

Paper PA-105

Tales from the Help Desk 7: Solutions to Common SAS ® Tasks

Bruce Gilsen, Federal Reserve Board, Washington, DC

ABSTRACT

In 30 years as a SAS ® consultant at the Federal Reserve Board, questions about some common SAS tasks seem to surface again and again. This paper collects some of these common questions, and provides code to resolve them. The following tasks are reviewed.

1. Save and restore SAS option values.
2. Pad multiple non-continuous time series with missing values to make continuous time series.
3. Read a SAS data set backwards (last observation to first).
4. Create a data set containing the last 5 observations of an existing data set.
5. Drop the last n observations in each BY group.
6. Write data to multiple external files in a DATA step, determining file names dynamically from data values.
7. Compare observations in the same data set.

In the context of discussing these tasks, the paper provides details about SAS system processing that can help users employ the SAS system more effectively. This paper is the seventh of its type; see the references for six previous papers.

1. Save and restore SAS option values

If your application changes the value of SAS options, good programming practice is to save existing option values at the start of the application and restore them at the end. You can use the following procedures.

- PROC OPTSAVE saves current SAS system option settings in the SAS registry or a SAS data set.
- PROC OPTLOAD reads SAS system option settings stored in the SAS registry or a SAS data set and puts them into effect.

For example, suppose our application needs to change the LINESIZE and PAGESIZE options, which can be abbreviated to LS and PS, respectively.

At the start of the application, save SAS system option settings to WORK data set _OPTSAVE. In Linux SAS 9.3, _OPTSAVE has 256 observations and two variables (OPTNAME and OPTVALUE). The number of observations varies based on the version and operating system.

```
proc optsave out=_optsave; run;
```

Now, we can modify the LINESIZE and PAGESIZE values.

```
options ls=120 ps=500;
```

We can verify that the option values changed as follows.

```
proc options option=(ls ps); run;
```

At the end of the application, we can restore the old option values.

```
proc optload data=_optsave; run;
```

Another option is to selectively restore option values. For example, restore only the LINESIZE value as follows.

```
proc optload data=_optsave (where = (optname="LINESIZE")); run;
```

2. Pad multiple non-continuous time series with missing values to make continuous time series

Data set ONE has data as follows.

Obs	date	income	invest	gov	taxes
1	19950421	1	2	3	4
2	19950424	5	6	7	8
3	19950425	9	10	11	12
4	19950428	13	14	15	16
5	19950502	17	18	19	20
6	19950503	21	22	23	24
7	19950509	25	26	27	28

- DATE is a SAS date displayed with the YYMMDDN8. format.
- INCOME, INVEST, GOV, and TAXES are numeric variables.
- The date interval for this data is WEEKDAY. April 21, 1995 is a Friday. The missing observations that we need to insert are as follows:
 - o 2 dates are missing after Tuesday, April 25.
 - o 1 date is missing after Friday, April 28.
 - o 3 dates are missing after Wednesday, May 3.
- DATE is assumed to be in ascending order; if not, sort the data by DATE at the beginning of the code.

First, read the DICTIONARY.COLUMNS DICTIONARY table and create two macro variables.

- NUMERIC_VARS contains the names of all numeric variables in data set ONE other than DATE, space-separated.
- NUM_NUMERIC_VARS contains the number of numeric variables other than DATE. It is copied from the automatic macro variable SQLOBS, which contains the number of rows selected by the last PROC SQL statement.

In this example, the macro variables have the following values:

```
NUMERIC_VARS:      income invest gov taxes
NUM_NUMERIC_VARS:  4
```

```
proc sql noprint;
  select name
  into :numeric_vars separated by ' '
  from dictionary.columns
  where libname="WORK"
  and memname="ONE"
  and type = "num"
  and upcase(name) ne "DATE";
%let num_numeric_vars = &sqlobs;
```

```
quit;
```

Now, create the data set padded with missing values as needed.

The following statement below tests if the date from the current and previous observations are contiguous. It returns the number of intervals between two SAS dates for a given frequency.

```
interval = intck('weekday',prevdate, date);
```

- If INTERVAL = 1, the current and previous observation are contiguous, and the current observation is output without any modification.
- If INTERVAL is greater than 1, then the number of missing periods of data is one less than the value of INTERVAL (one period of data is missing if INTERVAL = 2, two periods are missing if INTERVAL = 3, and so on). In this case, we do the following.
 - o Save the data values to a temporary array.
 - o Shift DATE to the beginning of the missing period(s) with the INTNX function.
 - o Pad the data set with additional observations in which the value of numeric variables other than DATE are set to missing.
 - o Restore the original data values and output them.

