Paper 107-28

%Fun &With %SYSFUNC

Derek Morgan, Washington University Medical School, St. Louis, MO

Abstract

The %SYSFUNC macro has allowed access to the SAS component language inside of traditional DATA step programming. We have used this to great advantage during version 6 to version 8 migration and in our work with our external collaborators. We've used %SYSFUNC to simplify renaming variables, changing case of variable names, and to simulate the IMPORT wizard as a part of a larger table creation task. The first two will be covered in this paper.

Macro 1: %varnames: Modify Data, Alter Variable Names

This macro was developed because of incompatible data structures. One of our collaborators sent us a SAS® System table that was to be pooled with ours. However, they used an entirely different data structure. Our data structure used one character variable (with the marker name as the variable name) to represent both alleles of a genetic marker, they used two numeric variables to represent each allele, and the variable name was the marker name, with either a "1" or "2" appended. The task therefore, had two components. Collapse their paired numeric variables into a single character variable, then remove the suffix to name the new variable. Given that the variable names are easy to mistype, an automated process was preferred. The code is as follows:

```
/* remove extra variables from orig dataset, and
create work dataset */
1   DATA marsh;
2   SET collab (DROP=_OBSTAT_);
3   run;

4   %MACRO varnames;
/* Open dataset to get variable names */
5   %LET ds = %SYSFUNC(OPEN(marsh,i));

/* marker data starts with 9th variable in dataset */
6   %DO i=9 %TO %SYSFUNC(ATTRN(&ds,NVARS)); ①

/* put orig var name in indexed macro var */
7       %LET dsvn&i = %SYSFUNC(VARNAME(&ds,&i));

/* remove last two chars of orig var name to get
pooled var name */
8       %LET l = %LENGTH(&&dsvn&i);
9       %LET vn&i = %SUBSTR(&&dsvn&i,1,
                            %EVAL(&l-2));
10  %END;
```

```
/* create data set in FBPP format */
11  DATA final.markers;
12  SET marsh;

/* original data are paired so index by 2 */
13  %DO i=9 %TO %SYSFUNC(ATTRN(&ds,NVARS)) %BY
    2;   ②

/* Define length of FBPP char variable */
14      LENGTH &&vn&i $ 16;

/* odd number index is where allele1 for a given
marker is stored, add 1 to get allele2 */
15      %LET k = %EVAL(&i+1);

/* Create FBPP char var - statement translates to
FBPP variable-name = trim(left(put(GENOA allele1
varname,3.))) || '/' || left(put(GENOA allele 2
varname,3.));
Execute once for every 2 variables obtained from
dataset.  */
16      &&vn&i = TRIM(LEFT(PUT(&&dsvn&i,3.)))
                || '/' || LEFT(PUT(&&dsvn&k,3.));
17  %END;

/* Remove all original allele variables */
18  DROP _numeric_;
19  RUN;

/* close orig dataset */
20  %LET rc = %SYSFUNC(CLOSE(&ds));
21  %MEND varnames;

22  %varnames;
```

That's A Lot Of && and %: How Does it Work?

One thing about %SYSFUNC is that it is a macro function, so it works with macro variables and macro functions. Macro functions give you a lot of "%" in your code. Moreover, you can't use arrays to address macro variables in a loop, so you need to use indexed macro variables, (e.g., &v1, &v2, &v3...). Of course, this means that you will be using the "&&v&i" method of referencing them, which is three times the "&" for your money. Now let's look at the code in detail.

The first section (lines 1-3) creates a temporary data set to work with. I use it to make sure nothing happens to the original, and to remove any variables that are not needed in the final dataset. Line 4 starts the macro because I can't use macro calls in open code. Line 5 gets things started by getting a dataset identifier (&ds) for the SCL functions to be used.

The %DO loop that starts in line 6 loops through the variables in the dataset. This is how the variable names that we want in the final dataset are assigned. Since they are based on the original variable names, this enables us to avoid typing them. The ① is used in line 6 to draw your attention to the ATTRN() function. When called in SCL, the second parameter of the function (NVARS here) is normally a character variable, and would have quotes around it. Here, since it is passed through a macro, the symbol is automatically character, and quotes will cause this to fail. The DO loop starts at the ninth variable because the first eight variables in the dataset are identifiers, and do not need to be translated.

Line 7 reads each variable name using the SCL function VARNAME() and assigns it to an indexed macro variable. The following lines take that macro variable, remove the last two characters, and store it in a new macro variable with the same index. For example, if the twenty-third variable in the data set has the name "GBCB344A3_1", then &dsvn23 = "GBCB344A3_1". The new macro variable we create with the %SUBSTR macro will be named &vn23, and it will be equal to "GBCB344A3", having lost the last two characters on &dsvn23. This ends the first part of the process. We now have an old_variable_name / new_variable_name correspondence stored in indexed macro variables.

