Paper 050-2007

Changing Data Set Variables into Macro Variables

William C. Murphy
Howard M. Proskin and Associates, Inc., Rochester, NY

ABSTRACT

For reusing code, macro variables are an indispensable part of the SAS® system. You can create multi-use programs in which titling, subsetting, and even analysis variables can be controlled by changing the values of macro variables. To create a few macro variables, the %LET statement works extremely well. But if you want to automate the production of macro variables, DATA steps with the CALL SYMPUT statement or PROC SQL provide a better method. If a large number of macro variables need to be created from data set variables, however, even these solutions many require a great deal of typing. Furthermore, if the number of macro variables created for each data set variable and observation is extremely large, the memory demands on the system can be huge. The SAS system however, provides an elegant way to overcome these problems with the CALL SET routine. The macro version of this routine, %SYSCALL SET provides a simple way to convert data set variables into the equivalent macro variables with few lines of code. Furthermore, the routine allows you to convert and process one observation of data set variables at a time, thus saving on system memory usage. The %SYSCALL SET routine is the ideal solution for changing data set variables into macro variables.

INTRODUCTION

Being a programmer in a small statistical consulting company, I am always trying to find ways to reuse code. If the program worked once, why write it again, especially if you are using it on a near identical project. Macro variables are one of the more useful tools of the SAS system that can expedite the reuse of code. You can readily take a program that was used for one project and generalize it with macro variables to be used with other projects. For example, a given study designation, 'Pharmaceutical Study 203', that was used in various places in a program could be easily changed to 'Pharmaceutical Study 204' if this information was stored in a macro variable. You could also use the same program for running off a series of tables on different study populations (Safety, Intent-To-Treat, Per Protocol, *etc*) if the population information was contained in a macro variable. You can use macro variables to store modeling information for statistical statements, titling and footnote information, analysis variable information, and data sources, all to keep a program useful in a variety of situation. All this could be done even without using any macro programming. However, the additional use of a macro %DO-loop combined with a series of macro variables allows programs to do repetitive processing with ease.

The first step is of course to create the macro variables and populate them with the information that we need. The SAS system provides many methods for making these macro variables. You can make them manually using the %LET statement, or you can make the process automatic using the CALL SYMPUT in the DATA step or the INTO statement in PROC SQL. To create a large number of different macro variables with different content, SAS software also offers us the power of the CALL SET statement.

%LET STATEMENT

Most everyone who programs the SAS DATA step is familiar with the %LET statement:

```
%let Name=William;
```

where the macro variable &Name is created by the %LET statement with the value 'William'. You can then use this macro variable whenever you need to access its content in your program. Often used numbers and phrases can be stored in similar macro variables at the start of a program. However, if you have more than a few macro variables to create, typing the %LET statements can be tedious and error prone.

CALL SYMPUT

If the content that you need to store in macro variables is contained in a SAS DATA step, the CALL SYMPUT statement can be used to create these macro variables:

```
data _null_;
```

```
        set sashelp.class;
        call symput('Name', Name);
        run;
```

where the CALL SYMPUT statement creates the macro variable &Name containing the content of the DATA step variable Name. However, the way the DATA step is written only the last value of the variable Name will be stored in the macro variable. To place all of the values of Name into macro variables, we can employ a suffix:

```
data _null_;
        set sashelp.class;
        suffix=put(_n_,5.);
        call symput(cats('Name',suffix), Name);
        run;
```

where we use the concatenation function CATS to appended onto the Name a suffix created from the automatic variable that counts observations (_n_). The result is that the first value stored in the variable Name is now stored in the macro variable &Name1; the second value is stored in the macro variable &Name2…etc. But there are even more elegant ways to accomplish this task.

PROC SQL

You can also automate the creation of macro variable by using PROC SQL:

```
proc sql noprint;
        select Name
                into :Name1-:Name19
        from SAShelp.class;
        quit;
```

where you use the SELECT statement to choose the variable Name and the INTO clause to store the content of Name into macro variable designated by the prefix ':'. Unlike the DATA step method, you must know there are 19 suffixes for the macro Name, or you could first calculate a macro variable with that information:

```
proc sql noprint;
        select count(*)
                into :NObs
        from SAShelp.class;

        select Name
                into :Name1-:Name%left(&NObs)
        from SAShelp.class;
        quit;
```

where the COUNT function is used to determine the number of observation, which are then stored into the macro &NObs.

THE MACRO PROGRAM

Macro variables can be used anywhere in open code. For many program, this is all you need to make your code reusable. However, if a great deal or repetitive processing is done, the macro program comes into use. For example, if we wanted to do a separate analysis and report on each person named in sashelp.class, we might write

```
%macro doit;
        %do i=1 %to &NObs;
                %put # # # Processing &&Name&i # # #;

                %* ---- Analysis Code Goes Here ----- *;

                %end;
        %mend;

%doit;
```

where the macro processor will resolve &&Name&i into &Name1, &Name2,… These variables along with &NObs will be further resolved to the values assigned in the last PROC SQL code. Somewhere in the 'Analysis Code', the programmer would have referred to these variables, perhaps in WHERE or IF statement, so the analysis is done only on the desired subject.

