Paper 232-30

Explaining Unexpected Log Messages and Output Results from DATA Step Code

Debbie Buck, D. B. & P. Associates, Houston, TX
Larry Stewart, SAS Institute Inc., Cary, NC

ABSTRACT

Everyone knows that you should always read the SAS® log every time you execute a SAS program, especially if it includes a DATA step. When you read the log, do you ever find SAS notes or warnings that puzzle you? You know the ones like "The variable XYZ is uninitialized" or "The variable ABC in the DROP, KEEP, or RENAME has never been referenced." But when you look at your output everything seems to be okay. So what do those messages mean and should you be concerned? Do you submit DATA steps that produce no notes or error messages in the log, but the code produces unexpected results? For example, you write a basic IF/THEN statement to delete specific observations, but it deletes all the observations. If you answered yes to any of the questions above, then this might be the presentation for you. This presentation focuses on common unexpected notes in the SAS log and why they occur. It also addresses common logic errors in DATA steps that lead to unexpected results.

INTRODUCTION

SAS programs generate numerous notes, warnings, and error messages in the SAS log to help you understand what your code does and to help you debug your programs.

This presentation offers a collection of examples that introduce common notes, warnings, error messages, and unexpected results that occur in SAS programs, and attempts to explain what the messages mean and why the unexpected results occur. Most of the examples are geared toward beginning to intermediate level SAS programmers.

UNINITIALIZED VARIABLES

This example uses a DATA step to produce a temporary SAS data set JANUARY that contains a subset of the data stored in a permanent SAS data set SUGI.SALESDATA. The step also includes a FORMAT statement that displays the sales figures as monetary values. In the SAS log shown below, there is a note indicating that the variable Sale is uninitialized. What does that mean?

Partial SAS Log

```
2   data January;
3      set sugi.salesdata;
4      if Month=1;
5      format Sale dollar11.2;
6   run;

NOTE: Variable Sale is uninitialized.
```

The note means that the DATA step encountered a variable (Sale) that cannot be assigned any values. In this example, that means that the variable Sale is not in the input data set SUGI.SALESDATA, and it is not used in any programming statements that assign values to it. The new data set, JANUARY, contains all the variables from SUGI.SALESDATA plus a new variable named Sale. The variable Sale has all missing values. This type of error typically occurs when a variable name is misspelled, as is the case in this example. The correct variable name is Sales.

MISSING VALUES

A very common task in a DATA step is to sum the values of various variables in each row of a data set. In this example, sales values for January, February, and March are being summed to get a total sales figure for the first quarter. When the code is executed, a note is written to the SAS log (shown below) indicating that missing values were generated as a result of performing an operation on missing values. What does that note mean?

Partial SAS Log

```
3    data Quarter1;
4       set sugi.sales;
5       Qtr1Sales=Jan+FebSales+MarSales;
6    run;

NOTE: Missing values were generated as a result of performing an operation on
      missing values.
Each place is given by:
(Number of times) at (Line):(Column).
2 at 5:17
```

The note means that the value assigned to Qtr1Sales is missing (Missing values were generated as a result of) because at least one of the values being summed (values of Jan, FebSales, and MarSales) is missing (performing as operation on missing values). The second part of the note identifies exactly how many times a missing value was generated (Number of times=2) and which line (Line=5) in the DATA step generated the missing values. The column value (Column=17) is the location of the first plus sign (+), indicating that the addition is the operation that was performed on the missing values.

If you want SAS to sum only the non-missing values, you can use the SUM function instead of the addition operator. The Qtr1Sales variable is assigned the total of the non-missing values of Jan, FebSales, and MarSales. No note indicating that missing values were generated is written to the SAS log, as shown below.

Partial SAS Log

```
3    data Quarter1;
4       set sugi.sales;
5       Qtr1Sales=sum(Jan,FebSales,MarSales);
6    run;

NOTE: There were 4 observations read from the data set SUGI.SALES.
NOTE: The data set WORK.QUARTER1 has 4 observations and 5 variables.
```

NON-REFERENCED VARIABLES

In this DATA step, the task is to change the name of the variable Jan so it is consistent with FebSales and MarSales. A simple way to accomplish this is to use the RENAME= data set option in the DATA statement. However, when the following code is submitted, a warning message is written to the SAS log indicating that there is a non-referenced variable in the DATA step. What does the warning mean?

