How to Use and Not Abuse the Macro Facility

Jane Stroupe

A SAS programmer often falls into one of four categories: those who have no idea how to use the SAS Macro Facility; those who know a little bit about macro variables; those who use macro programs and variables for every situation; and those who use the SAS Macro Facility appropriately. This presentation helps you identify which type of user you are and move from the first three categories into the fourth category. The SAS Macro facility is very powerful. Learn how to use it to your advantage or abuse it at your own risk.

Which User Type Are You?

Have you ever seen a program like this and wondered what it meant?

```
%means()
```

Based on your answer, you would fall into one of the four categories mentioned above. Regardless of your macro status, you can always learn more to improve your macro skills.

What Does it Mean?

The SAS code is calling a macro routine named MACRO using the default parameters. One technique for determining what the program actually does is to use the MPRINT and MLOGIC options.

```
options mprint mlogic;
%means()
```

In the log, you can see the macro routine code as it has been resolved.

LOG

```
303  options mprint mlogic;
304  %means()
MLOGIC(MEANS):  Beginning execution.
MLOGIC(MEANS):  Stored compiled macro in libref CANDY compiled 18OCT12:08:28:02 with V9.3.
MLOGIC(MEANS):  Parameter STATS has value mean median
MLOGIC(MEANS):  %LET (variable name is I)
MLOGIC(MEANS):  %LET (variable name is ONESTAT)
MLOGIC(MEANS):  %DO %WHILE(&onestat ne) loop beginning; condition is TRUE.
MLOGIC(MEANS):  %IF condition not (%upcase(&onestat) in               ALPHA CHARTYPE CLASSDATA
                    CLM COMPLETETYPES CSS CV DATA DESCEND DESCENDING DESCENDTYPES EXCLNPWGT
                 EXCLNPWGTS EXCLUSIVE FW IDMIN KURTOSIS LCLM MAX MAXDEC MEAN MEDIAN
         MIN MISSING MODE N NDEC NMISS NOLABELS NONOBS NOPRINT NOTHREADS              NOTRAP
      NWAY ORDER P1 P10 P20 P25 P30 P40 P5 P50 P60 P70 P75 P80               P90 P95 P99
      PCTLDEF PRINT PRINTALL PRINTALLTYPES PRINTIDS PRINTIDVARS               PROBT Q1 Q3
      QMARKERS QMETHOD QNTLDEF QRANGE RANGE SKEWNESS STACKODS               STACKODSOUTPUT
      STDDEV STDERR SUM SUMSIZE SUMWGT T THREADS UCLM USS               VAR VARDEF) is FALSE
MLOGIC(MEANS):  %LET (variable name is I)
MLOGIC(MEANS):  %LET (variable name is ONESTAT)
MLOGIC(MEANS):  %DO %WHILE(&onestat ne) condition is TRUE; loop will iterate again.
MLOGIC(MEANS):  %IF condition not (%upcase(&onestat) in               ALPHA CHARTYPE CLASSDATA
                    CLM COMPLETETYPES CSS CV DATA DESCEND DESCENDING DESCENDTYPES EXCLNPWGT
                 EXCLNPWGTS EXCLUSIVE FW IDMIN KURTOSIS LCLM MAX MAXDEC MEAN MEDIAN
         MIN MISSING MODE N NDEC NMISS NOLABELS NONOBS NOPRINT NOTHREADS              NOTRAP
```

```
      NWAY ORDER P1 P10 P20 P25 P30 P40 P5 P50 P60 P70 P75 P80               P90 P95 P99
      PCTLDEF PRINT PRINTALL PRINTALLTYPES PRINTIDS PRINTIDVARS               PROBT Q1 Q3
      QMARKERS QMETHOD QNTLDEF QRANGE RANGE SKEWNESS STACKODS               STACKODSOUTPUT
      STDDEV STDERR SUM SUMSIZE SUMWGT T THREADS UCLM USS               VAR VARDEF) is FALSE
MLOGIC(MEANS):  %LET (variable name is I)
MLOGIC(MEANS):  %LET (variable name is ONESTAT)
MLOGIC(MEANS):  %DO %WHILE() condition is FALSE; loop will not iterate again.
MPRINT(MEANS):   proc sql;
MPRINT(MEANS):   select distinct Brand_Name, count(distinct Brand_Name) into :B_N1-:B_N4,:num
from chocolate;
NOTE: The query requires remerging summary statistics back with the original data.
MPRINT(MEANS):   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


MLOGIC(MEANS):  %DO loop beginning; index variable I; start value is 1; stop value is 2; by
      value is 1.
MLOGIC(MEANS):  %LET (variable name is TITLE2)
MPRINT(MEANS):   proc means data=chocolate mean median;
MPRINT(MEANS):   where Brand_Name="Lindt";
MPRINT(MEANS):   title "Lindt Chocolate from Switzerland";
MPRINT(MEANS):   title2;
MPRINT(MEANS):   run;
NOTE: There were 53 observations read from the data set WORK.CHOCOLATE.
      WHERE Brand_Name='Lindt';
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds


MLOGIC(MEANS):  %DO loop index variable I is now 2; loop will iterate again.
MLOGIC(MEANS):  %LET (variable name is TITLE2)
MPRINT(MEANS):   proc means data=chocolate mean median;
MPRINT(MEANS):   where Brand_Name="Sprungli";
MPRINT(MEANS):   title "Sprungli Chocolate from Switzerland";
MPRINT(MEANS):   title2 "Sprungli is a division of Lindt Chocolate";
MPRINT(MEANS):   run;
NOTE: There were 19 observations read from the data set WORK.CHOCOLATE.
      WHERE Brand_Name='Sprungli';
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


MLOGIC(MEANS):  %DO loop index variable I is now 3; loop will not iterate again.
MLOGIC(MEANS):  Ending execution.
```

