Paper 56-28

Nine Steps to Get Started using SAS® Macros

Jane Stroupe, SAS Institute, Chicago, IL

ABSTRACT

Have you ever heard your coworkers rave about macros? If so, you've probably wondered what all the fuss was about. This introductory tutorial unravels the mystery of why and how you use the SAS macro facility. In the nine easy steps outlined in this paper, you follow the process of testing your code, substituting macro variables into the code, and turning your SAS program into a macro definition with conditional processing.

INTRODUCTION

The macro facility is comprised of two components:

- macro variables
- macro definitions.

Simply put, the purpose of both of these is text substitution. Utilizing the macro facility can make your code

- easy to maintain
- flexible
- dynamic.

You can write a macro definition to execute code conditionally in nine steps.

The macro definition developed in this tutorial prints hotel information from a SAS data set named EXPENSES that is updated with resorts' costs and amenities as they change. You need to run the program often in order to monitor any changes. In this example, if you submit the program in June, July, or August, you want to print a list of the cheapest hotels (those with room rates less than the average room rate for the entire data set.) Otherwise, you want to print a list of all the hotels in the data.

STEP 1:

WRITE YOUR PROGRAM AND MAKE SURE IT WORKS.

One of the macro tricks is to make sure that your code executes correctly and efficiently before you begin the process of 'macroizing' the program. It is much easier to debug your code when it is not in a macro or does not reference macro variables. So hard code your program and check it out first.

```
proc means data=expenses mean;
   var RoomRate;
run;
```

```
              Analysis Variable : RoomRate

                          Mean
                      ------------
                       221.1090000
                      ------------
```

After looking at the output report to determine the average value, submit the PRINT procedure step with a WHERE statement.

```
proc print data=expenses;
   title 'Lowest Priced Hotels in the
          EXPENSES Data Set';
   footnote 'On June 1, 2003';
   var ResortName RoomRate Food;
   where RoomRate<=221.109;
run;
```

When you run the program on June 1, the following report is generated.

```
      Lowest Priced Hotels in the EXPENSES Data Set

 Obs    ResortName                        RoomRate   Food

  1  Joe's Pretty Good Resort & Bait Shop  165.89  45.5
  2  Larry, Curly, and Motel               178.9     64
  3  Sand And Shores                       215.32    76
  4  Array of Sun Hotel                    210.78    54
  5  Report Resort                         189.87    49
  7  Dew Drop Inn                          197.12    45
 10  Come On Inn                           187.98    36

.
.
.

                     On June 1, 2003
```

REASONS TO USE THE MACRO FACILITY

1. The macro facility can automatically change this program every time you access the data set or the EXPENSES file updates.

STEP 2:

USE MACRO VARIABLES TO FACILITATE TEXT SUBSTITUTION.

One technique for making a change to the program is to replace text by utilizing the REPLACE feature of your operating environment; however, there are downsides of that technique. For example, you might accidentally change a word that you do not want changed, or you might have to avoid that problem by using FIND → REPLACE repetitively. Furthermore, you must repeat the process every time you want to change the program.

Instead of replacing values manually, let SAS macro variables do it for you. Macro variables provide text substitution that you typically use for individual words or phrases, rather than for blocks of code. There are

- automatic macro variables such as the date on which you invoked SAS
- user-defined macro variables created with the %LET statement
- user-defined macro variables created with the CALL SYMPUT routine in a DATA step or with the SQL procedure.

