Paper 074-29

Tales from the Help Desk: Solutions for Simple SAS ® Mistakes

Bruce Gilsen, Federal Reserve Board

INTRODUCTION

In 19 years as a SAS ® consultant at the Federal Reserve Board, I have seen SAS users make the same mistakes year after year. This paper reviews some common mistakes, shows how to fix them, and suggests why some of the mistakes occur so often.

1. USING THE LAG FUNCTION IN A CONDITIONAL STATEMENT.

Data set ONE has the following data.

```
VAR1   VAR2
 1      1
-1      2
 1      3
-1      4
 1      5
```

A user wants to assign values to the variable LAGVAR2 as follows.

If VAR1 is greater than 0, then LAGVAR2 = the value of VAR2 in the previous observation.

Otherwise, LAGVAR2 = missing (.).

The user submits the following DATA step. The user might include an ELSE statement after the IF statement, ELSE LAGVAR2 = .;, but this statement does not change the results, so it is not included in this DATA step.

```
data two;
  set one;
  if var1 > 0 then lagvar2 = lag(var2);
run;
```

Data set TWO has the following data. Note that LAGVAR2 is 1 in the third observation; 2 was expected. Also, LAGVAR2 is 3 in the fifth observation; 4 was expected.

```
VAR1   VAR2   LAGVAR2
 1      1        .
-1      2        .
 1      3       1
-1      4        .
 1      5       3
```

Users make this error because the intuitive definition of a LAG function is that it always returns the value in the previous observation. In fact, the LAG function works as follows. Every time LAG(VAR2) is executed,

the current value of VAR2 is stored on a queue

the value of VAR2 from the last time LAG(VAR2) was executed is retrieved

LAG functions that are executed conditionally (as in DATA step TWO above) only store and retrieve values from observations for which the condition is satisfied. This is best illustrated by example. In DATA step TWO, the condition is satisfied (VAR1 is greater than 0) in the first, third, and fifth observations. LAGVAR2 is assigned as follows.

Observation 1.
The current value of VAR2, 1, is stored. No previous value of VAR2 is available to retrieve, so LAGVAR2 is set to missing (.).

Observation 2.

The IF statement is false, so the LAG function is not executed, and nothing is stored or retrieved.

Observation 3.
The current value of VAR2, 3, is stored. The most recently stored value of VAR2, 1, is retrieved and assigned to LAGVAR2.

Observation 4.
The IF statement is false, so the LAG function is not executed, and nothing is stored or retrieved.

Observation 5.
The current value of VAR2, 5, is stored. The most recently stored value of VAR2, 3, is retrieved and assigned to LAGVAR2.

To get the expected results, move the LAG function outside of the conditional code, so that it executes in every observation, as in the following DATA step.

```
data two;
  set one;
  lagvar2 = lag(var2);
  if var1 <= 0 then lagvar2 = .;
run;
```

Data set TWO has the following data.

```
VAR1    VAR2    LAGVAR2
 1       1         .
-1       2         .
 1       3         2
-1       4         .
 1       5         4
```

This DATA step executes as follows.

Observation 1, statement with LAG function.
The current value of VAR2, 1, is stored. No previous lagged value of VAR2 is available to retrieve, so LAGVAR2 is set to missing.

Observation 1, IF statement.
The IF statement is false, so no action is taken.

Observation 2, statement with LAG function.
The current value of VAR2, 2, is stored. The most recently stored value of VAR2, 1, is retrieved and assigned to LAGVAR2.

Observation 2, IF statement.
The IF statement is true, so LAGVAR2 is set to missing.

Observation 3, statement with LAG function.
The current value of VAR2, 3, is stored. The most recently stored value of VAR2, 2, is retrieved and assigned to LAGVAR2.

Observation 3, IF statement.
The IF statement is false, so no action is taken.

Observation 4, statement with LAG function.
The current value of VAR2, 4, is stored. The most recently stored value of VAR2, 3, is retrieved and assigned to LAGVAR2.

Observation 4, IF statement.
The IF statement is true, so LAGVAR2 is set to missing.

Observation 5, statement with LAG function.
The current value of VAR2, 5, is stored. The most recently stored value of VAR2, 4, is retrieved and assigned to LAGVAR2.

Observation 5, IF statement.
The IF statement is false, so no action is taken.

The LAG function returns values that are lagged one time. The related functions LAG2, LAG3,..., LAG100 return values that are lagged 2, 3, ..., 100 times. These functions behave analogously to the LAG function in conditional code.