For example, if DATE is the SAS date for 19950502 (and PREVDATE is thus the SAS date for 19950428), INTERVAL = 2, so one date is missing.

```
data two (keep=date &numeric_vars);
  set one;
  retain prevdate; /* value of DATE in the previous observation */

  /* Array with all numeric variables except DATE */
  array all_numeric (*) &numeric_vars;
  /* Temporary array that is the same size as ALL_NUMERIC */
  array all_numeric_temp (&num_numeric_vars) _temporary_;

  /* Compare current date to previous date to see if there is a gap */
  interval = intck('weekday',prevdate,date);

  if interval > 1 then do ;

    /* Save value of numeric variables in temporary array. Set numeric variables
       to missing prior to outputting the missing periods. */
    do i = 1 to dim(all_numeric);
      all_numeric_temp(i) = all_numeric(i);
      all_numeric(i)=.;
    end;

    /* Shift DATE to beginning of missing period(s) */
    date=intnx('weekday',date,-interval+1,"sameday");

    /* Write observations for missing periods */
    do i = 1 to interval-1;
      output;
      date=intnx('weekday',date,1,"sameday"); /* increment date */
    end;

    /* Restore values of numeric variables for current observation */
```

```

do i = 1 to dim(all_numeric);
  all_numeric(i) = all_numeric_temp(i);
end;

end; /* of if interval > 1 then do ; */

/* Write out current observation.*/
output;

/* Save current date for next observation */
prevdate = date;
run;

```

Data set TWO has the following data.

Obs	date	income	invest	gov	taxes
1	19950421	1	2	3	4
2	19950424	5	6	7	8
3	19950425	9	10	11	12
4	19950426	.	.	.	.
5	19950427	.	.	.	.
6	19950428	13	14	15	16
7	19950501	.	.	.	.
8	19950502	17	18	19	20
9	19950503	21	22	23	24
10	19950504	.	.	.	.
11	19950505	.	.	.	.
12	19950508	.	.	.	.
13	19950509	25	26	27	28

This example used a date interval of WEEKDAY but the same code would work for other intervals.

An alternate method suggested by Heidi Markovitz is as follows.

- Create a “date list” data set, `_ONE_DATE_RANGE`, for the given date interval (WEEKDAY in this example), with one record for each date between first and last date in the data set. In this example, the data set will have 18 observations, with variable DATE containing WEEKDAY dates from April 21, 1995 to May 10, 1995.
- Combine the real data set and date list data set, keeping only one record for each date.
 - o For non-missing periods, select the observation from the real data set.
 - o For missing periods, select the observation from the date list data set. All variables other than DATE will be missing.

Data sets ONE (repeated from above) and `_ONE_DATE_RANGE` have data as follows. In the final DATA step, all observations from ONE and observations marked in red from `_ONE_DATE_RANGE` are selected.

ONE						<code>_ONE_DATE_RANGE</code>	
Obs	date	income	invest	gov	taxes	Obs	date
1	19950421	1	2	3	4	1	19950421
2	19950424	5	6	7	8	2	19950424
3	19950425	9	10	11	12	3	19950425
4	19950428	13	14	15	16	4	19950426
5	19950502	17	18	19	20	5	19950427

6	19950503	21	22	23	24	6	19950428
7	19950509	25	26	27	28	7	19950501
						8	19950502
						9	19950503
						10	19950504
						11	19950505
						12	19950508
						13	19950509

```

/* Create macro variables FIRST_DATE and LAST_DATE with the
   earliest and latest dates in the data set */
proc sql noprint;
  select min(date), max(date)
  into :first_date, :last_date
  from one;
quit;

/* Create "date list" data set with required date range */
data _one_date_range;
  date=&first_date;
  do while (date le &last_date);
    output;
    date = intnx('weekday', date, 1);
  end;
run;

/* Combine real data set and date list data set.
   Keep only one record for each date */
data three;
  set one
      _one_date_range;
  by date;
  if first.date;
run;

```

3. Read a SAS data set backwards (last observation to first)

Normally, SAS reads observations sequentially (first to last) in the DATA step. The POINT= option causes SAS to read with direct access: only the observations specified with POINT= are read, and we can read the data set from last to first.

The following is true when POINT= and NOBS= are used to read a data set with direct access.

