Paper BB-72

Tales from the Help Desk 6: 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. Convert a variable from character to numeric or vice versa and keep the same name.
2. Convert multiple variables from character to numeric or vice versa and keep the same names.
3. Convert character or numeric values to SAS date values.
4. Use a data set when the custom format assigned to a variable cannot be found.
5. Use an array definition in multiple DATA steps.
6. Use values of a variable in a data set throughout a DATA step by copying the values into a temporary array.
7. Use values of multiple variables in a data set throughout a DATA step by copying the values into a 2-dimensional temporary array.

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 sixth of its type; see the references for five previous papers.

1. Convert a variable from character to numeric or vice versa and keep the same name

This issue arises frequently because users want to merge two data sets by a variable that is numeric in one data set and character in the other. To merge, the variable must have the same type and name in both data sets.

Convert between numeric and character without keeping the same name

You can easily convert numeric variables to character variables with the PUT function, and convert character variables to numeric variables with the INPUT function, as in the following code.

```
data one;
    /* create numeric variable */
  stcode = 33 ;
    /* convert numeric variable to character variable */
  state = put(stcode, 2.) ;
    /* convert character variable to numeric variable */
  stnumv = input(state,2.);
run;
```

Convert a single variable and keep the same name

It's less obvious how to convert a variable from numeric to character or vice versa in a DATA step and keep the same name. Suppose we want to convert character variable STATE in data set ONE to a numeric variable. One way to do it is as follows.

```
data two ;
  set one (rename = (state = Z_state)) ;
  drop Z_state ;
  state = input (Z_state, 12.);
run;
```

Let's review how this code is processed.

- The DATA statement specifies that DATA step results are written to data set TWO.
- The SET statement controls how data set ONE is read. The RENAME option has the syntax *oldname* = *newname*. It does not change the name of STATE in data set ONE, but causes the name to be Z_STATE in this DATA step.
- The DROP statement prevents the variable Z_STATE, which is available in the DATA step, from being written to the output data set, TWO. This prevents an extra variable from cluttering data set TWO.
- Since data set TWO does not have an incoming character variable called STATE (it was renamed to Z_STATE in the SET statement), we can create a new numeric variable called STATE in data set TWO with the INPUT function. The 12. informat allows us to convert values with a width up to 12 characters.

An equivalent way to code this step is to remove the DROP statement and add DROP=Z_STATE to the DATA statement, as follows.

```
data two (drop=Z_state);
```

The following code does not work because character variable STATE from data set ONE exists during the DATA step and is renamed to Z_STATE when output to data set TWO, so the INPUT function fails because Z_STATE does not exist.

```
data two;   /* Incorrect code #1 */
  set one;
  rename state = Z_state;
  state = input (Z_state, 12.);
run;
```

The following code does not work because STATE is an existing character variable and the statement with the INPUT function tries to create a numeric variable called STATE.

```
data two;   /* Incorrect code #2 */
  set one;
  state = input (state, 12.);
run;
```

2. Convert multiple variables from character to numeric or vice versa and keep the same names

Suppose data set ONE contains character variables X1 (length 1), X4 (length 4), and X8 (length 8), all containing digits that we want to convert to numeric variables with the same name. In real life, the data set would also have additional variables and observations that are omitted here for simplicity.

```
Obs     x1     x4      x8

 1      1      22      333
 2      4      5555    66666666
```

The code presented here is analogous to the code in the previous section, with macro loops used to process multiple variable names. It does the following.

- In the previous section, a SET statement with the RENAME option renamed STATE to Z_STATE for use in this DATA step. Here, a SET statement with the RENAME option similarly renames the existing character variables by prefixing them with Z_ (X1 to Z_X1, X4 to Z_X4, and X8 to Z_X8).

- In the previous section, a DROP statement prevented temporary character variable Z_STATE from being written to the output data set. Here, we generate a DROP statement to prevent the temporary character variables (Z_X1, Z_X4, Z_X8), which are available during the DATA step, from being written to the output data set.

