Paper 137-2009

Tales from the Help Desk 3: More Solutions for Common SAS® Mistakes

Bruce Gilsen, Federal Reserve Board

INTRODUCTION

In 20 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 common mistakes that occur during the following tasks, and shows how to fix them.

1. Removing duplicate observations with PROC SORT.

2. Incrementing a SAS date value with the INTNX function.

3. Reading variable length character fields in delimited text (CSV, TAB, and DLM) files.

4. Executing a system command in conditional code.

5. Doing calculations using BY variables from a MERGE statement.

In the context of reviewing these mistakes, the paper provides details about SAS system processing that can help users employ the SAS system more effectively. This paper is the third of its type; see the references for two previous papers that review other common mistakes.

1. Removing duplicate observations with PROC SORT.

In data set ONE, observations three and five are identical, as are observations four, six, and seven.

Obs	aa	bb	cc	dd
1	200	210	220	230
2	300	310	320	330
3	1	2	3	4
4	1	20	30	40
5	1	2	3	4
6	1	20	30	40
7	1	20	30	40

To sort by the variable AA and eliminate duplicate observations, PROC SORT is executed with the NODUPRECS option.

```
proc sort data=one out=two noduprecs;
  by aa;
run;
```

Data set TWO is expected to have four observations, but actually has six observations, as follows.

Data set TWO (expected)

Obs	aa	bb	cc	dd
1	1	2	3	4
2	1	20	30	40
3	200	210	220	230
4	300	310	320	330

Data set TWO (actual)

Obs	aa	bb	cc	dd
1	1	2	3	4
2	1	20	30	40
3	1	2	3	4
4	1	20	30	40
5	200	210	220	230
6	300	310	320	330

This problem occurs because NODUPRECS works as follows. First, the data set is sorted as usual. Then, as observations are written to the output DATA set, observations identical to the previous observation are eliminated. Identical observations that are not contiguous are not eliminated.

In this example, the observations are sorted as follows before NODUPRECS is applied.

```
Obs    aa     bb     cc     dd

 1      1      2      3      4
 2      1     20     30     40
 3      1      2      3      4
 4      1     20     30     40
 5      1     20     30     40
 6    200    210    220    230
 7    300    310    320    330
```

NODUPRECS eliminates observation five, which is identical to observation four. Observations one and three and observations two and four are identical but not contiguous and are all written to the output data set.

One solution is to use PROC SQL. Use the DISTINCT operator to remove duplicate observations and the ORDERBY clause to sort by AA, as in the following code.

```
proc sql noprint;
  create table two as
  select distinct *
  from one
  order by aa;
quit;
```

This code removes all duplicate observations, and data set TWO has four observations, as follows.

```
Obs    aa     bb     cc     dd

 1      1      2      3      4
 2      1     20     30     40
 3    200    210    220    230
 4    300    310    320    330
```

A second solution, noted in the *Base SAS 9.1.3 Procedures Guide* (*The SORT Procedure* chapter, in the description of NODUPRECS), is to sort on all variables. The four variables in this example could be easily hard-coded in the PROC SORT BY statement, but a large list of variables should be generated in the program. Compared to the first solution, PROC SORT should be faster but slightly more complicated (because of the need to generate the variable list).

In the following code, PROC SQL is used to read the list of variables in data set ONE from the DICTIONARY tables and create a macro variable, VAR_NAMES, containing a space-separated list of all variables except AA. Then, data set ONE is sorted by all variables. AA is omitted from the macro variable generated by PROC SQL and coded first in the BY statement to ensure that the data set is sorted by AA as intended.

```
proc sql noprint;
  select name into :var_names separated by ' '
  from dictionary.columns
  where upcase(libname)='WORK' and
        upcase(memname)='ONE' and
        upcase(name) ne 'AA';
quit;
proc sort data=one out=two noduprecs;
  by AA &var_names;
run;
```

To eliminate duplicate observations without regard to the order of observations in the new data set, simplify the code as follows.