And if the program is coming from an stored compiled macro where the code has been stored with the macro routine, you can use %COPY to view the macro program itself:

```
%copy means/source
```

The log contains the actual code for the macro program.

LOG

```
302  %copy means/source;
%macro means(Stats=mean median)/store source minoperator;
   %let i=1;
   %let onestat=%upcase(%scan(&Stats,&i));
   %do %while (&onestat ne );

      %if not (%upcase(&onestat) in
              ALPHA CHARTYPE CLASSDATA
              CLM COMPLETETYPES CSS CV DATA DESCEND DESCENDING DESCENDTYPES EXCLNPWGT
              EXCLNPWGTS EXCLUSIVE FW IDMIN KURTOSIS LCLM MAX MAXDEC MEAN MEDIAN
              MIN MISSING MODE N NDEC NMISS NOLABELS NONOBS NOPRINT NOTHREADS
              NOTRAP NWAY ORDER P1 P10 P20 P25 P30 P40 P5 P50 P60 P70 P75 P80
              P90 P95 P99 PCTLDEF PRINT PRINTALL PRINTALLTYPES PRINTIDS PRINTIDVARS
              PROBT Q1 Q3 QMARKERS QMETHOD QNTLDEF QRANGE RANGE SKEWNESS STACKODS
              STACKODSOUTPUT STDDEV STDERR SUM SUMSIZE SUMWGT T THREADS UCLM USS
              VAR VARDEF)
       %then %do;
          %put ERROR: You must specify one of the appropriate statistics.;
          %put ERROR: Look in SAS Help to see what they are.;
          %return;
       %end;
       %let i=%eval(&i+1);
       %let onestat=%upcase(%scan(&Stats,&i));
       %end;

   proc sql;
       select distinct Brand_Name, count(distinct Brand_Name) into :B_N1-:B_N4,:num
           from chocolate;
   quit;

   /* Alternative Method. */
/*
   proc sort data=chocolate(keep=Brand_Name) out=c nodupkey;
     by Brand_Name;
   run;
   data _null_;
      set c end=last;
      call symputx(cats('B_N',_n_),Brand_Name);
      if last then call symputx('num',_n_);
   run;
*/

   %do i=1 %to &num;
      %let title2=%sysfunc(IFC(&&B_N&i=Sprungli,
               title2 "Sprungli is a division of Lindt Chocolate",
               title2));
      proc means data=chocolate &stats;
         where Brand_Name="&&B_N&i";
         title "&&B_N&i Chocolate from Switzerland";
         &title2;
      run;
   %end;
%mend;
```

There's a lot of syntax in this macro program, and the following discussion focuses on how the macro was developed to help understand what the macro is doing and to illustrate best practices.

Best Practices for Developing a Macro Routine

Step 1

The first stage of developing a macro routine is to hard code the program and ensure that it executes correctly and efficiently.

```
proc means data=chocolate mean median;
   where Brand_Name='Lindt';
   title "Lindt Chocolate from Switzerland";
run;
```

This stage is critical, because if the program itself has syntax errors and/or is inefficient, no amount of macro is going to help debugging it.