- The end-of-file indicator is not read, so a STOP statement is needed to prevent an infinite loop.
- The clause NOBS=_N in the SET statement causes SAS at compilation time (before the DATA step executes) to set the variable _N to the number of observations in the data set. _N is thus available in the DO statement. In this example, the data set has 5 observations. _N is 5, and the DO statement below resolves as follows.

```

do _i = _n to 1 by -1;      resolves to
do _i = 5 to 1 by -1;

```

- The POINT and NOBS variables, _I and _N, are not included in data set TWO, so they need not be dropped.

- The implicit output statement normally in effect at the end of a DATA step does not take place, so an explicit OUTPUT statement is needed to write observations to the new data set.

Data set ONE has the following data.

Obs	num1
1	1
2	2
3	3
4	4
5	5

The following code reads data set ONE backward.

```
data two;
  do _i = _n to 1 by -1;
    set one point=_i nobs=_n;
    num1times2 = num1 * 2; /* Create a new variable */

    /* Optionally, do other things here */

    /* Write the current observation to the new data set, TWO */
    output;
  end;
  /* STOP is needed with POINT and NOBS to prevent an infinite loop! */
  stop;
run;
```

Data set TWO has the following data.

Obs	num1	num1times2
1	5	10
2	4	8
3	3	6
4	2	4
5	1	2

4. Create a data set containing the last 5 observations of an existing data set

As described in the previous section, we can use the following.

- The POINT= option does a direct access read of the last 5 observations.
- The NOBS= option determines the number of observations at compilation time.
- OUTPUT; writes observations to the output data set.
- STOP; ends the DATA step and prevents an infinite loop.

If there are less than 5 observations in the original data set, we want to read the entire data set. We handle this possibility with the MAX function.

In this example, data set ONE has 8 observations, so `_N` is 8 and the DO statement below resolves as follows.

```
do _i = max(_n-4, 1) to _n;    which resolves to
do _i = max(8-4, 1) to 8;      which resolves to
do _i = max(4, 1) to 8;        which resolves to
do _i = 4 to 8;
```

Data set ONE has the following data.

Obs	num1	num2
1	1	11
2	2	22
3	3	33
4	4	44
5	5	55
6	6	66
7	7	77
8	8	88

The following code creates data set TWO containing the last 5 observations of data set ONE.

```
data two;
  do _i = max(_n-4, 1) to _n;
    if _i > 0 then do;
      set one nobs=_n point=_i;
      output;
    end;
  end;
  stop;
run;
```

Data set TWO has the following data.

Obs	num1	num2
1	4	44
2	5	55
3	6	66
4	7	77
5	8	88

5. Drop the last n observations in each BY group

We want to drop the last n observations in each BY group, and drop all the observations for BY groups with n or less observations. We can code a simple macro with four parameters, as follows.

Parameter	Meaning
BYVAR	The BY variable
NUMTODROP	The number of trailing observations to drop in each BY group
IN	The input data set
OUT	The output data set

The code below includes the following steps.

- PROC MEANS calculates how many observations are in each BY group.
- Merge the PROC MEANS output with the original data and use it to determine which observations are written to the output data set.

```
%macro droplastinby(byvar=, numtodrop=, in=, out=);

    /* Sort the data */
    proc sort data = &in;
        by &byvar;
    run;

    /* PROC MEANS outputs number of observations in each BY group */
    proc means data = &in noprint;
        by &byvar;
        var &byvar;
        output out=&in.means (drop= _type_ _freq_) n = n_&byvar;
    run;

    data &out;
        retain n &numtodrop; /* How many to drop at end of BY groups */
        retain count ;      /* Count down from N_&BYVAR to count
                               how many in BY group so far */
        drop n count n_&byvar;
        merge &in _&in.means;
        by &byvar;
        if first.&byvar then count = n_&byvar;
                           else count = count - 1;
        if count gt n;
    run;

    /* Print final data set */
    proc print data = &out;
    run;

%mend droplastinby;
```

In this example, data set ONE has the following data, and we want to drop the last 2 observations in each BY group for BY variable KEYVAR and write the output to data set ONEDROPLAST.

Obs	keyvar	othervar
1	1	11
2	1	22
3	1	33
4	1	44
5	4	111
6	2	55
7	2	66
8	3	77
9	3	88
10	3	99

```
/* Invoke macro, specify number trailing observations to drop in each BY group */
%droplastinby(byvar=keyvar, numtodrop=2, in=one, out=onedroplast)
```

Data set _ONEMEANS with PROC MEANS output has the following data.