Partial SAS Log

```
3  data Quarter_sales (rename=(JanSales=Jan));
4     set sugi.sales;
5     Qtr1Sales=sum(Jan,FebSales,MarSales);
6  run;

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

The warning indicates that the variable JanSales is not defined anywhere in the DATA step (the variable JanSales in the RENAME list has never been referenced), which means that it does not exist in the input data set SUGI.SALES, and is not created by any statements in the DATA step. So what went wrong?

Unlike assignment statements, the form of a RENAME= data set option is old-variable=new-variable. If you change the RENAME= data set option to (Jan=JanSales), the program runs correctly.

READING PAST THE END OF A DATA LINE

SAS DATA steps are often used to read files that contain data lines with fields that are delimited with blanks. An example is the SALESREP.DAT file shown below.

c:\salesreps.dat

```
Lee Max K. East
Marks Sara South
Bell Bob T. North
Cox Tom C. West
```

The DATA step shown in the SAS log below uses list input to read the SALESREP.DAT file. The note in the SAS log indicates that SAS went to a new line when the INPUT statement reached past the end of a line. What does the note mean?

Partial SAS Log

```
1   data SalesReps;
2      infile 'c:\salesreps.dat';
3      input LName $ FName $ MI $ Region $;
4   run;

NOTE: 4 records were read from the
      infile 'c:\salesreps.dat'.
      The minimum record length was 15.
      The maximum record length was 17.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

Viewing the resulting data set helps explain the meaning of the note.

```
 Obs    LName    FName    MI       Region

 1      Lee      Max      K.       East
 2      Marks    Sara     South    Bell
 3      Cox      Tom      C.       West
```

When you use list input, the INPUT statement scans for the first nonblank character on the data line, reads the value until it reads a blank, and assigns the value to the first variable on the INPUT statement. It then scans for the next nonblank character, reads the value until it reads a blank, and assigns the value to the second variable on the INPUT statement. This process continues until it has read a value for each variable on the INPUT statement. In the SALESREP.DAT file, the person in the second data line does not have a value for MI, so there are only three fields on that data line. The values of LName and FName are assigned correctly, but MI receives the value “South” from the third field on the data line, which is the value that should be assigned to Region. The value for MI should be missing. The INPUT statement recognizes that there is a fourth variable on the INPUT statement and scans for the next nonblank character. Because there is not a fourth data field on the data line, SAS reaches the end of that data line (INPUT statement reached past the end of a line) and moves to the next data line in the file (SAS went to a new line) to find a value for the fourth variable, Region. In this case, it finds the value Bell, which is the LName of the person on the third data line.

The solution to this problem is to put a placeholder for the missing field in the SALESREP.DAT file as shown below (note the single period in the second data line after the value Sara), and execute the same DATA step shown above.

c:\salesreps.dat

```
Lee Max K. East
Marks Sara . South
Bell Bob T. North
Cox Tom C. West
```

CHARACTER VARIABLE LENGTHS

Concatenating SAS data sets is a common DATA step task and can produce unexpected results. For example, if the data sets being concatenated have a common character variable but the variable has different lengths in the data sets, truncation can occur in the resulting data set. In the example below, the variable State has a length of 11 in the SUGI.REGION1 data set and a length of 14 in the SUGI.REGION2 data set.

SUGI.REGION 1

State	Pop2000
Connecticut	3,405,584
Maine	1,274,923

SUGI.REGION2

State	Pop2000
North Carolina	3,405,584
South Carolina	1,274,923

If the two data sets are concatenated using the following DATA step, the values of State from the SUGI.REGION2 data set are truncated in the resulting data set named BOTH.

SAS Program

```
data both;
   set sugi.region1 sugi.region2;
run;
```

This truncation occurs because the length of the variable State in the resulting data set is defined when SAS first encounters the variable in the DATA step. Because SUGI.REGION1 is listed first in the SET statement, then the length of the variable State in the BOTH data set is set to the length of State in that data set, which is 11. Therefore, only the first 11 characters of the State values are stored in the BOTH data set.

A simple solution is to define the length of State using a LENGTH statement. However, a common error is to place the LENGTH statement is the wrong place. In the DATA step below, the LENGTH statement is placed after the SET statement which produces a warning message.