Regardless of how you create the macro variables, to reference them in your program, preface the name of the macro variable with an **&**.

```
options symbolgen;  ①

%let dsn=expenses;  ②
%let varlist=ResortName RoomRate Food;

proc means data=&dsn mean;
   var RoomRate;
run;

/* You still have to look at the report */
/* to determine the average amount.     */

%let average=221.109;
```

```
proc print data=&dsn;
   title "Lowest Priced Hotels in the &dsn ③
          Data Set";
   footnote "On &sysdate9"; ④
   var &varlist;
   where RoomRate<=&average;
run;
```

① The global system option SYMBOLGEN prints in your log how the macro variable is resolved. Without it, your macro substitution is difficult to debug. When your macro variables substitute perfectly, you can turn the option off by submitting
  `options nosymbolgen;`

② To create a macro variable with a %LET statement, use **`%LET`** ***`MacroVariableName`***`=`*`MacroVariableValue`*`;`
The name of a macro variable can be up to 32 characters in length; the value can be up to 64K in length. Typically, the macro variable value is not enclosed in quotation marks since the macro facility assumes that the macro variable value is text. If you do put quotation marks around the word expenses or the value 221.109, the quotation marks are stored as part of the value. When you create macro variables with %LET, the values of the macro variables are stored in an area of memory that is called the global symbol table. When your SAS session is over, the global symbol table is deleted.

③ To reference a macro variable use an **&** immediately before the name of the macro variable. SAS then substitutes the value of the macro variable into the code before it is compiled. When using a macro variable in a TITLE statement (or in fact, any time you would normally use single quotation marks), you must put the quoted text in double quotation marks. The macro facility cannot do text substitution when there are single quotation marks.

④ The automatic macro variable SYSDATE9 is the date on which you invoked SAS. &SYSDATE9 prints the date as 01JUN2003. There is another automatic macro variable, &SYSDATE, that prints the date as 01JUN03.

REASONS TO USE THE MACRO FACILITY
2. Macro variables make your code easy to maintain.

STEP 3:

USE A MACRO FUNCTION TO CAPITALIZE THE NAME OF THE DATA SET.
The case of the macro variable value is the same as the case in which it was typed in the %LET statement. If you want to ensure that the word "EXPENSES" is in upper case in the title, you can use the %UPCASE function. The %UPCASE function is one of many functions that can have macro variables as arguments. Other functions such as %SCAN and %SUBSTR can be used for more complicated applications.

```
options symbolgen;

%let dsn=expenses;
%let varlist=ResortName RoomRate Food;

proc means data=&dsn mean;
   var RoomRate;
run;

%let average=221.109;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";  ①
   footnote "On &sysdate9";
   var &varlist;
   where RoomRate<=&average;
run;
```

① The macro variable functions like %UPCASE can be used in statements in which, by default, DATA step functions cannot be used. DATA step functions are applied to values in a buffer called the Program Data Vector that is created only with the DATA step. Macro functions are applied to text values (often supplied by a macro variable) but not to DATA set variables.

REASONS TO USE THE MACRO FACILITY
3. Macro functions can improve the functionality of macro variables.

STEP 4

CREATE A MACRO VARIABLE FROM A SAS DATA SET.
One limitation of the previous steps is that you must run the MEANS procedure and determine the value of the average room rate before creating the macro variable AVERAGE with the %LET statement. A more useful technique would be to create the average value in a SAS data set and store that value in a macro variable without having to type the %LET.

In addition, you might want to format the date in the TITLE statement rather than using the default format of the SYSDATE9 automatic macro variable.

To store data set variables in a macro variable, you cannot use the %LET statement. One technique that you can utilize is the CALL SYMPUT routine. The CALL SYMPUT routine is used only in a DATA step and can

- create a macro variable that contains constant text value, value of a DATA step variable, or value of a macro variable
- use data step functions to create the value for a macro variable
- assign a value to a macro variable using DATA step logic.

```
options symbolgen;

%let dsn=expenses;
%let varlist=ResortName RoomRate Food;

/* Create a SAS data set rather than */
/* a report with PROC MEANS.         */

proc means data=&dsn noprint;
   output out=stats mean=avg;
   var RoomRate;
run;

data _null_;  ①
   set stats;
   dt=put(today(),mmddyy10.); ②
   call symput('date',dt); ③
   call symput('average',put(avg,7.2));④
run;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   footnote "On &date";
   var &varlist;
   where RoomRate<=&average;
run;
```

① DATA _NULL_ enables you to use the data step syntax without creating a data set. Since you only want to create a macro variable, using DATA _NULL_ is more efficient than creating a data set in addition to the macro variable as the **DATA** *DataSetName* syntax would do.