Obs	keyvar	n_keyvar
1	1	4
2	2	2
3	3	3
4	4	1

Data set ONEDROPLAST with the final results has the following data. Since there are two or less values of KEYVAR=2 and KEYVAR=4, these BY groups are dropped entirely from the output.

Obs	keyvar	othervar
1	1	11
2	1	22
3	3	77

6. Write data to multiple external files in a DATA step, determining file names dynamically from data values

Data set ONE has the following data.

Obs	bankid	income
1	bank1	10
2	bank1	20
3	bank3	80
4	bank3	90
5	bank3	100
6	bank2	40
7	bank2	50

We want to write data values to multiple output files in the Linux directory /my/home/directory/out1. The file names and records to write to those files are determined by the value of the BY variable, BANKID, for example:

Value of BANKID	File that the record is written to
bank1	/my/home/directory/out1/bank1
bank2	/my/home/directory/out1/bank2

Linux files are used in this example, but the code can be used on other platforms by changing the file names to platform-appropriate values.

The code below includes the following steps.

- Sort by BANKID.
- Create a retained character variable, FILEID, that contains the name of the current output file. The length of FILEID must be large enough to hold the longest possible file name.
- Change FILEID by appending the current value of BANKID to the directory path whenever the value of BANKID changes.
- Use a FILE statement that references the current output file with the clause FILEVAR=FILEID. The FILE statement is an executable statement that must be executed in every observation, so it must not be within the IF/THEN statement. The FILE statement includes a placeholder file specification, TEST, that is ignored

but required – an error occurs if it is omitted.

```

/* Sort the data set */
proc sort data=one out=two;
  by bankid;
run;

data three ;
  set two;
  by bankid;
  /* FILEID must be large enough to hold the longest possible file name */
  length fileid $40;
  retain fileid;

  /* Whenever BANKID changes, the output file changes */
  if first.bankid then do;
    fileid = "/my/home/directory/out1/" || bankid;
  end;

  /* The FILE statement executes in every observation and includes a
     placeholder file specification, TEST, that is ignored, but required */
  file test filevar=fileid;
  put income; /* write to an external file */
run;

```

This program creates three external files with the following values.

```

/my/home/directory/out1/bank1:
    10
    20

/my/home/directory/out1/bank2:
    40
    50

/my/home/directory/out1/bank3:
    80
    90
   100

```

7. Compare observations in the same data set

Users might know that PROC COMPARE compares observations in two data sets, but might not realize that it can also compare observations in the same data set that are any fixed distance (number of observations) apart.

The following code compares adjacent observations of data set ONE.

```

proc compare base =one
  comp = one (firstobs = 2);
run;

```

The following code compares observations of data set ONE that are two observations apart.

```

proc compare base =one
  comp = one (firstobs = 3);

```

```
run;
```

CONCLUSION

This paper provided SAS code for some simple, common SAS programming tasks, and in the context of discussing these tasks, 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.

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/Proceedings/nesug03/bt/bt010.pdf>>

Gilsen, Bruce (2007), "More Tales from the Help Desk: Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2007 Conference*. <<http://www2.sas.com/proceedings/forum2007/211-2007.pdf>>

Gilsen, Bruce (2009), "Tales from the Help Desk 3: More Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2009 Conference*. <<http://support.sas.com/resources/papers/proceedings09/137-2009.pdf>>

Gilsen, Bruce (2010), "Tales from the Help Desk 4: Still More Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2010 Conference*. <<http://support.sas.com/resources/papers/proceedings10/146-2010.pdf>>

Gilsen, Bruce (2012), "Tales from the Help Desk 5: Yet More Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2012 Conference*. <<http://support.sas.com/resources/papers/proceedings12/190-2012.pdf>>

Gilsen, Bruce (2015), "Tales from the Help Desk 6: Solutions to Common SAS Tasks," *SESUG 2015: The Proceedings of the SouthEast SAS Users Group, Myrtle Beach, SC, 2015*.

SAS Institute Inc. (2012), "*Base SAS 9.3 Procedures Guide, Second Edition*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2011), "SAS 9.3 Language Reference by Name, Product, and Category," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2011), "SAS 9.3 Macro Language: Reference," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Heidi Markovitz and Donna Hill at the Federal Reserve Board. Their support is greatly appreciated.

CONTACT INFORMATION

For more information, contact the author, Bruce Gilsen, by mail at Federal Reserve Board, Mail Stop N-122, Washington, DC 20551; by e-mail at bruce.gilsen@frb.gov; or by phone at 202-452-2494.

TRADEMARK INFORMATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.