SAS Program

```
data both;
   set sugi.region1 sugi.region2;
   length State $ 14;
run;
```

SAS Log Note

```
WARNING: Length of character variable has already been set. Use the LENGTH statement
         as the very first statement in the DATA STEP to declare the length of a
         character variable.
```

By placing the LENGTH statement after the SET statement, State is first encountered when the SET statement opens the SUGI.REGION1 data set. As the warning indicates, the LENGTH statement must be placed before the SET statement. The correct code is shown below.

SAS Program

```
data both;
   length State $ 14;
   set sugi.region1 sugi.region2;
run;
```

FORMAT WIDTH TOO SMALL

In the example below, PROC PRINT is used to produce a listing of the SUGI.SALESDATA data set. A FORMAT statement is used to display the values of the Sales variable with dollar signs, commas, and two decimal places. Also, a SUM statement is used to display a total for the Sales column. There are no error messages, but there is a note indicating that at least one W.D format was too small for the number to be printed. What does that note mean?

Partial SAS Log

```
2  proc print data=sugi.salesdata;
3     title 'Sales Data for All Regions';
4     format Sales dollar9.2;
5     sum Sales;
6  run;

NOTE: At least one W.D format was too small for the number to be printed. The
      decimal may be shifted by the "BEST" format.
```

In the PROC PRINT output, the commas aren't displayed for the Sales values and neither the $ or comma is displayed for the total of the Sales column. The total of the Sales column is what produced the note. The width of the specified format, 9.2, is not wide enough to display the value with the $ and the comma (at least one W.D format was too small for the number to be printed). The width you specify in a format must be wide enough to display the value and any characters that are normally displayed with the format. In this case, the total of the Sales column requires eight spaces for the digits (6 for the dollar amount and 2 for the decimal places or cents), one space for the decimal point, one space for the $, and one space for the comma for a total width of 11 spaces. The FORMAT statement in the code only specifies nine spaces for the width. SAS automatically switches to the BEST9. format to display the value (the decimal may be shifted by the "BEST" format). The BEST format displays the values the best way it can with the specified width, as shown in the output below.

SAS Output

```
    Sales Data for All Regions

  Obs    Region    Month        Sales

    1       1        1      $18928.23
    2       1        2      $53978.84
    3       2        1      $39092.03
    4       2        2      $19756.74
    5       3        1      $46329.44
    6       3        2      $20984.16
    7       4        1      $33824.77
    8       4        2      $27492.62
    9       5        1      $19744.93
   10       5        2      $28474.11
                            =========
                            308605.87
```

The solution is to increase the width to 11 (format Sales dollar11.2).

DATA CONVERSION

One note that often appears in the SAS log is that numeric values have been converted to character values or that character values have been converted to numeric values. When this happens, you may or may not see any unexpected results in the SAS output. In this example, two variables (Building and Room) are concatenated to create a variable named Location. Building is a character variable and Room is numeric. Because the concatenation operator (||) performs a character operation, SAS expects both variables to be character variables. The values of Room are automatically converted from numeric to character (Numeric values have been converted to character values) so they can be concatenated to the values of Building. The note also identifies where in the code the conversion took place (at the places given by: (Line):(Column). 5:28). The 5 specifies the line number in the SAS log (the Location= assignment statement) and the 28 specifies the location in the line. The 28th character in the line is the "R" in Room, indicating that the values of Room were converted from numeric to character.

Partial SAS Log

```
  3    data Classrooms;
  4       set sugi.Rooms;
  5       Location=Building||'-'||Room;
  6    run;

  NOTE: Numeric values have been converted to character values at the places given
        by: (Line):(Column). 5:28
```

Data conversion often has no adverse effect on the results, but in this case it produces unexpected results. The PROC PRINT output below displays some extra blanks embedded in the values of Location.

