month=month(datepart(dt));
run;
```

I just happened to type an invalid datetime when I was testing my examples. This code looks almost identical to the original code but the errors are different.

```
 data one;
     dt='23otc1954:07:20:08.003'dt;
     month=month(dt);
run;
```

The log shows:

```
data one;
NOTE: SCL source line.
50         dt='23otc1954:07:20:08.003'dt;
              -------------------------
              77
ERROR: Invalid date/time/datetime constant '23otc1954:07:20:08.003'dt.
ERROR 77-185: Invalid number conversion on '23otc1954:07:20:08.003'dt.
```

```
51          month=month(dt);
52          run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.ONE may be incomplete.  When this step was stopped
there were 0
          observations and 2 variables.
WARNING: Data set WORK.ONE was not replaced because this step was stopped.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The log for this code is very different. Your original problem is still there but another error was encountered first so you never see the notes about the second problem. Unlike the previous example, this error actually stopped processing. Notice there is a NOTE about SCL. What? You weren't using any SCL or don't even know what it is? The SCL NOTE gets generated due to the processing SAS does behind the scenes. Two ERROR messages were generated when you really have one mistake. Notice that the ERROR messages are in a slightly different format with the second ERROR having an error number printed as well. Unlike the previous example, there data errors are not counted. If you had additional records, you would not see any additional errors because the system stopped processing and did not read any more records.

The fix for this error is to provide a valid date/time value. In case you didn't notice, I had merely typed OCT as OTC.

Looks can be Deceiving

Some one asked why the following statement did not work yet it had before on the same computer.

```
unqdocid = put(docind, z12.);
```

He received the following in his log:

```
 NOTE: SCL source line.
1248      unqdocid = put(docind, z12.);
                                 ----
                                 484
NOTE 484-185: Format $Z was not found or could not be loaded.
```

He asked how to reload the formats. This happens to be a SAS-supplied format. The real answer wasn't that his format had disappeared. Read the note carefully and you see the format SAS is looking for starts with a "$" which indicates it is to be used with a character field. SAS thinks the field docind is character and is looking for a character format. However, the format Z12. is used for placing lead zeroes on numeric fields. This contradictory information is the key to the problem. He did not have a problem with his format but a problem with the data itself. Somehow, his numeric field docind had become a character field. This could have been caused by other code earlier in the program or perhaps by a change in the source of his data.

Eliminating Notes/Warnings/Errors
SAS-L is an electronic newsgroup. Readers often see questions from someone asking how to turn off these Note/Warning/Error messages. The subsequent response is usually from a very experienced user asking some variation of "Why would you want to delete vital information?" I hope that you've seen some examples that show why the notes/warnings/errors are indeed vital information.

However, there are cases where there is a valid reason to try to reduce the size of the log. For example, some statistical techniques require many iterations in the same job so the log becomes quite lengthy. It is possible to run out of space for the log especially if you are working in display manager. A recommended option is to route the log to another location. There are multiple ways to do this and the best one depends upon your operating system, whether you are using SAS in a batch or interactive environment, and whether you want the entire log or just a portion of it rerouted. Some methods such as ATLLOG are only available at the time you invoke SAS. The following example can be used within the program to route just part of the log.

```
proc printto log='specify-appropriate-location-for-your-system';
   run;
```

The log after this point will go to your specified location. To return the log to the default location in the same program or interactive session, just use the PROC again without specifying the log location as in:

```
proc printto;
run;
```

If you really need eliminate some of the messages, you should explore options such as nonotes, nosource, nosource2. Also, if your code has macros, you'll want to use nomprint. You'll still see the ERROR messages even with these options.

Additional Resources
The SAS documentation provides additional information on these messages.

SAS-L is also useful. There are several ways to access SAS-L. One if through http://www.listserv.uga.edu/archives/sas-l.html It is recommended that you search previous messages before posting a question since your question may already have been asked.

Previous SUGI and regional conference presentations may also assist you. Lex Jansen has created a search engine for the online papers. The search engine can be accessed at www.lexjansen.com/sugi.

Summary
This paper covers a few of the possible notes, warnings and errors you might get in your program. I tried to find a list of all the possible errors but could not. That might be an impossible task.

Some messages make the problem quite clear. In other cases, however, the messages suggest it is one thing but the real problem is something else that did not generate a message. This can be due to the data itself. You should look at your data as well as your code. My examples used very simple data sets. This same method can be used to help you determine if some errors are due to your code or the data because you control the data. Besides looking at the log, you may need to look at other tools such as PROC CONTENTS or the VAR command to see what SAS really has stored since it may not be what you think.

Contact Information

Deb Cassidy
Debc_sugstuff@usa.com