② The PUT function uses the TODAY() function to retrieve today's date as a SAS date and to create a data step variable containing the date in the form 06/01/2003.

③ In the CALL SYMPUT syntax, the first argument is the name of the macro variable, and the second argument is the value to store in the macro variable. Since the name of the macro variable is a constant, it is enclosed in quotation marks. The value to be stored is a data set variable; therefore, it is not enclosed in quotation marks.
When you create macro variables with CALL SYMPUT, the values of the macro variables are stored in an area of memory that is called the global symbol table. When your SAS session is over, the global symbol table is deleted.
Steps 2 and 3 could be combined into the statement
call symput('date', put(today(), mmddyy10.));

④ The PUT function creates a character value to be stored in the macro variable named AVERAGE. Since macro variable values are stored as text, using the PUT function creates character values thus avoiding the problems of implicit numeric to character conversions.

REASONS TO USE THE MACRO FACILITY
4. Macro variables can get values from a data set.

Instead of creating the DATE macro variable to format the date in the footnote, you can use the %SYSFUNC function. The %SYSFUNC function uses DATA step functions in statements such as the TITLE or FOOTNOTE statements that could not normally use DATA step functions. For example, the footnote could be recoded as:

```
footnote "On %sysfunc(today(), mmddyy10.)";
```

STEP 5:

MAKE THE PROGRAM INTO A MACRO DEFINITION.
Suppose you want to make the program even more dynamic or perhaps you need to make the code reusable. In that case, the first stage is to convert the program into a macro definition.

```
%let dsn=expenses;
%let varlist=ResortName RoomRate Food;

%macro vacation;  ①

proc means data=&dsn noprint;
   output out=stats mean=avg;
   var RoomRate;
run;

data _null_;
   set stats;
   dt= put(today(),mmddyy10.);
   call symput('date',dt);
   call symput('average',put(avg,7.2));
run;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   footnote "On &date";
   var &varlist;
   where RoomRate<=&average;
run;

%mend vacation;  ②
```

```
options symbolgen mprint;  ③
%vacation ④
```

① Start the macro definition with
%MACRO *MacroName;*
where the name of the macro can be up to 32 characters in length.

② End the macro with
%MEND *MacroName;*
By default, when you submit the code for the macro definition, SAS stores the compiled macro routine in a catalog named WORK.SASMACR; however, there are techniques to store the macro definition permanently. For more information about the methods to store a macro definition permanently, consult the *SAS® Macro Language: Reference*.

③ By default, when you use the macro definition, the program for the macro definition is not printed in the log. To help with debugging, use the global system option MPRINT to view the macro code with the macro variables resolved.

④ To invoke the macro definition, use:
%MacroName
Notice there is no semicolon at the end of the **%VACATION** statement. It is wise not to put a semicolon at the end of the macro call in case you end a program statement prematurely. When you call the macro **%VACATION**, by default, SAS looks in the catalog WORK.SASMACR for the VACATION macro and executes the macro code.

REASONS TO USE THE MACRO FACILITY
5. Macro definitions make your code reusable.

STEP 6:

USE PARAMETERS IN THE MACRO AND SPECIFY THE PARAMETERS WHEN THE MACRO IS CALLED.
Rather than providing the macro variable values by using %LET statements, you can use parameters. The names of the parameters are the names of the macro variables used in the program.

```
%macro vacation(dsn,varlist);  ①

proc means data=&dsn noprint;
   output out=stats mean=avg;
   var RoomRate;
run;

data _null_;
   set stats;
   dt=put(today(),mmddyy10.);
   call symput('date',dt);
   call symput('average',put(avg,7.2));
run;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   footnote "On &date";
   var &varlist;
   where RoomRate<=&average;
run;

%mend;

options symbolgen mprint;
%vacation(expenses,ResortName RoomRate)  ②
```

① Using parenthesis, specify positional parameters for the macro variables, separated by commas.