SAS Output

```
   Obs     Building     Room       Location

    1         A          125    A-          125
    2         A          135    A-          135
    3         B          101    B-          101
    4         C          345    C-          345
```

These extra blanks are produced by the data conversion. Because SAS is converting a number to a character string, it must select a format and a default width to use to determine how to display that string. The default format is the BEST12. format. That means that SAS writes the number the best way it can within a width of 12 spaces and converts that value to a character string. Because the value being converted is numeric, SAS right aligns the resulting string within the 12 spaces. The Room values are all three digits, so there are nine blanks stored before the three digits. Those 12 characters are then concatenated onto the end of the values of Building which produces the nine unexpected embedded blanks. This problem is easily corrected using the PUT function. The PUT function enables you to specify what format and width to use when the conversion is performed. In the code below, the PUT function writes the values of Room as three digits (3. in the second argument) and converts the resulting value to a character string that is three bytes long. Those three characters are then concatenated to the end of the Building values. Because a format of 3. is used instead of the default BEST12., there are no extra blanks. Also, the data conversion note is not written to the SAS log.

```
   data Classrooms;
      set sugi.Rooms;
      Location=Building||'-'||put(Room,3.);
   run;
```

CONCATENATION OF CHARACTER VARIABLES

Concatenation is often used to combine two or more character variables into a new variable. In this example three character variables, FName, MI, and LName from the SALESREPS data set are being concatenated in the SALESREPS1 data set to create the variable FullName using the concatenation operator (||). The program below results in a number of unexpected embedded blanks in FullName. What happened here?

SAS Program

```
   data SalesReps1;
      set SalesReps;
      FullName=FName||' '||MI||' '||LName;
   run;
```

SAS Output

```
   Obs    LName   FName   MI    FullName
    1     Lee     Max     K.   Max     K.     Lee
    2     Marks   Sara         Sara           Marks
    3     Bell    Bob     T.   Bob     T.     Bell
    4     Cox     Tom     C.   Tom     C.     Cox
```

One solution to remove the blanks is to use the TRIM function. The purpose of this function is to remove trailing blanks. When the TRIM function is used with FName and MI this solves the problem for most of the observations. However, the second observation still has multiple blanks between the first and last names due to the missing MI value.

SAS Program

```
   data SalesReps1;
      set SalesReps;
      FullName=trim(FName)||' '||trim(MI)||' '||LName;
   run;
```

SAS Output

```
   Obs   LName  FName  MI   FullName
    1    Lee    Max    K.  Max K. Lee
    2    Marks  Sara       Sara   Marks
    3    Bell   Bob    T.  Bob T. Bell
    4    Cox    Tom    C.  Tom C. Cox
```

A second solution would be to write conditional statements which would concatenate FullName differently when MI is missing. However, in SAS 9 there is a new function CATX that handles this problem without additional coding. The CATX function concatenates character strings, removes leading and trailing blanks and inserts separators. The form of the CATX function is CATX(separator,string-1,string-2, etc.). The following program results in the desired output.

SAS Program

```
   data SalesReps1;
      set SalesReps;
      FullName=catx(' ',FName,MI,LName);
   run;
```

SAS Output

```
   Obs    LName    FName    MI     FullName

    1     Lee      Max      K.    Max K. Lee
    2     Marks    Sara           Sara Marks
    3     Bell     Bob      T.    Bob T. Bell
    4     Cox      Tom      C.    Tom C. Cox
```

DATA STEP LOOPING

The next example introduces a note in the SAS log that doesn't occur as frequently as other notes and error messages presented in this paper, but it can cause some confusion. The objective in this example to is read only the first three observations from the SUGI.SALESDATA data set. Every DATA step creates a temporary numeric variable named _N_ that counts the number of times the DATA step begins execution. In most DATA steps, SAS executes once for each row that is read from an input file. The code below uses the variable _N_ in a subsetting IF statement to prevent any rows from being written to the SUBSET data set after the third execution of the DATA step. The two notes in the SAS log seem to indicate that the correct number of rows were read from SUGI.SALESDATA and written to SUBSET, so why is there a note indicating that the DATA step stopped due to looping?

Partial SAS Log

```
   2    data subset;
   3       if _n_ le 3;
   4       set sugi.salesdata;
   5    run;

   NOTE: DATA STEP stopped due to looping.
   NOTE: There were 3 observations read from the data set SUGI.SALESDATA.
   NOTE: The data set WORK.SUBSET has 3 observations and 3 variables.
```

By default, anytime a DATA step includes a statement that reads data from a file (for example, an INPUT, SET or MERGE statement), the DATA step automatically executes until the read statement encounters an end-of-file marker. By placing the subsetting IF statement before the SET in the code above, the DATA step will never execute the SET statement after the third execution of the DATA step. Therefore, the SET statement can never reach the end-of-file marker. Because the DATA step naturally executes until it reaches the end-of-file marker, this code produces an infinite loop. However, SAS recognizes the problem and stops the DATA step from continuing to execute.