- In the previous section, an assignment statement containing the INPUT function converted the temporary variable Z_STATE to numeric variable STATE. Here, we generate one assignment statement per variable. Each statement uses the INPUT function to convert the value in a temporary character variable to a numeric value in a variable with the original name (X1, X4, or X8).

```
%macro changetype;
  %local j varcurrent dropvars numvars renamevars chartonum;

    /* Create a macro variable with the names to convert to numeric */
  %let chartonum= x1 x4 x8;

    /* Loop through the list of variables and build two macro variables:
       1. RENAMEVARS with a paired list of variables to rename.  In this example
       we generate the following:  x1=Z_x1 x4=Z_x4 x8=Z_x8
       2. DROPVARS with a list of temporary variables to drop. In this example
       we generate the following:  Z_x1 Z_x4 Z_x8
    */
  %let varcurrent = %scan(&chartonum, 1, %str( )); /* parse 1st variable */
  %let j = 1; /* parse 2nd, 3rd, ... variable in %DO loop */
  %do %while (&varcurrent ne ) ; /* stop when %scan returns null */

      /* Add current variable in form varname=Z_varname to list of paired
         variables to rename */
    %let renamevars = &renamevars &varcurrent=Z_&varcurrent;

      /* Add current variable to list of variables with Z_ prefix to drop */
    %let dropvars = &dropvars Z_&varcurrent;

    %let j=%eval(&j+1); /* set counter to parse next variable */
    %let varcurrent = %scan(&chartonum, &j, %str( )); /* parse next variable */
  %end;
  %let numvars = %eval(&j-1); /* Number of variables processed, 3 in this example*/

  data two;

      /* The SET statement includes the RENAME option values created above */
    set one (rename =(&renamevars));

      /* Drop the temporary Z_ variables so they are not written to the
         output data set */
    drop &dropvars;

       /* Generate one assignment statement per variable that uses the INPUT
          function to convert a character variable with a temporary name
          starting with Z_ to a numeric variable with the original name.
       */

    %do j=1 %to &numvars;
       %let varcurrent= %scan(&chartonum, &j, %str( )); /* parse current variable*/
       &varcurrent=input(Z_&varcurrent,8.);          /* convert current variable */
    %end;
```

```
  run;
%mend changetype;
%changetype
```

3. Convert character or numeric values to SAS date values

Character values can be directly converted to SAS dates, but some users find it confusing that numeric values cannot be directly converted.

Suppose data set ONE has one observation and two variables: CHARVAR1 is character and NUMVAR1 is numeric. We want to convert CHARVAR1 and NUMVAR1 to SAS dates.

```
CHARVAR1     NUMVAR1
20141229     20141229
```

In general, we can convert numeric values to character values with the PUT function, and convert character values to numeric values with the INPUT function. SAS dates are numeric values.

The INPUT function converts a character value to a SAS date value as follows. The first argument is the character value to convert, and the second argument is an informat that corresponds to the appearance of the data.

```
sasdate1 = input(charvar1, yymmdd8.) ;
```

But, converting a numeric value is not as straightforward, for the following reasons.

- The first argument to the INPUT function must be a character value, so the following statement is not valid.

  ```
  sasdate1 = input(numvar1, yymmdd8.) ;
  ```

- The result of the PUT function is always a character value, so the PUT function cannot create a SAS date, which is numeric.

To create a SAS date from a numeric value, first convert the numeric value to a character value with the PUT function. Then, convert the character value to a SAS date with the INPUT function.

```
tempchar = put (numvar1, 8.) ;
sasdate2 = input(tempchar, yymmdd8.) ;
```

This statement is equivalent to the previous two statements. It avoids creating the variable TEMPCHAR, but is somewhat less readable.

```
sasdate2 = input(put(numvar1,8.),yymmdd8.) ;
```

4. Use a data set when the custom format assigned to a variable cannot be found

When a format of a variable in a SAS data set cannot be found, an error can occur. This problem can be introduced when a data set is copied from one platform to another or received from an external source without the associated format library that contains the format for one or more variables in the data set.

