Paper 026-2007

Affordable SAS® Tips

Teo Gamishev, Office of Research and Statistics, Columbia, SC

ABSTRACT

SAS continue to impress with a large number of powerful tools, providing the user with almost unlimited possibilities to find the best solution for any programming situation. While enjoying the life of SAS programmer, you may find yourself surrounded by various problems, demanding urgent decision. This article contains some of the answers you may need and use. The following tips do not require any special training or knowledge in order to be applied into your coding arsenal. They are really affordable.

PROBLEMS

Here is a list of the problems covered on the following pages:

- How to control the flow of the SAS program;
- How to create custom time periods, like fiscal or state years;
- How to automatically create new data set in the beginning of the new time period;
- How to generate custom date FORMATs and INFORMATs;
- How to read large number of datasets located in the same folder;
- How to clean dataset variables from bugs;

SOLUTIONS

CONTROL THE SAS PROGRAM FLOW WITH MACROS turns to be the very effective and quick solution. You can perform a check for certain condition, for example; check the number of observations, and based on that to continue or stop execution of the program. Wrapping the program with macro is very simple and assures that the program will be executed or suspended on certain conditions.

EXAMPLE

```
1.  ***  Check for matching numbers of observations  ***;
2.  %let clienth=257866;
3.  %macro check;
4.         proc sql noprint;
5.         select count(*) into :nbr_rows from tmpclih;
6.         quit;
7.  %if &nbr_rows = &clienth %then %do;
8.         %put  CHEK OK, MY_TABLE HAS CORRECT NUMBER OF OBS -- &nbr_rows ;
9.         ***  UPDATE MASTER FILE  ***;
10.        PROC APPEND BASE=DSS.CLIHIST DATA=TMPCLIH;
11.        RUN;
12. %end;
13. %else %do;
14.        %put *  ERROR *  ERROR *  ERROR *  ERROR *  ERROR *  ERROR * ;
15. %end;
16. %mend check;
17. %check
```

In lines #4 to #6 the number of the observations in the data set is stored in a macro variable nbr_rows, using INTO statement from within the PROC SQL. INTO statement is similar to CALL SYMPUT in a data step. The number is compared to the control number on line #7 and a statement is written to the log, if the logic is TRUE. In case of a FALSE outcome the lines from #8 to #12 are simply skipped and no appending takes place. Line #14 generates an entry in the log, flagging for discrepancies between the numbers.

CREATING CUSTOM TIME PERIODS like fiscal year, state year, semi year or quarter is an easy task to perform when using INTNX function. The form of the function is:

```
INTNX( interval, from, n <, alignment > )
```

The general form of an interval name is:

```
name<multiplier><.shift-index>
```

Multiplier - specifies a multiple of the interval. It sets the interval equal to a multiple of the interval type. For example, YEAR2 consists of two-year, or biennial, periods. Shift-index - specifies the starting point of the interval. By default,

the starting point is 1. A value that is greater than 1 shifts the start to a later point within the interval. The unit for shifting depends on the interval. For example, YEAR.7 specifies yearly periods that are shifted to start on the first of July of each calendar year and to end in June of the following year.

SAS uses the arbitrary reference time of midnight on January 1, 1960, as the origin for non-shifted intervals. Shifted intervals are defined relative to January 1, 1960.

The following example shows how to create calendar, fiscal and state (beneficiary) yearly time periods.

EXAMPLE

```
1. data _null_;
2. cydmv=intnx('year',intnx('month',today(),-1),0);   * cal year data month;
3. bydmv=intnx('year.7',intnx('month',today(),-1),1); * bene year data month;
4. fydmv=intnx('year.10',intnx('month',today(),-1),1);* fisc year data month;
5. call symput('CYDM',put(cydmv,YEAR2.));             * cal year data month;
6. call symput('BYDM',put(bydmv,YEAR2.));             * bene year data month;
7. call symput('FYDM',put(fydmv,YEAR2.));             * fisc year data month;
8. run;
9. %put _user_;
```

The cydmv variable, containing calendar year, is included here for the purpose of being able to compare it with bydmv for state and fydmv for fiscal years. The last value of 1 on lines #3 and #4 forces the values of the state and fiscal years one year ahead of the current calendar year. It is very convenient to use the above macro variable for creating generic dataset names like: old&BYDM indicating data set, containing state year data, here. Those generic names can be easily manipulated in SAS codes when handling with monthly or yearly datasets and reports.