One solution is to move the subsetting IF statement after the SET statement, but that is inefficient because the SET statement would read every row in the SUGI.SALESDATA data set when only three rows are needed. A better solution is to use the OBS= data set option, as shown below.

Partial SAS Log

```
2   data subset;
3      set sugi.salesdata(obs=3);
4   run;

NOTE: There were 3 observations read from the data set SUGI.SALESDATA.
NOTE: The data set WORK.SUBSET has 3 observations and 3 variables.
```

UNEXPECTED RESULTS USING CONDITIONAL LOGIC

This example uses conditional logic to delete unwanted rows from a SAS data set. The specific problem is to create a temporary SAS data set named USSALES by reading the SUGI.SALESDATA data set and deleting any rows where the value of Region is 3 or 5. There are five different regions (1,2,3,4,5) in the SUGI.SALESDATA data set so the resulting data set should contain rows for regions 1, 2, and 4. However, the note in the SAS log below indicates that the USSALES data set contains no observations. Why were all the rows deleted?

Partial SAS Log

```
2    data USSales;
3       set sugi.salesdata;
4       if Region=3 or 5 then delete;
5    run;

NOTE: There were 10 observations read from the data set SUGI.SALESDATA.
NOTE: The data set WORK.USSALES has 0 observations and 3 variables.
```

The reason all the rows are deleted is because of the way SAS evaluates the two conditions in the IF statement. When a condition involves a comparison (Region=3), SAS compares the value of Region to the constant value 3 and returns a 1 if the values match (condition is true) or 0 if they do not match (condition is false). However the second condition (5) is not a comparison. It is a constant. SAS evaluates a constant as true if it is a nonzero non-missing value and false if it is zero or missing. The value 5 is a nonzero non-missing value, so SAS evaluates it as being true regardless of the value of Region. Therefore, the DELETE on the IF statement is executed for every row read from SUGI.SALESDATA. Two solutions to the problem are shown below.

Solution 1

```
data USSales;
   set sugi.salesdata;
   if Region=3 or Region=5 then delete;
run;
```

Solution 2

```
data USSales;
   set sugi.salesdata;
   if Region in(3,5) then delete;
run;
```

UNEXPECTED RESULTS USING CONDITIONAL LOGIC

The purpose of this program is to create a temporary SAS data set LOW_SALES from the permanent SAS data set SUGI.SALESDATA keeping those observations with Sales values less than $20,000 in Regions 1 or 2. When the following program is run, there are no warnings or error messages, but the output contains an undesired observation with a Sales value greater than $20,000. What happened here?

SAS Program

```
data Low_sales;
   set sugi.salesdata;
   if Region=1 or Region=2 and Sales lt 20000;
run;
```

SAS Output

```
Region    Month      Sales

   1        1      18928.23
   1        2      53978.84
   2        2      19756.74
```

When SAS executed the DATA step, the comparisons were not grouped as intended because the AND operator has higher priority than the OR operator. Instead of selecting rows with Sales values less than $20,000 in Regions 1 or 2, it selects rows satisfying Region=1 regardless of Sales amount and rows where Region=2 and Sales were less than 20000. The solution to this problem is to use parentheses to ensure that SAS interprets the ‘or’ and ‘and’ grouping in the manner intended.

SAS Program

```
data Low_sales;
   set sugi.salesdata;
   if (Region=1 or Region=2) and Sales lt 20000;
run;
```

UNEXPECTED RESULTS USING CONDITIONAL LOGIC

Conditional logic is often used to create new variables. In this example the task is to create a new variable SalesGroup in the temporary SAS data set GROUPS that has values of Low, Medium, or High based on the Sales variable amount. When the program is submitted, the results are not what were expected. The output shows SalesGroup values of ‘High’ on observations that should be ‘Low’ (see rows 1, 4, and 9 in the output below). What is happening here?