The second part of the process (lines 11-19) is the DATA step that creates our data set with the correct structure. Again, we start looping through the variables in the dataset starting with the ninth variable because that's where the translating starts. However, we are counting by 2 in this loop (②), since we only want one character variable created for each pair of numeric variables. The first part of our new variable is in variables ending in "_1", and the second part is in those that end in "_2". Our pairing is in line 15. If the index &i represents the number of the first variable ("*varname*_1") in a pair, then %EVAL(&i+1) represents the number of the second variable in the pair, and it is assigned to the macro variable &K.

The character variable is created in line 16 by concatenating the character equivalents of the two original numeric variables. The indexed macro variables contain the variable names themselves. Using the previous example of variable number 23, we are taking the macro variables &dsvn23 ("GBCB344A3_1") and &dsvn24 ("GBCB344A3_2), and creating &vn23 ("GBCB344A3"). We use the PUT function to convert the numeric values in the variables GBCB344A3_1 and GBCB344A3_2 into character values. We concatenate them to create our final character value. That value is then stored in the character variable GBCB344A3. So, if GBCB344A3_1=126, and GBCB344A3_2=128, the value in the variable GBCB344A3 will be equal to "126/128". Once all the variables in the original dataset have been processed, we can end the loop, and remove the original numeric variables from our final dataset (line 18). Before the macro ends, the original working dataset is closed using the SCL function CLOSE(), to release the internal file handle. It's a highly recommended bit of housekeeping.

%varnames Summary

This could have been done without %SYSFUNC, using PROC CONTENTS with a dataset, reading that in, and creating the indexed macro variables with a DATA step and CALL SYMPUT(). We did not benchmark that alternative, but it seems that using %SYSFUNC for one pass through the original dataset would be more efficient than one pass with a PROCEDURE, and a second one with a DATA step that uses two calls to the SYMPUT() function.

Macro 2: %chgvcase: Set all variable names in a dataset to upper or lower case.

This macro was developed because a collaborator sent data to us with the correct variable names, but all in mixed case. It was quite a surprise when we did a merge to find that we had 750 variables instead of the 450 we expected. Here's the code:

```
/* CHGVCASE macro
Changes the case of all variables in a given dataset
There are 2 parameters:
  dsn=Name of the dataset to be altered (any alter
password is also part of parameter
  case=U or L for Upper or Lower case var names
(default is upper)    */

1   %MACRO chgvcase(dsn=,case=U);
2   %GLOBAL ds;
3   %LET ds = %SYSFUNC(OPEN(&dsn,i));
4   %DO i=1 %TO %SYSFUNC(ATTRN(&ds,NVARS));
5      %LET dsvn&i = %SYSFUNC(VARNAME(&ds,&i));
6      %IF &case = L %THEN
7          %LET vn&i = %QLOWCASE(&&dsvn&i);
8      %ELSE
9          %LET vn&i = %QUPCASE(&&dsvn&i);
10  %END;
11  %LET rc = %SYSFUNC(CLOSE(&ds));

12  DATA &dsn;
13  SET &dsn;
14  %LET ds = %SYSFUNC(OPEN(&dsn,i));
15  %DO i=1 %to %SYSFUNC(ATTRN(&ds,NVARS));
16      RENAME &&dsvn&i = &&vn&i;
17  %END;
18  %LET rc = %SYSFUNC(CLOSE(&ds));
19  RUN;

20  %MEND chgvcase;
```

This application uses indexed macro variables as the first one did (recycled code is good,) but instead of chopping off the last two characters, the results are translated into upper or lower case using the appropriate macro function. Note that we use %QUPCASE() and %QLOWCASE() (①). If we were to use just %UPCASE and %LOWCASE, the macro would fail when it encountered variable names with quotable characters. Although most people would not use an operator for a variable name, it can happen. This keeps the macro from blowing up if it does. Another caution is that the QLOWCASE macro is part of one of the autocall libraries distributed in the SAS System, therefore,

it may or may not be available at a particular installation. The "SAS® Guide to Macro Processing, Version 6 Second Edition" contains a %LOWCASE macro. With additional coding to mask the reserved words, you could create a %QLOWCASE-like macro of your own.

Once the indexed macro variables have been created (lines 4-10), they are used in a RENAME statement in a subsequent DATA step (lines 12-19) as a macro loop works through all the variables. This is basic code that ultimately writes a multitude of RENAME statements in a DATA step. The advantage is that the programmer doesn't have to do it manually. With a little creativity, this could be adapted to perform as a renaming tool for larger datasets.

Summary

The %SYSFUNC macro function allows access to the SCL functions, many of which have no DATA step equivalent, especially those that access table-level properties of datasets. We have taken advantage of this to modify variable names within tables, or to create derived variables from variable names, all in an automated fashion. There are many other possible uses for this powerful macro function. Maybe you can find one to help you solve some of your problems.

Further inquiries are welcome to:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wubios.wustl.edu

This and other SAS System examples and papers can be found on the World Wide Web at:

http://www.biostat.wustl.edu/~derek/sasindex.html

References:

SAS Institute Inc., *SAS® Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc, 1990

Trademarks