For example, we want to print data set ONE in the folder c:\myfiles that was received from an external source without the format DAYWORD that maps 1 to Sunday, 2 to Monday, and so on.

```
          ONE
  dayofweek income

      1       10
      2       20
      3       30
```

We reference and print the data set as follows.

```
libname xxx 'c:\myfiles';
proc print data=xxx.one;
run;
```

PROC PRINT generates an error similar to the following.

```
ERROR: Format DAYWORD not found or couldn't be loaded for
       variable dayofweek.
NOTE: The SAS System stopped processing this step because
      of errors
```

Three ways to fix this problem are described here; all allow PROC PRINT to successfully print the data set.

Method 1. Disassociate the format in a DATA step.

```
data xxx.one;
  set xxx.two;
  format dayofweek;
run;
```

Method 2. Disassociate the format with PROC DATASETS.

Method 1 reads the data set observation by observation. This method reads only the data set header, so it is much faster, but it only works if the data set is in a "native format." That is, if the data set is from another platform, it will not work, and you should use Method 1.

```
proc datasets library=xxx;
  modify two;
  format dayofweek;
quit;
```

Method 3. Use the system option NOFMTERR.

NOFMTERR tells SAS not to generate an error when a format is not found. The default value of this option is FMTERR. This method is useful if you cannot or prefer not to copy or change the data set.

```
OPTIONS nofmterr;
run;
```

5. Use an array definition in multiple DATA steps

An array definition is only in effect in the DATA step in which it is defined and cannot be stored in a data set.

If you use the same array in multiple steps, as users often do, it can be helpful to define it in only one place. Then, if the array definition changes, you only need to change the code once.

To define the same array in more than one DATA step, you can do one of the following.

- Create a macro variable containing the array definition with a %LET statement, and use the macro variable in each DATA step.

- Create a macro containing the array definition, and execute the macro in each DATA step.

Here is an example with a macro variable.

```
    /* Create a macro variable called ARRAYDEF.  It contains an
       array definition that is used in subsequent steps. */
%let arraydef= array gnp (*) consume invest gov tax;

    /* Use the array definition in a DATA step */
data two;
  set one;
  &arraydef; /* define array GNP */

   /* more SAS statements */

run;
     /* Use the array definition in another DATA step */
data three;
  set one;
  &arraydef; /* define array GNP */

   /* more SAS statements */

run;
```

Here is an example with a macro.

```
    /* Create a macro called ARRAYDEF.  It contains an
       array definition that is used in subsequent steps. */
%macro arraydef;
  array gnp (*) consume invest gov tax;
%mend arraydef;

    /* Use the array definition in a DATA step */
data two;
  set one;
  %arraydef /* define array GNP */

   /* more SAS statements */

run;
     /* Use the array definition in another DATA step */
data three;
  set one;
  %arraydef /* define array GNP */

   /* more SAS statements */

run;
```

6. Use values of a variable in a data set throughout a DATA step by copying the values into a temporary array

Suppose we want all values of variable BANKID from data set ONE available while we read and process data set TWO and create data set THREE but do not need to write the values of BANKID to THREE.

Users often merge data sets in this situation, but a simple alternative is to copy the values of BANKID to a temporary array at the beginning of the DATA step. Values in a temporary array are automatically retained, and are thus available throughout the DATA step.

Here are data sets ONE and TWO.

```
            ONE                              TWO

obs bankid   var1   var2          var3   var4   var5

 1    111     10     11             1      2      3
 2    222     20     12             4      5      6
 3    333     30     13             7      8      9
 4    444     40     14
```

Method 1

First, use PROC SQL to create two macro variables.