SAS Program

```
data Groups;
   set sugi.salesdata;
   length SalesGroup $ 6;
   if Sales lt 20000 then SalesGroup='Low';
   if Sales ge 20000 and Sales lt 40000 then SalesGroup='Medium';
   else SalesGroup='High';
run;
```

SAS Output

```
                               Sales
Obs    Region       Sales      Group

  1       1      18928.23      High
  2       1      53978.84      High
  3       2      39092.03      Medium
  4       2      19756.74      High
  5       3      46329.44      High
  6       3      20984.16      Medium
  7       4      33824.77      Medium
  8       4      27492.62      Medium
  9       5      19744.93      High
 10       5      28474.11      Medium
```

Using ELSE statements in conjunction with IF-THEN statements can make the program more efficient because SAS doesn't have to check every IF statement after an IF condition is satisfied. However, the ELSE is associated only with the IF-THEN statement that immediately precedes it. In this case, the second IF-THEN and associated ELSE statement overwrote the SalesGroup value from the first IF-THEN. The solution is to insert ELSE before the second IF-THEN so that all three statements are linked.

SAS Program

```
data Groups;
   set sugi.salesdata;
   length SalesGroup $ 6;
   if Sales lt 20000 then SalesGroup='Low';
   else if Sales ge 20000 and Sales lt 40000 then SalesGroup='Medium';
   else SalesGroup='High';
run;
```

USING A FORMATTED VALUE AS THE DATA VALUE

Currency data are often displayed with formats in order to include dollar signs and commas in the output. A subset of the following data is desired that extracts only those observations from SUGI.SALESDATA with a Sales value less than $20,000 for inclusion in the temporary data set LOW_SALES_REV.

SAS Output

```
Obs    Region    Month          Sales

  1       1        1      $18,928.23
  2       1        2      $53,978.84
  3       2        1      $39,092.03
  4       2        2      $19,756.74
  5       3        1      $46,329.44
  6       3        2      $20,984.16
  7       4        1      $33,824.77
  8       4        2      $27,492.62
  9       5        1      $19,744.93
 10       5        2      $28,474.11
```

The following code is submitted and results in a number of error messages in the log that are not quite clear. What happened in this program?

SAS Program

```
data low_sales_rev;
   set sugi.salesdata;
   if Sales lt $20,000;
run;
```

Partial SAS Log

```
3    data low_sales_rev;
4       set sugi.salesdata;
NOTE: SCL source line.
5       if Sales lt $20,000;
                    -
                    390
                    76
                     --
                     200
ERROR 390-185: Expecting an relational or arithmetic operator.

ERROR 76-322: Syntax error, statement will be ignored.

ERROR 200-322: The symbol is not recognized and will be ignored.

6    run;
```

SAS is looking for a variable, a constant, or some other expression after the LT operator that it can compare to the Sales variable value for each observation (Error 390-185). Because SAS doesn't see any of these, it considers this a syntax error and recognizes that the dollar sign is not a valid symbol in this statement and therefore ignores the statement. (Errors 76-322 and 200-322)

A formatted value is simply a way of displaying a value. The data is not stored in that form. The solution is to remove the dollar sign and comma and use the numeric form of 20000 which is consistent with the Sales variable values for comparison in the IF statement.

SAS Program

```
data low_sales_rev;
   set sugi.salesdata;
   if sales lt 20000;
run;
```

SAS DATE CONSTANTS

The following is PROC PRINT output from a data set named SUGI.MONEY. The purpose of this program is to create a data set MONEY_BEFORE_APRIL that only keeps observations in the first quarter (Date value occurring before April 1). However, when the DATA step is run, several notes indicating invalid numeric data, '04/01/04' are written to the SAS log. What happened in the SAS program below?

SAS Output

```
Obs      Date       Sales
 1     01/01/04    18928.23
 2     03/01/04    39092.03
 3     05/01/04    46329.44
 4     07/01/04    33824.77
```

Partial SAS Log

```
3    data Money_before_April;
4       set sugi.money;
5       if Date le '04/01/04';
6    run;

NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      5:15
NOTE: Invalid numeric data, '04/01/04' , at line 5 column 15.
Date=16071 Sales=18928.23 _ERROR_=1 _N_=1
NOTE: Invalid numeric data, '04/01/04' , at line 5 column 15.
Date=16131 Sales=39092.03 _ERROR_=1 _N_=2
NOTE: Invalid numeric data, '04/01/04' , at line 5 column 15.
Date=16192 Sales=46329.44 _ERROR_=1 _N_=3
NOTE: Invalid numeric data, '04/01/04' , at line 5 column 15.
Date=16253 Sales=33824.77 _ERROR_=1 _N_=4
NOTE: There were 4 observations read from the data set SUGI.MONEY.
NOTE: The data set WORK.MONEY_BEFORE_APRIL has 0 observations and 2 variables.
```