- For the first solution (PROC SQL), omit the ORDER BY clause.
- For the second solution (PROC SORT), omit the PROC SQL step and change the PROC SORT BY statement to the following.

```
by _all_;
```

2. Incrementing a SAS date value with the INTNX function.

The data step function INTNX returns a SAS date value incremented by a specified number of intervals (days, weeks, months, quarters, years, etc.).

In the following DATA step, DATE1 is set to the SAS date value for October 18, 2005, which is 16,727. INTNX is used to increment the date by two days, two months, and two years. DATEPLUS2DAY has the expected value, but DATEPLUS2MONTH and DATEPLUS2YEAR do not.

```
data one;
  date1 = '18oct2005'd;
  dateplus2day = intnx('day',date1,2);      * want to increment by 2 days;
  dateplus2month = intnx('month',date1,2);  * want to increment by 2 months;
  dateplus2year = intnx('year',date1,2);    * want to increment by 2 years;
run;
```

		Expected		Actual	
Variable	Description	Value	SAS date	Value	SAS date
dateplus2day	2 days after 10/18/2005	16729	10/20/2005	16729	10/20/2005
dateplus2month	2 months after 10/18/2005	16788	12/18/2005	16771	12/1/2005
dateplus2year	2 years after 10/18/2005	17457	10/18/2007	17167	1/1/2007

To help understand this problem, here is a somewhat informal and incomplete review of the syntax of INTNX. Information is provided for SAS date values, but not for datetime and time values, and sub-arguments to the interval value are omitted. For complete syntax, see the *SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3*.

INTNX has three required arguments and one optional argument, commonly used as follows for SAS date values.

INTNX(*interval*, *start-from*, *increment* <,*alignment*>);

- *interval* is the unit of measure (days, weeks, months, quarters, years, etc.) by which *start-from* is incremented.
- *start-from* is a SAS date value to be incremented.
- *increment* is the integer number of intervals by which *start-from* is incremented (negative values = earlier dates).
- *alignment* is where *start-from* is aligned within *interval* before being incremented. Possible values are BEGINNING, MIDDLE, END, and (new in Version 9) SAMEDAY. This argument is optional, and defaults to BEGINNING.

INTNX's default *alignment* is BEGINNING, so by default *start-from* is aligned to the beginning of the period before being incremented. This leads to unexpected results for intervals other than DAY, as in the examples from the previous DATA step. As before, DATE1 is the SAS date value for October 18, 2005.

1. dateplus2day = intnx('day',date1,2);

   SAS first aligns to the start of October 18, 2005 (the beginning of the interval, DAY), which has no effect, then increments by two days. DATEPLUS2DAY is 16,729, the SAS date value for October 20, 2005, as expected.

2. dateplus2month = intnx('month',date1,2);

   SAS first aligns to October 1, 2005 (the beginning of the interval, MONTH), then increments by two months. The result is 16,771, the SAS date value for December 1, 2005.

3. dateplus2year = intnx('year',date1,2);

   SAS first aligns to January 1, 2005 (the beginning of the interval, YEAR), then increments by two years. The result is 17,167, the SAS date value for January 1, 2007.

In Version 9, a new alignment value, SAMEDAY, was added. SAMEDAY preserves the SAS date value's alignment within the interval before it is incremented, generating the expected results. To prevent the problem shown in this example, always set alignment to SAMEDAY for intervals other than DAY (when interval is DAY, SAMEDAY is not necessary).