② To call the macro containing positional parameters, specify values within parenthesis, separated by commas, in the order of the parameters. When you use positional parameters, the macro variables are stored in an area of memory called a local symbol table. Since the CALL SYMPUT is executed within this macro definition, the local symbol table also contains the DT and AVERAGE macro variables. The local symbol table is deleted when the macro definition completes execution.

REASONS TO USE THE MACRO FACILITY
6. Macro definitions can use parameters to change and modify code based on those parameters.

STEP 7:

CHANGE THE MACRO DEFINITION TO PROVIDE DEFAULT VALUES FOR THE MACRO VARIABLES.
If you use positional parameters, you must provide those parameters each time you invoke the macro definition. Perhaps you want to specify a default value for those parameters. In that case, try keyword parameters.

```
%macro vacation(dsn=expenses,varlist=_all_);
                                           ①

proc means data=&dsn noprint;
   output out=stats mean=avg;
   var RoomRate;
run;

data _null_;
   set stats;
   dt=put(today(),mmddyy10.);
   call symput('date',dt);
   call symput('average',put(avg,7.2));
run;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   footnote "On &sysdate9";
   var &varlist;
   where RoomRate<=&average;
run;

%mend;

options symbolgen mprint;

%vacation(dsn=hexpenses,varlist=ResortName)②
%vacation(varlist=ResortName RoundOfGolf)  ③

%vacation()  ④
```

① The parameters are now keyword parameters that are given default values using
KEYWORD=*value*
Just like with positional parameters, with keyword parameters, commas separate the parameters. Keyword parameters and positional parameters can both be used; positional parameters must be specified first in the syntax.

② To use the macro definition, specify the value of the parameter by specifying the KEYWORD=*value* syntax, not necessarily in the order they are specified when the macro routine was defined. In other words, you could use

```
%vacation(varlist=ResortName, dsn=hexpenses)
```

When you use keyword parameters, the macro variables are stored in an area of memory called a local symbol table. The local symbol table is deleted when the macro definition completes execution.

③ You do not have to specify a KEYWORD=*value*. If you do not, the default is used. If you specify a value, you must use the KEYWORD=*value* syntax.

④ To use the defaults for both parameters, use the parentheses with no parameters.

REASONS TO USE THE MACRO FACILITY
7. Macro definitions, with keyword parameters, can set defaults for the parameters.

STEP 8

USE PROC SQL TO CREATE THE MACRO VARIABLES.
The previous code uses PROC MEANS to store the average value in a data set variable and the DATA step to create the macro variable. If you use the SQL procedure, you avoid using two steps to store the MEAN statistic in a macro variable.

```
%macro vacation(dsn=expenses,varlist=_all_);

proc sql noprint;  ①
   select mean(RoomRate),  ②
          put(today(),mmddyy10.)  ③
      into :average, :date  ④
      from &dsn;
quit;

proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   footnote "On &date";
   var &varlist;
   where RoomRate<=&average;
run;

%mend;

options symbolgen mprint;

%vacation(dsn=newexpenses)

%vacation(varlist=ResortName)
```

① Use the SQL procedure with the NOPRINT option to create a macro variable without having a printed report. Because the macro variables are created within a macro definition code, they are stored in the local symbol table.

② Selecting MEAN(ROOMRATE) calculates the statistic for the ROOMRATE column for the entire data set.

③ Selecting the constant stores the date in the macro variable DATE.

④ The **INTO :*macroname, :macroname*** syntax stores the values of the average ROOMRATE and today's date in the MMDDYY10. format in the macro variables AVERAGE and DATE.