Another way to correctly code the DATA step is as follows.

```
data two;
  set one;
  drop _templagvar2;
  _templagvar2 = lag(var2);
  if var1 > 0 then lagvar2 = _templagvar2;
run;
```

2. CHARACTER VARIABLE TRUNCATION.

In data set ONE, created by the following DATA step, CHARVAR1 has the expected value, “fffggghhhiii”, but CHARVAR2 has the value “jjj”, not ”jjjkkklllmmm”.

```
data one;
  numvar = 100;
  charvar1 = "aaabbbcccddd";
  charvar2 = "eee";
  if numvar > 0 then
  do;
    charvar1 = "fffggghhhiii";
    charvar2 = "jjjkkklllmmm";
  end;
run;
  /* display results and related info */
proc print data=one;
run;
proc contents data=one;
run;
```

Here is output from PROC PRINT.

```
Obs   numvar     charvar1      charvar2

 1      100    fffggghhhiii     jjj
```

To help understand the result, here is output from PROC CONTENTS.

```
#    Variable    Type    Len    Pos
-----------------------------------
2    charvar1    Char     12      8
3    charvar2    Char      3     20
1    numvar      Num       8      0
```

Note that the length of CHARVAR1 is 12 and CHARVAR2 is 3. The length of a character variable is determined the first time it is used (“first usage”). Subsequent statements in the DATA step cannot change the length.

In the DATA step above, first usage of CHARVAR1 and CHARVAR2 is in the following statements. The lengths of "aaabbbcccddd", 12, and “eee”, 3, determine the lengths CHARVAR1 and CHARVAR2.

```
  charvar1 = "aaabbbcccddd";
  charvar2 = "eee";
```

To avoid this problem, use a LENGTH statement to create the variables. Code the LENGTH statement before other usage of the variables in the DATA step, so it is the first usage of the variables. Set the length of each variable to its largest possible value.

```
data one;
```

```
    length charvar1 $12 charvar2 $12;
    numvar = 100;
    charvar1 = "aaabbbcccddd";
    charvar2 = "eee";
    if numvar > 0 then
    do;
      charvar1 = "fffggghhhiii";
      charvar2 = "jjjkkklllmmm";
    end;
  run;
```

Here is output from PROC PRINT. CHARVAR2 now has the correct value.

```
  Obs    numvar     charvar1       charvar2

   1       100    fffggghhhiii   jjjkkklllmmm
```

Here is output from PROC CONTENTS. The length of CHARVAR2 is now 12.

```
  #     Variable     Type     Len     Pos
  -----------------------------------
  2     charvar1     Char      12       8
  3     charvar2     Char      12      20
  1     numvar       Num        8       0
```

3. TRUNCATION READING AND WRITING EXTERNAL FILES ON THE UNIX AND WINDOWS PLATFORMS.

On the UNIX and Windows platforms, the default record length of an external file is 256 bytes. Unless a longer record length is specified, the following is true.

When reading from an external file with INPUT statements, input values past column 256 are not read.

When writing to an external file with PUT statements, output values are not written past column 256.

Two reasons that new users encounter truncation problems are as follows.

The default record length is documented in operating system-specific documentation such as the *SAS Companion for UNIX Environments, Version 8*, which new users are unlikely to read.

No warning or error messages are written to the SAS log. Innocuous-looking notes that identify the problem are written to the SAS log, but many users ignore SAS log notes.

Example 1. Input values past column 256 are not read.

The UNIX file /my/external/file contains two records.

Record 1:
columns 1-90 have the letter “a”
columns 91-180 have the letter “b”
columns 181-270 have the letter “c”

Record 2:
columns 1-90 have the letter “d”
columns 91-180 have the letter “e”
columns 181-270 have the letter “f”

The user submits the following code.

```
  data one;
    infile '/my/external/file';
    input cv1 $90. cv2 $90. cv3 $90.;
  run;
```

Data set ONE has one observation, not two, as follows.

CV1 has 90 a’s, as expected.

CV2 has 90 b’s, as expected.

CV3 has 90 d’s, not 90 c’s as expected. The informat for CV3, $90, requests that 90 characters be read. Since the 90 characters beginning in column181 extend past column 256, they are not read. Instead, SAS goes to the next line, reads the 90 d’s in columns 1-90, and assigns them to CV3.

The SAS log contains the following notes. In this case, it’s easy to notice that the data set has only one observation. More generally, a user might not know how large the data set should be, and might not notice the smaller than expected number of observations. Other important information in the SAS log (maximum record length 256, lines were truncated), is easy to ignore.

```
NOTE: 2 records were read from the infile '/my/external/file'.
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.ONE has 1 observations and 3 variables.
```

Example 2. Output values are not written past column 256.

Data set ONE has two observations, as follows. CV1, CV2, and CV3 are all character variables of length 90.

Observation 1:
CV1 has 90 a’s
CV2 has 90 b’s
CV3 has 90 c’s

Observation 2:
CV1 has 90 d’s
CV2 has 90 e’s
CV3 has 90 f’s

To write data values to an external file, the following code is submitted.

```
data _null_;
  set one;
  file '/my/external/file2';
  put cv1 $90. cv2 $90. cv3 $90.;
run;
```

The expected result is as follows.

Record 1:
columns 1-90 have the letter “a”
columns 91-180 have the letter “b”
columns 181-270 have the letter “c”

Record 2:
columns 1-90 have the letter “d”
columns 91-180 have the letter “e”
columns 181-270 have the letter “f”

The external file actually has four records, as follows.