- ALL_BANK_IDS contains the values of variable BANKID, space-separated.
- NUM_OBS contains the number of values of BANKID in macro variable ALL_BANK_IDS. 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:

```
ALL_BANK_IDS: 111 222 333 444
NUM_OBS:      4
```

Then, in a DATA step, define temporary array ALL_BANKIDS using the two macro variables created with PROC SQL to specify the array size and values, and read data set TWO. The temporary array values will be available throughout the DATA step.

```
  /* Create macro variables with the values of BANKID
     and the number of values of BANKID */
proc sql noprint;
  select bankid into :all_bank_ids separated by ' '
  from one;
  %let num_obs = &sqlobs;
quit;

data three;
     /* Create temporary array with values of BANKID */
   array all_bankids (&num_obs) _temporary_ (&all_bank_ids);

    /* Read data set TWO.  The values of BANKID in the ALL_BANKIDS
       temporary array are available throughout the DATA step. */
  set two;

     /* Standard DATA step processing of TWO goes here. */
```

```
run;
```

Method 2

This method is not as simple as Method 1, but could be helpful to review because it is analogous to the method used to store multiple variables in a 2-dimensional array in the next section.

First, we use PROC SQL to create macro variable NUM_OBS containing the number of observations in data set ONE, 4. Then, in a DATA step, we read the values of BANKID into the temporary array ALL_BANKIDS at the start, then process data set TWO.

```
  /* Set macro variable NUM_OBS to the number of observations in ONE.
     It is used as the size of the temporary array */
proc sql noprint;
   select count(*)
   into :num_obs
   from one;
quit;

data three;
  array all_bankids (&num_obs) _temporary_;
  drop bankid;

    /* Read the values of BANKID into the temporary array */
  if _n_=1 then do;
    do _i = 1 to &num_obs;
      set one (keep=bankid) ;
      all_bankids(_i) = bankid;
    end;
  end;

    /* Read data set TWO one observation at a time in the standard way.
       The values of BANKID in the ALL_BANKIDS temporary array are
       available throughout the DATA step. */
  set two;

     /* Standard DATA step processing of TWO goes here. */

run;
```

7. Use values of multiple variables in a data set throughout a DATA step by copying the values into a 2-dimensional temporary array

In the previous section, we copied all values of a single variable to a temporary array for use throughout a DATA step. More generally, we could copy all values of multiple variables to a 2-dimensional temporary array.

In this example, we copy all numeric variables in data set ONE that begin with the letter "X" (X1 and X2) into a temporary array.

Here are data sets ONE and TWO.

```
           ONE                              TWO

obs   x1      x2    var1          var3   var4   var5

 1    111     10     11             1      2      3
 2    222     20     12             4      5      6
 3    333     30     13             7      8      9
 4    444     40     14
```

First, use PROC SQL to create three macro variables.

- NUM_OBS contains the number of observations in ONE.
- VARS_XSTART contains a space-separated list of variables in ONE that begin with the letter "X".
- NUM_VAR contains the number of variables in ONE that begin with the letter "X".

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

```
NUM_OBS:      4
VARS_XSTART:  x1 x2
NUM_VAR:      2
```

```
proc sql noprint;
    /* Copy number of observations to macro variable NUM_OBS*/
  select nobs
  into :num_obs
  from dictionary.tables
  where libname="WORK"
  and memname="ONE";

    /* Copy variables starting with X to space-separated macro variable
       VARS_XSTART and number of variables to macro variable NUM_VAR */
  select name
  into :vars_xstart separated by ' '
  from dictionary.columns
  where libname="WORK"
  and memname="ONE"
  and type = "num"
  and upcase(substr(name,1,1)) = "X";
  %let num_var = &sqlobs;
quit;
```

Then, read the values of the variables beginning with X (X1 and X2) into the 2-dimensional temporary array at the start of the DATA step. The values are available throughout the DATA step as we process data set TWO.

```
data three;
  array all_xvars (&num_obs, &num_var) _temporary_;
  array xvars (&num_var) &vars_xstart;
  drop x: _i _j;

  if _n_=1 then do;
      /* Read the values into the temporary array */
    do _i = 1 to &num_obs;
      set one (keep=x:);
```

```
        do _j = 1 to &num_var;
          all_xvars(_i,_j) = xvars(_j);
        end;
      end;
    end;  /* of if _n_=1 then do; */

      /* Now read data set TWO one observation at a time in the standard way.
         The values of X1 and X2 are available in all observations. */
    set two;

       /* DATA step processing of TWO goes here */

  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>

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. Heidi's contributions included Method 1 in Section 6 and improvements to the code in several sections. 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