Here are the examples from the previous DATA step with the SAMEDAY argument added. As before, DATE1 is the SAS date

value for October 18, 2005.

```
SAS Statement                                        Description                Value SAS date

dateplus2day=intnx('day',date1,2,"sameday");         2 days after 10/18/2005    16729 10/20/2005
dateplus2month=intnx('month',date1,2,"sameday");     2 months after 10/18/2005  16788 12/18/2005
dateplus2year=intnx('year',date1,2,"sameday");       2 years after 10/18/2005   17457 10/18/2007
```

Here are additional examples for some interesting dates. Note that 2000 and 2004 but not 2003 are leap years.

```
SAS Statement                                        Description                Value  SAS date

date2=intnx('year','29feb2000'd,1,"sameday");        1 year after 2/29/2000     15034  2/28/2001
date3=intnx('year','29feb2000'd,4,"sameday");        4 years after 2/29/2000    16130  2/29/2004
date4=intnx('month','31mar2003'd,-1,"sameday");      1 month before 3/31/2003   15764  2/28/2003
date5=intnx('month','31mar2004'd,-1,"sameday");      1 month before 3/31/2004   16130  2/29/2004
```

Note that until SAS Version 9.2, SAMEDAY should only be used with single, non-shifted date intervals (DAY, WEEK, WEEKDAY, TENDAY, SEMIMONTH, MONTH, QTR, SEMIYEAR, YEAR), because the following intervals might return the wrong answer with no error or warning.

- multiple (e.g., month2 = two-month interval)
- shifted (e.g., month.4 = month interval starting on April 1)
- time
- datetime

3. Reading variable length character fields in delimited text (CSV, TAB, and DLM) files.

A comma-separated values (CSV) file contains 31 records and 3 columns. The first row contains column names: "company,i,itimes10". The other rows are filled as follows: the first column has 10 records with "ibm", then 10 records with "aol", and then 10 records with "microsoft". The second column has the numbers 1-30, and the third column has the numbers 10, 20, ..., 290, 300. Here is the CSV file.

```
company,i,itimes10
ibm,1,10
ibm,2,20
ibm,3,30
ibm,4,40
ibm,5,50
ibm,6,60
ibm,7,70
ibm,8,80
ibm,9,90
ibm,10,100
aol,11,110
aol,12,120
aol,13,130
aol,14,140
aol,15,150
aol,16,160
aol,17,170
aol,18,180
aol,19,190
aol,20,200
microsoft,21,210
microsoft,22,220
microsoft,23,230
microsoft,24,240
microsoft,25,250
microsoft,26,260
microsoft,27,270
microsoft,28,280
microsoft,29,290
microsoft,30,300
```

The CSV file was read with PROC IMPORT.

```
proc import
  out=csvin
  datafile = '/mydir/csv1.csv'
  dbms=csv;
  getnames=yes;
  datarow=2;
run;
```

In data set CSVIN, COMPANY has been truncated from “microsoft” to “mic” in observations 21-30.

Obs	company	i	itimes10
1	ibm	1	10
2	ibm	2	20
3	ibm	3	30
4	ibm	4	40
5	ibm	5	50
6	ibm	6	60
7	ibm	7	70
8	ibm	8	80
9	ibm	9	90
10	ibm	10	100
11	aol	11	110
12	aol	12	120
13	aol	13	130
14	aol	14	140
15	aol	15	150
16	aol	16	160
17	aol	17	170
18	aol	18	180
19	aol	19	190
20	aol	20	200
21	mic	21	210
22	mic	22	220
23	mic	23	230
24	mic	24	240
25	mic	25	250
26	mic	26	260
27	mic	27	270
28	mic	28	280
29	mic	29	290
30	mic	30	300

This problem occurs because by default, the Import Wizard, PROC IMPORT, and the External File Interface (EFI) scan the first 20 records to determine variable attributes such as field length when reading delimited text (CSV, TAB, and DLM) files. If character fields have longer values past the first 20 records, they are truncated by PROC IMPORT, the Import Wizard, and the EFI.

Before Version 9.1, this problem could only be prevented by manually updating the SAS registry. Starting in Version 9.1, the PROC IMPORT GUESSINGROWS= option tells SAS how many records to scan to determine variable attributes, and comparable methods are provided by the Import Wizard and the EFI.