So it looks like we can do just about anything by creating macro variables with a PROC SQL and maybe using a macro %DO-loop. But what happens if there a great number of macro variables to create and an even larger number of suffixes for these variables. Is this really the best technique?

MANY MACRO VARIABLES

You are given a data set like sashelp.class and told to convert all of the variables in the DATA step into macro variables. All of the information contained in the macro variables will be used in a macro %DO-loop in processing each observation in the data set. So you could run a PROC CONTENTS to see what variables you have in the file. Then list these variables and their corresponding macro variables on the SELECT and INTO statements in PROC SQL. Alternatively, you could write a CALL SYMPUT for each variable in a DATA step. But if your data set has dozens or even hundreds of variables to be converted, this process is not only tedious but prone to error. However, most of these problems can easily be overcome with some DATA step programming:

```
data _null_;
       set sashelp.class;
       suffix=put(_n_,5.);

       array xxx{*} _numeric_;
       do i =1 to dim(xxx);
              call symput(cats(vname(xxx[i]),suffix),xxx[i]);
              end;

       array yyy{*} $ _character_;
       do i =1 to dim(yyy);
              call symput(cats(vname(yyy[i]),suffix),yyy[i]);
              end;
       run;
```

As in the previous example, we are appending the suffix created from the automatic variable _n_ onto the variable name supplied by the function VNAME. The SAS system requires separate ARRAY statements for characters and numbers. By using the automatic variables _numeric_ and _character_, we can assign all of the variables in the data set to these arrays. Then we use the '*' in the array definition and the DIM function in the loop to count the number of variables.

This DATA step will convert all data set variables to macro variables with the same name and an observation suffix, regardless of how many variables or observations are in the data set. However, if the data set has hundreds of observations and dozens of variables, a great deal of system resources may be used up in storing the macro variables. The CALL SET statement offers a solution to this problem.

CALL SET

The CALL SET statement can be used to replace the SET statement in a SAS DATA step. However, it requires you to 'OPEN' your data source first and specify the variable attributes that you are inputting into your DATA step. Quite frankly, I believe that the SET function is superior to the CALL SET in *open code*. Why someone would use CALL SET instead of SET is a mystery to me. However, in macro coding, CALL SET is transformed into %SYSCALL SET, which provides an easy way to transfer data set variables into macro variables.

To use the power of %SYSCALL SET, we can rewrite our macro sample from above:

```
%macro doit;
       %let id=%sysfunc(open(sashelp.class));
       %let NObs=%sysfunc(attrn(&id,NOBS));
       %syscall set(id);
       %do i=1 %to &NObs;
              %let rc=%sysfunc(fetchobs(&id,&i));
```

```
        %put # # # Processing &Name # # #;

        %* ---- Analysis Code Goes Here ----- *;

        %end;
%let id=sysfunc(close(&id));
%mend;
```

The first new line uses the OPEN function and the %SYSFUNC to assign a number to the macro variable &id which the SAS system uniquely associates with the data set sashelp.class. On the second new line, &id along with the ATTRN function and &SYSFUNC are used to assign the number of observations in the data set to the macro variable &NObs. The next step uses the %SYSCALL SET on the data set specified by &id (note: do not use the '&' in the %SYSCALL SET).

If you stopped the program at this point, you would think that the %SYSCALL SET did nothing. Actually, behind the scenes, %SYSCALL SET is establishing a link between the data set variables and the SAS macro system. In effect, it creates a macro variable for each variable in the data set. These macro variables have the same names as the associated data set variables. At this point, these macro variables are empty. However, each time through the %DO-loop the FETCHOBS function in the %SYSFUNC loads the variable values in the &i observation of the data set into the associated macro variables. This loop works much like the DATA step with a SET function: going through the input data line-by-line and processing one observation at a time. The last new line closes the data set you opened at the start.

Now you have all of your data set variables replaced by macro variables, and you do not need any code containing PROC SQL or CALL SYMPUT! Furthermore the macro variables need no suffix, which means that you do not need a macro variable for each observation. This can be a huge savings on system resources.

CONCLUSIONS

For creating a single macro variable, the %LET function is more than adequate. If you want to create a few macro variables, this can be accomplished with CALL SYMPUT in a DATA step or by using a PROC SQL. But if you want several macro variables which correspond to the value of variables in every observation of a data set, the use of a CALL SET within a macro will lead to clear and concise coding. Furthermore, the CALL SET method is less error prone and will use less system resources.

REFERENCES

SAS Institute Inc. 2002-2006, *SAS OnLineDoc® 9*.1.3 "Macro Language Dictionary: %LET Statement", "Functions and CALL Routines: CALL SYMPUT Routine", "Macro Language Dictionary: INTO Clause", "Functions and CALL Routines: CALL SET Routine". http://support.sas.com/onlinedoc/913/docMainpage.jsp.

CONTACT INFORMATION

Your comments and questions are values and encouraged. Contact the author at

William C. Murphy
Howard M. Proskin & Associates, Inc.
300 Red Creek Dr., Suite 330
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com