Although displayed with an MMDDYY8. format, the Date variable is a SAS date, and as such, is a numeric variable. SAS has converted the '04/01/04' to a numeric value to make it compatible for comparison with the variable Date in the subsetting IF statement (Line 26 of the log at Column 14 where the '04/01/04' value begins). However, this is not a valid numeric value, so the conversion returns a missing numeric value and a note is written to the SAS log. For each observation read, the conversion happens again and a NOTE is displayed.

The solution to this is to remember that when referencing a SAS date, the form is 'ddMMMyy'D, displayed as day, followed by month abbreviation, then year, all enclosed in quotes and followed by the 'D'. The 'D' is necessary to tell SAS this is a SAS date, not a character value. The following program provides the solution to the problem.

SAS Program

```
data money_before_April;
   set sugi.money;
   if Date le '01apr04'd;
run;
```

NESTED COMMENTS

In the following example a data set is created called QUARTERLY_SALES which contains a beginning Date and the Sales data for each quarterly period. As is the practice in many companies, a comment is inserted next to each variable name in the INPUT statement to help identify what information each variable represents.

SAS Program

```
data Quarterly_Sales;
   input date mmddyy8. /*Starting Date of Period*/
         sales;        /*Revenue for Period*/
   cards;
01/01/04 18928.23
04/01/04 19756.74
07/01/04 46329.44
10/01/04 20984.16
run;
```

After running this DATA step, checking the log, and finding no problems, a PROC PRINT is added to output the results to a listing file. Since everything appears to be correct in the DATA step, the DATA step is commented out using the /* */ type of comment. This type of comment begins with /* and ends with */ and can contain quotes and semicolons.

Since the QUARTERLY_SALES data set now exists in the WORK library, it is not necessary to rerun the DATA step in order to print it out. Unexpectedly, the following log appears with both notes and error messages and no output is produced. Why did this happen?

SAS Program

```
   /*-----------
data Quarterly_Sales;
   input date mmddyy8. /*Starting Date of Period*/
         sales;        /*Revenue for Period*/
   cards;
01/01/04 18928.23
04/01/04 19756.74
07/01/04 46329.44
10/01/04 20984.16
run;
   ------------*/
proc print data=Quarterly_Sales;
run;
```

Partial SAS Log

```
10      /*-----------
11   data Quarterly_Sales;
12      input date mmddyy8. /*Starting Date of Period*/
NOTE: SCL source line.
13            sales;
              -----
              180

ERROR 180-322: Statement is not valid or it is used out of proper order.

NOTE: SCL source line.
14      cards;
        -----
        180

ERROR 180-322: Statement is not valid or it is used out of proper order.
```

```
NOTE: SCL source line.
15   01/01/04 18928.23
     --
     180
ERROR 180-322: Statement is not valid or it is used out of proper order.

16   04/01/04 19756.74
17   07/01/04 46329.44
18   10/01/04 20984.16
19   run;

NOTE: SCL source line.
20      ------------*/
        -
        180
ERROR 180-322: Statement is not valid or it is used out of proper order.

21   proc print data=Quarterly_Sales;

22   run;
```

The program starts out correctly with the DATA step commented out, but begins to have problems as soon as the second comment is encountered. From this point forward, SAS does not recognize any statements (or data) in the program as valid. The comments with the input variables are nested within the overall comment around the DATA step. SAS does not support nested comments.

The solution to this is either to remove the comments from the INPUT statement or to submit only the PROC step.

CONCLUSION

This paper has offered a collection of examples that introduce common notes, warnings, error messages, and unexpected results that occur in SAS programs, and has attempted to explain what the messages mean and why the unexpected results happen. Hopefully, these examples will help you avoid common problems, or to know how to recognize and remedy them when they occur.

CONTACT INFORMATION

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

Debbie Buck
D. B. & P. Associates
10418 Indian Paintbrush Lane
Houston, TX 77095
Work Phone: (281)256-1619
Fax: (281)256-1634
Email: debbiebuck@houston.rr.com

Larry Stewart
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: (919)531-7358
Fax: (919) 677-4444
Email: larry.stewart@sas.com