The following code tells SAS to scan the first 16,000 records, and reads the CSV file correctly.

```
proc import
  out=csvin
  datafile = '/mydir/csv1.csv'
  dbms=csv;
  getnames=yes;
  guessingrows=16000;
  datarow=2;
run;
```

One caution is that for large files, GUESSINGROWS significantly increases execution time, as shown by the following test results. A 16,000 observation data set was read with SAS Version 9.1.3 for Linux. The first two tests were with the CSV file

and PROC IMPORT code used in this section. For the third and fourth test, 32 additional integer numeric columns were added to the CSV file.

```
Number of variables   GUESSINGROWS value   CPU seconds

 3                    not specified          .11
 3                    16000                 5.92
35                    not specified          .31
35                    16000                50.63
```

Note that the CSV file used in this section could be created by the following SAS code.

```
data one;
  length company $10;
  do i=1 to 30;
    itimes10 = i * 10;
    if i le 10 then company="ibm";
      else if i le 20 then company="aol";
      else company="microsoft";
    output;
  end;
run;
proc export
  data=one
  outfile = '/mydir/csv1.csv'
  dbms=csv;
run;
```

4. Executing a system command in conditional code.

The following DATA step is part of a daily Linux application. The objective is to make a backup copy of a file every Sunday, but the file is copied every day instead. Note that the TODAY function returns the current date as a SAS date value, the WEEKDAY function returns the day of the week for a SAS date value (1=Sunday, 2=Monday, ... 7=Saturday), and cp is the Linux command to copy a file.

```
data one;
  if weekday(today()) = 1 then do;   * true on Sundays;
    x 'cp important_file important_file.bak';
  end;
  /* more SAS code */
run;
```

To understand this problem, it is helpful to understand a little about DATA step processing. SAS processes a DATA step in two stages: it is first compiled, then executed. Compilation includes the following tasks.

- Check the syntax of the statements and convert them to machine code.
- Do initial DATA step setup (for example, create the program data vector and determine data set and variable attributes).
- Execute global statements, which include DM, ENDSAS, FILENAME, FOOTNOTE, %INCLUDE, LIBNAME, ODS, OPTIONS, PAGE, RUN, TITLE, and X statements. Global statements are listed in the *SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3* (*Statements* chapter, *Global Statements* section).
- Execute DATA step declarative statements, which are statements that SAS only executes once for the DATA step, not once per DATA step iteration. Declarative statements are listed in the *SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3* (*Statements* chapter, *Executable and Declarative Statements* section) and include ARRAY (array definition, not array reference), ATTRIB, BY, CARDS, CARDS4, DATA, DATALINES, DATALINES4, DROP, END, FORMAT, INFORMAT, KEEP, LABEL, Statement labels, LENGTH, RENAME, RETAIN, WHERE, and WINDOW.

This problem occurs because in a DATA step, global statements and declarative statements are processed during compilation and always execute, even if they are in conditional code (such as an IF statement). SAS determines if the conditional code is true or false later, during step execution.

To prevent this problem, do not use global statements or declarative statements in conditional code in a DATA step. In this case, one solution is to use the CALL SYSTEM routine instead of the X statement. CALL SYSTEM is similar to X, but executes during DATA step execution, so it can be used with conditional code. The following DATA step only copies the file on Sundays as intended.

```
data one;
  if weekday(today()) = 1 then do;   * true on Sundays;
    call system ('cp important_file important_file.bak');
  end;
  /* more SAS code */
run;
```

More generally, an execution-time equivalent to a global statement or declarative statement is not available. In that case, one solution is to conditionally *generate* the code in a macro rather than conditionally *execute* the code in a DATA step. This is illustrated in the following macro, CONDCODE.

```
%macro condcode;
  %if &sysday = Sunday %then %do;
    x 'cp important_file important_file.bak';
  %end;
  data one;
    /* more SAS code */
  run;
%mend condcode;
%condcode;
```

The following code is generated on Monday - Saturday.

```
  data one;
    /* more SAS code */
  run;
```

The following code is generated on Sunday.

```
  x 'cp important_file important_file.bak';
  data one;
    /* more SAS code */
  run;
```

Note the following about macro CONDCODE.