REASONS TO USE THE MACRO FACILITY
8. The SQL procedure can calculate statistics and create macro variables containing those statistics in one step.

STEP 9:

USE THE %IF…%THEN/%ELSE STATEMENTS WITHIN A MACRO DEFINITION TO EXECUTE CODE CONDITIONALLY.
The final step in printing a list of the lowest priced hotels in the summer and a list of all hotels during the rest of the year is to execute PROC PRINT conditionally. Since conditional logic using IF…THEN/ELSE statements is available only in the DATA step, you cannot execute procedures conditionally. However, that is exactly what you want to do. You want to write a program like this:

```
   If it is June, July, or August, PROC PRINT
   the cheapest hotels; else PROC PRINT all the
   hotels.
```

Using the %IF…%THEN/%ELSE statements in a macro definition makes this possible.

```
%macro vacation(dsn=expenses,varlist=_all_);

proc sql noprint;
   select mean(RoomRate),
          put(today(),mmddyy10.),
          month(today())
      into :average,:date,:mon ①
      from &dsn;

%if &mon=6 or &mon =7 or &mon =8 %then %do;②
   proc print data=&dsn;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
      footnote "On &date";
      var &varlist;
      where RoomRate<=&average;
   run;
%end; ③

%else %do; ④
   proc print data=&dsn;
      title "All Room Information in the
             %upcase(&dsn) Data Set";
      footnote "On &date";
      var &varlist;
   run;
%end;
%mend;

options symbolgen mprint mlogic;  ⑤
%vacation( )
```

① Create one more macro variable in the PROC SQL step. The macro variable, MON, stores the MONTH(today()).
② Use the %IF…%THEN/%ELSE statements to compare the MON macro variable with the constants 6, 7, or 8 in order to print the cheapest hotel when today's date is in June, July or August. (You cannot use the IN operator in the %IF…%THEN/%ELSE statements.)
③ The %END statement ends the %DO group.
④ The %ELSE %DO group executes the PRINT procedure showing all hotels when the macro definition is executed for any month other than June, July, or August.
⑤ The MLOGIC system option prints the value of the condition &mon=6 or &mon=7 or &mon=8 in the log. MLOGIC is useful for debugging.

REASONS TO USE THE MACRO FACILITY
9. Macro definitions enable conditional execution of steps, statements, or parts of statements.

The previous macro program for VACATION executes the PROC PRINT step conditionally. It is not necessary to execute an entire step in %IF…%THEN/%ELSE logic. You can execute just an individual statement. Rewriting the previous program to execute only the TITLE and WHERE statements conditionally gives you the following program.

```
%macro vacation(dsn=expenses,varlist=_all_);

proc sql noprint;
   select mean(RoomRate),
          put(today(),mmddyy10.),
          month(today())
      into :average,:date,:mon
      from &dsn;

proc print data=&dsn;
   footnote "On &date";
   var &varlist;

%if &mon=6 or &mon=7 or &mon=8 %then %do;
   title "Lowest Priced Hotels in the
          %upcase(&dsn) Data Set";
   where RoomRate<=&average;
%end;

%else %do;  ①
   title "All Room Information in the
          %upcase(&dsn) Data Set";
%end;

run;  ②

%mend;
```

① The second TITLE statement is in a %ELSE %DO - %END group to ensure that the semicolon that ends the TITLE statement is compiled correctly.
② The RUN statement is always executed.

CONCLUSION
Using macro variables and/or macro definitions can help you write code that is flexible and dynamic. Though macros will not make your code more efficient, they can certainly make you a more efficient programmer. Begin with hard coding the program, advance though creating macro variables, and finish off with writing a macro definition. Soon you will be creating programs that almost write themselves.

TRADEMARKS

REFERENCES
SAS OnlineDoc®
SAS® Macro Language: Reference
Course notes for SAS Macro class.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Jane Stroupe
SAS Institute
Chicago IL
312/819-6800 ext 8823
Email: Jane.Stroupe@sas.com