AUTOMATED DATASET CREATION IN THE BEGINNING OF THE NEW TIME PERIOD

It may occur in many occasions that you need to create a new dataset in the beginning of the new time period, for example a new calendar or fiscal year. One simple solution could be found here.
In the following example, a SAS macro *newyearset* is comparing the year of last data month &DMYY and the year of the previous data month &BDMYY on line #3. If they happened to be different, then a new dataset DSS.ccomp&DMYY will be created, containing the data from the last month, only. It is assured by the statement if benmonth=&DMYYYYMM on line #7**.**
The tip here is to initially refer to the year of the month before the last data month. By this way you will be able to get the old year dataset and create the dataset for the New Year, if necessary.

EXAMPLE

```
1.  * MACRO checking for a new year and creating a new year set  *;
2.  %macro newyearset;
3.          %if &DMYY NE &BDMYY
4.          %then %do;
5.                  data DSS.ccomp&DMYY;
6.                  set DSS.ccomp&BDMYY;
7.                  if benmonth=&DMYYYYMM;
8.                  run;
9.          %end;
10. %mend newyearset;
11. %newyearset;
```

Wrapping the code in a macro assures full control over the flow of the program in case of a TRUE and FALSE outcomes.

CREATING THE MONTH NUMBER WITHIN A QUARTER is a task you can solve by using INTCK time interval function. It counts the number of interval boundaries between two dates or between two datetime values. The form of the INTCK function is:

```
INTCK( interval, from, to )
```

In the following example X on line #3 - indicates from and Y on line #4 - indicates to variables, calculated separately for simplicity. If the REPORT month and the FIRST MONTH of the quarter are the same, the INTCK outcome will be zero. For that purpose 1 is added to the value of Z on line #5. The number Z is assigned to a macro variable on line #6 and used to create a generic data set name on line #9.

EXAMPLE

```
1.  * Creating a macro variable for # of the REPORT month within the quarter;
```

```
2.  data _null_;
3.  X=intnx('qtr',intnx('month',today(),-2),0); *from date-the first qtr day;
4.  Y=intnx('month',today(),-2); * to date-the first day of the REPORT month;
5.  Z=1+intck('month',x,y);      * measuring the month interval b/n x and y;
6.  call symput("qtrmth",trim(left(put((z),1.))));* creating macro variable;
7.  run;
8.  %put &qtrmth;
9.  data qmth&qtrmth;
10.       set regcty;
11. run;
```

CUSTOM DATE FORMATS AND INFORMATS are very often desperately needed. It turns out that by using them simultaneously, you can solve any unusual date format issue. The first step is to create the desired FORMAT with the powerful DIRECTIVES in PICTURE statements. Then you can use the newly created FORMAT to generate the relevant INFORMAT.

EXAMPLE

```
1.  * creating user defined FORMAT using PICTURE statement and DIRECTIVES;
2.  proc format ;
3.        picture myfmt low-high = '%Y/%b/%d' (datatype = date) ;
4.  run ;
5.  * creating control data set for user defined INFORMAT;
6.  data infmt ;
7.  retain fmtname "yearmd" type "I" ;
8.  do label = "01jan1999"d to "01jan2003"d ;
9.        start = put(label,myfmt11.) ;
10.       start = trim (left (start) ) ;
11.       output ;
12. end ;
13. run ;
14. * output the custom INFORMAT;
15. proc format cntlin = infmt ;
16. run ;
17. * use the custom INFORMAT;
18. data _null_ ;
19.       _txtdate = "2002/APR/17" ;
20.       _sasdate = input (_txtdate,yearmd.) ;
21. put _sasdate = ;
22. run ;
23. LOG: 15447
```

Here is a list of the most common DIRECTIVES in use:

%A	Full weekday name – Monday, Friday
%a	Abbreviated weekday name – Mon, Fri
%B	Full month name – April, October
%b	Abbreviated month name – Apr, Oct
%Y	Year with century – 2005, 2020
%y	Year without century – 5, 20
%0y	Year without century – 05, 20
%m	Month as a decimal – 1, 11
%0m	Month as a decimal – 01, 11
%U	The week number of the year – 1, 52
%0U	The week number of the year – 01, 52
%j	The day of the year – 1, 75, 352
%0j	The day of the year – 001, 075, 352
%d	Day of the month as a decimal number - 1
%0d	Day of the month as a decimal number - 01
%w	Weekday as a decimal number – 1, 5, 7
%p	The day half – AM, PM
%H	The hour – 24 hrs clock – 4, 17, 23
%0H	The hour – 24 hrs clock – 04, 17, 23
%I	The hour – 12 hrs clock – 4, 5, 11
%0I	The hour – 12 hrs clock – 04, 05, 11
%M	The minute – 1, 19, 25
%0M	The minute – 01, 19, 25

%S	The second – 5, 15, 35
%0S	The second – 05, 15, 35

In combination with the possibility of including text, as in the code below, these DIRECTIVES can produce almost any kind of date value you need.

EXAMPLE

```
1. proc format ;
2.        picture long low-high = '%dth %B is a %A ' (datatype = date) ;
3. data _null_ ;
4.        myday = "17apr2006"d ;
5.        put myday : long40. ;
6. run ;
7. LOG: 17th April is a Monday
```

CONCATENATING MULTIPLE DATA SETS from a pool of datasets is a regular task for a SAS programmer. One simple approach to read all datasets in the pool is to store the names as a character string in a macro variable.
Whenever a new libname is assigned, SAS immediately collects all the data from the libname address. You can access that information via sashelp vtable, as shown on line #5. In the example below, the datasets names are read from the sashelp vtable and stored in a macro variable mysets, as seen on line #4, separated by blank spaces. The where statement on line #6 assures that only sets from the libname OLD will be selected. Some additional restrictions are applied on line #7. Finally, on line #9, all datasets are concatenated in OLDSETS.

EXAMPLE

```
1. LIBNAME OLD 'e:\dss\fipacket\applications';
2. proc sql noprint;
3.    select 'OLD.'||memname
4.        into: mysets separated by ' '
5.        from sashelp.vtable
6.    where libname='OLD'
7.        and memname NE substr(memname,6,3)='FLS';
8. quit;
9. DATA OLDSETS; SET &mysets; RUN;
```

DEBUGGING A VARIABLE IN A DATA SET is something you may need to do due to a customer or project requirements. In some occasions, certain observations within a particular variable may contain a bug (unreadable sign) you want to get rid of. The following code is offering a simple answer to that task:

EXAMPLE

```
1. data mydata (rename=(cleaned_variable=my_variable));
2.    set mydata;
3.        cleaned_variable = my_variable;
4.        do  i = 1 to length(my_variable);
5.              substr(cleaned_variable,i,1) =
6.              compress(substr(my_variable,i,1),
7.              compress(substr(my_variable,i,1), collate(32, 126)));
8.        end;
9.    drop i my_variable;
10.run;
```

No new dataset is generated here. The cleaned_variable is created for the purpose of debugging and is immediately renamed to the original variable my_variable, as you can see from lines #1 and #9 in the example. In line #7 the program is cleaning all ASCII codes with a number outside the 32-126 scope. The range of 32-126 contains all numbers and characters, only. In general, all you need to do is to replace mydata and my_variable with the name of your data set and variable to be cleaned, submit the program and the job will be done.

CONCLUSION

This paper offers solutions for common problems new SAS user may encounter during the daily programming routine. They are simple to implement and really affordable. Hopefully the above tips will reduce the amount of tedious work and leave more space to enjoy creative coding.

REFFERENCES

SAS Institute Inc. (2006). "*What's New in SAS® 9.0, 9.1, 9.1.2, and 9.1.3*." Cary, NC: SAS Institute Inc.

Venky Chakravarthy. 2002. "Have a Strange DATE? Create your own INFORMAT to Deal with Her." *Proceedings of the Twenty Seventh Annual SAS Users Group International Conference*, Orlando, FL, 101-27.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Teo Gamishev, M.S. in Statistics,
gamishev@hotmail.com
38 Crossbow Lakes Ct
Columbia, SC 29212