- The SYSDAY automatic macro variable contains the day of the week (Sunday, Monday, ..., Saturday, with the first letter in upper case) that the SAS session began executing. For multi-day SAS sessions, SYSDAY differs from the DATA step function TODAY, which returns the current date. For multi-day SAS sessions, change the first line of CONDCODE to the following.

  ```
  %local week_day;
  %let week_day = %sysfunc(weekday(%sysfunc(today())));
  %if &week_day = 1 %then %do;
  ```

  %SYSFUNC executes SAS functions from within a macro. Functions cannot be nested within %SYSFUNC, so %SYSFUNC is used once for the WEEKDAY function and once for the TODAY function.

- X, a declarative statement, can be used because macro CONDCODE only generates the X statement when the &IF statement is true, on Sunday.

- The conditional processing of the X statement is unrelated to the DATA step code, so the %IF, X, and %END statements are coded before the DATA statement. The desired result is also generated if these three statements follow the DATA statement.

5. Doing calculations using BY variables from a MERGE statement.

Data set ONE contains the following values.

```
Obs    state   income

 1       1        10
 2       1        20
 3       1        30
 4       2       100
 5       2       200
 6       2       300
 7       3      1000
 8       3      2000
 9       3      3000
```

In the following code, the mean for each state is calculated with PROC MEANS, and difference from mean income in each observation is calculated in a DATA step.

```
proc means data = one noprint;
  by state;
  var income;
  output out=two (drop= _type_ _freq_) mean = income_mean;
run;

data three;
  merge one two ;
  by state;
  income_diff_from_mean = income - income_mean;
run;
```

Data set TWO contains the mean income for each state.

```
Obs    state   income_mean

 1       1          20
 2       2         200
 3       3        2000
```

Data set THREE contains the following values.

```
Obs    state   income   income_mean   income_diff_from_mean

 1       1        10         20               -10
 2       1        20         20                 0
 3       1        30         20                10
 4       2       100        200              -100
 5       2       200        200                 0
 6       2       300        200               100
 7       3      1000       2000             -1000
 8       3      2000       2000                 0
 9       3      3000       2000              1000
```

Now, the code is changed slightly. Rather than create the new variable INCOME_DIFF_FROM_MEAN, the difference of INCOME and INCOME_MEAN is assigned to INCOME_MEAN.

```
data three;
   merge one two ;
   by state;
   income_mean = income - income_mean;
run;
```

Data set THREE contains the following values. In observations 2, 3, 5, 6, 8, and 9, INCOME_MEAN does not have the expected value.

```
Obs    state   income   income_mean   income_mean that was expected

 1       1        10        -10               -10
 2       1        20         30                 0
 3       1        30          0                10
 4       2       100       -100              -100
```

```
5       2       200         300              0
6       2       300           0            100
7       3      1000       -1000          -1000
8       3      2000        3000              0
9       3      3000           0           1000
```

This problem results from what might be called the "MERGE statement retain rule." As explained in the *SAS 9.1.3 Language Reference: Concepts* (*Reading, Combining, and Modifying SAS Data Sets* chapter, *Match-Merging* section, Example 2),

> "When SAS reads the last observation from a BY group in one data set, SAS retains its values in the program data vector for all variables that are unique to that data set until all observations for that BY group have been read from all data sets."

Let's see how the first several observations of data set THREE are generated.

Generating observation 1 of data set THREE.

- The MERGE statement encounters a new BY group value, STATE=1.
- INCOME is read from the first observation of data set ONE and set to 10.
- INCOME_MEAN is read from the first observation of data set TWO and set to 20.
- INCOME_MEAN = INCOME - INCOME_MEAN is calculated. It is 10-20 = -10, as expected.

Generating observation 2 of data set THREE.