Record 1:
columns 1-90 have the letter “a”, as expected
columns 91-180 have the letter “b”, as expected

Record 2:
columns 1-90 have the letter “c”

Record 3:

columns 1-90 have the letter “d”
columns 91-180 have the letter “e”

Record 4:
columns 1-90 have the letter “f”

The c’s and f’s were not written to columns 181-270 of the first and third record, because that would extend past column 256. The SAS log contains the following notes, which a new user might well ignore.

```
NOTE: 4 records were written to the file '/my/external/file2'.
      The minimum record length was 90.
      The maximum record length was 180.
NOTE: There were 2 observations read from the data set WORK.ONE.
```

Prevent this problem as follows.

When reading from an external file, add the LRECL= option to the INFILE statement. Set the value to the largest possible length of a record. For example, change the INFILE statement in DATA step ONE to the following.

```
infile '/my/external/file' lrecl=270;
```

When writing to an external file, add the LRECL= option to the FILE statement. Set the value to the largest possible length of a record. For example, change the FILE statement in the DATA _NULL_ step above to the following.

```
file '/my/external/file2' lrecl=270;
```

An equivalent solution is to use the LRECL= option in a FILENAME statement that is referenced in a FILE or INFILE statement, as in the following.

```
filename blah '/my/external/file' lrecl=270;
```

When using the LRECL option to read from an external file, the following INFILE statement options might be necessary. See the INFILE statement documentation in the *SAS Language Reference, Version 8*, for details.

The PAD option might be needed if the records in the external file have different lengths. This option pads the records read from an external file with blanks so that the length of each record is the value of the LRECL= option.

The MISSOVER option might be needed if the last (right-most) field(s) of some records in the external file might be missing. If MISSOVER is set, when an INPUT statement reaches the end of the record, variables without values are set to missing. Otherwise, SAS reads subsequent records to look for missing fields.

4. READING EXTERNAL FILES WITH NON-DISPLAY CHARACTERS.

External files contain non-display characters, which are characters such as tabs, newlines, and page feeds that cannot be seen with standard editors or display commands.

Non-display characters in external files differ based on the platform. For example, lines are typically separated by a carriage return followed by a line feed (hexadecimal ‘0D0A’) on Windows and a line feed (hexadecimal ‘0A’) on UNIX. The SAS System reads external files correctly by accounting for non-display characters appropriate to the platform where it is executing.

Errors can occur when reading external files that contain inappropriate (not standard on the current platform) non-display characters. At the Federal Reserve Board, most errors occur in the SAS System for UNIX when reading files created on another platform, such as a spreadsheet exported to an external file on the Windows platform. Errors most commonly occur when reading

at one or more random locations

the last field of every record

a specific field of every record other than the last record

the last n fields (where n is some fixed number) of every record

The UNIX command dos2unix often fixes these errors. It converts Windows characters in an external file, including non-display characters, to UNIX characters. For example, the following command runs dos2unix on the file blah.txt, and writes the results to the file bling.txt. The original file is unchanged.

```
  dos2unix  blah.txt  bling.txt
```

At the Federal Reserve Board, dos2unix fixes about 95% of troublesome UNIX files. A detailed discussion of how to fix the remaining files is beyond the scope of this paper, but one approach is to take the following steps.

Edit the file with an editor such as Emacs that displays hexadecimal characters.

If a field cannot be read, look for inappropriate non-display characters in that field and the field immediately before it.

Convert the offending non-display characters to characters appropriate to the current platform. The examples below are entered at a UNIX prompt and execute Perl.

1. Convert all Windows carriage return line feeds (hexadecimal '0D0A') to UNIX line feeds (hexadecimal '0A') in the file z.txt. Overwrite the existing file without backup.

```
  perl -pi -e 's/\x0d\x0a/\x0a/g' z.txt
```

2. Convert all occurrences of hexadecimal '0D' to spaces (hexadecimal '20') in the file blah.txt. Overwrite the existing file, and back it up to blah.txt.bak.

```
  perl -pi.bak -e 's/\x0d/\x20/g' blah.txt
```

CONCLUSION

This paper reviewed some common mistakes made by SAS users. It showed how to fix the mistakes, and suggested why some of the mistakes occur so often. Hopefully, this will reduce the frequency of these mistakes in the future.

For more information, contact

Bruce Gilsen
Federal Reserve Board, Mail Stop 157
Washington, DC 20551
phone: 202-452-2494
e-mail: bruce.gilsen@frb.gov

REFERENCES

Gilsen, Bruce (2003), "Deja-vu All Over Again: Common Mistakes by New SAS Users," in the Proceedings of the Sixteenth Annual NorthEast SAS Users Group Conference, 16.

Grant, Paul (1993), "The 'SKIP' Statement," in the Proceedings of the Sixth Annual NorthEast SAS Users Group Conference, 6, 197-198.

SAS Institute Inc (1999), "SAS Companion for the Microsoft Windows Environment," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Companion for the OS/390 Environment," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Companion for UNIX Environments," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Language Reference: Concepts," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Language Reference, Version 8," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Macro Language: Reference, Version 8," Cary, NC: SAS Institute Inc.

SAS Institute Inc (1999), "SAS Procedures Guide, Version 8," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Donna Hill and Steve Taubman at the Federal Reserve Board, and Avi Peled at the Federal Reserve Bank of Philadelphia. Their support is greatly appreciated.