- The MERGE statement encounters the same BY group value, STATE=1.
- INCOME is read from the second observation of data set ONE and set to 20.
- All observations have been read from data set TWO for the current BY group, so as per the “MERGE statement retain rule”, INCOME_MEAN is retained and not re-read from data set TWO This is the source of the unexpected results; some users expect the value of INCOME_MEAN to be re-read from data set TWO (and equal 20). But, INCOME_MEAN’s value is -10, as calculated in the first observation and retained.
- INCOME_MEAN = INCOME - INCOME_MEAN is calculated. It is 20 - -10 = 30, not 20 - 20 = 0 as expected.

Generating observation 3 of data set THREE.

- The MERGE statement encounters the same BY group value, STATE=1.
- INCOME is read from the third observation of data set ONE and set to 30.
- All observations have been read from data set TWO for the current BY group, so as per the “MERGE statement retain rule”, INCOME_MEAN is retained and not re-read from data set TWO. As in the previous observation, some users expect the value of INCOME_MEAN to be re-read from data set TWO (and equal 20). But, INCOME_MEAN’s value is 30, as calculated in the second observation and retained.
- INCOME_MEAN = INCOME - INCOME_MEAN is calculated. It is 30 - 30 = 0, not 30 - 20 = 10 as expected.

Generating observation 4 of data set THREE.

- The MERGE statement encounters a new BY group value, STATE=2.
- INCOME is read from the fourth observation of data set ONE and set to 100.
- INCOME_MEAN is read from the second observation of data set TWO and set to 200.
- INCOME_MEAN = INCOME - INCOME_MEAN is calculated. It is 100-200 = -100, as expected.

Here is one way to correctly calculate INCOME - INCOME_MEAN and store the result in INCOME_MEAN. The variable whose value is retained in observations 2 and 3 is now called INCOME_MEAN_TEMP, so retained values are not overwritten

when INCOME_MEAN is calculated.

```
data three;
   merge one two (rename = (income_mean = income_mean_temp)) ;
   by state;
   drop income_mean_temp;
   income_mean = income - income_mean_temp;
run;
```

CONCLUSION

This paper reviewed and showed how to fix some common mistakes made by SAS users, and, in the context of discussing these mistakes, provided details about SAS system processing. It is hoped that reading this paper enables users to better understand SAS system processing and thus employ the SAS system more effectively in the future.

For more information, contact the author, Bruce Gilsen, by mail at Federal Reserve Board, Mail Stop 157, Washington, DC 20551; by e-mail at bruce.gilsen@frb.gov; or by phone at 202-452-2494.

REFERENCES

Gilsen, Bruce (2003), "Deja-vu All Over Again: Common Mistakes by New SAS Users," *Proceedings of the Sixteenth Annual NorthEast SAS Users Group Conference*. <http://www.nesug.org/html/Proceedings/nesug03/bt/bt010.pdf>

Gilsen, Bruce (2006), "Improve Your Dating: The INTNX Function Alignment Value SAMEDAY," *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. <http://www2.sas.com/proceedings/sugi31/027-31.pdf>

Gilsen, Bruce (2004), "More Tales from the Help Desk: Solutions for Simple SAS Mistakes," *Proceedings of the Seventeenth Annual NorthEast SAS Users Group Conference*. <http://www.nesug.org/html/Proceedings/nesug04/pm/pm11.pdf>

SAS Institute Inc. (2004), "*Base SAS 9.1.3 Procedures Guide*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004), "*SAS 9.1.3 Language Reference: Concepts*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004), "*SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2005), SAS Note 001075, "How to scan more than 20 records to determine variable attributes in EFI." http://support.sas.com/techsup/unotes/SN/001/001075.html.

SAS Institute Inc. (2006), SAS Note 016184, "INTNX function with SAMEDAY alignment does not support multiple, shifted, time, or datetime intervals." http://support.sas.com/techsup/unotes/SN/016/016184.html.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Heidi Markovitz and Donna Hill at the Federal Reserve Board, Bryan Beverly at the Bureau of Labor Statistics, and Mike Rhoads at Westat. Their support is greatly appreciated.

TRADEMARK INFORMATION