Step 2

Use the %LET statement to create macro variables for text substitution and the SYMBOLGEN option to log how the macro variables are resolved.

```
options symbolgen;
%let B_N=Lindt;
%let Stats=mean median;
proc means data=chocolate &stats;
   where Brand_Name="&B_N";
   title "&B_N Chocolate from Switzerland";
run;
```

LOG

```
481  options symbolgen;
482  %let B_N=Lindt;
483  %let Stats=mean median;
484  proc means
484! data=chocolate &stats;
SYMBOLGEN:  Macro variable STATS resolves to mean median
485     where Brand_Name="&B_N";
SYMBOLGEN:  Macro variable B_N resolves to Lindt
SYMBOLGEN:  Macro variable B_N resolves to Lindt
486     title "&B_N Chocolate from Switzerland";
487  run;
```

The %PUT _GLOBAL_ statement reports the value of the global macro variables in the log:

LOG

```
478  %put _global_;
GLOBAL STATS mean median
GLOBAL B_N Lindt
```

The macro variables in this example are created in the global symbol table. The global symbol table is deleted when the SAS session ends. If you need to conserve memory, delete these macro variables by using the %SYMDEL statement:

```
%symdel B_N Stats;
```

The %PUT _GLOBAL_ statement shows that the global macro variables have been deleted.

LOG

```
479  %symdel B_N Stats;
480  %put _global_;
```

The %SYMDEL statement must name all the global macro variables to be deleted. If you want to delete all of the global macro variables, you can use the following macro definition. The program utilizes the SASHELP.VMACRO view to create a data set named TEMP that contains the name of all of the user defined global macro variables. The DATA _NULL_ step then uses the CALL SYMDEL routine to delete all of the user defined global macro variables.

```
%macro delvars;
   data temp;
     set sashelp.vmacro;
     where scope='GLOBAL';
   run;
   data _null_;
     set temp;
     call symdel(name);
   run;
%mend delvars;

%delvars
%put _global_;
```

Step 3

Once the program executes correctly, you can change it to a macro routine with parameters: By using named parameters, you can specify default values for the macro variables in the program.

```
options mcompilenote=all;
%macro means(B_N=Lindt,Stats=mean median);
   proc means data=chocolate &stats;
      where Brand_Name="&B_N";
      title "&B_N Chocolate from Switzerland";
   run;
%mend;
```

The option MCOMPILENOTE=ALL generates the following note in your log:

LOG

```
NOTE: The macro MEANS completed compilation without errors.
      9 instructions 320 bytes.
```

Submitting the macro code only creates the macro routine temporarily. The compiled macro is stored in the WORK.SASMACR SAS catalog.

In order to use the macro routine, you must call the macro. The following program calls the macro program using the default parameters.

```
options symbolgen mprint mlogic;
%means()
```

The options provide the following information in your log.

LOG

```
95  options symbolgen mprint mlogic;
496  %means()
MLOGIC(MEANS):  Beginning execution.
MLOGIC(MEANS):  Parameter B_N has value Lindt
MLOGIC(MEANS):  Parameter STATS has value mean median
SYMBOLGEN:  Macro variable STATS resolves to mean median
MPRINT(MEANS):   proc means data=chocolate mean median;
SYMBOLGEN:  Macro variable B_N resolves to Lindt
MPRINT(MEANS):   where Brand_Name="Lindt";
SYMBOLGEN:  Macro variable B_N resolves to Lindt
MPRINT(MEANS):   title "Lindt Chocolate from Switzerland";
MPRINT(MEANS):   run;
NOTE: There were 53 observations read from the data set WORK.CHOCOLATE.
      WHERE Brand_Name='Lindt';
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


MLOGIC(MEANS):  Ending execution.
```

To change the parameter values, you need to specify the name of the macro variable = the value of macro variable. You can change all of the values or just some of the values. For example:

```
%means(B_N=Sprungli,Stats=mean)
```

```
%means(Stats=median)
```

Because the macro variable values are parameter values, they are stored in the local symbol table. A local symbol table is created when the macro routine begins execution and is deleted when execution is complete. Therefore, it is unnecessary to delete the macro variables in order to conserve memory.

Step 4

To store the macro definition permanently as a compiled macro, use the LIBNAME statement to identify in which library to store the compiled macro, the MSTORED option and the SASMSTORE= options. In

addition, use the STORE and SOURCE options on the %MACRO statement to store the compiled macro and its associated code.

To add a second title when the brand of chocolate is Sprungli, use a %IF…%THEN…%DO group.

```
libname Candy 'C:\SAS Talks\Macro Nov 2012';
options mstored sasmstore=Candy;
%macro means(B_N=Lindt,Stats=mean median)/store source;
   proc means data=chocolate &stats;
      where Brand_Name="&B_N";
       title "&B_N Chocolate from Switzerland";
       %if &B_N=Sprungli %then %do;
         title2 "Sprungli is a division of Lindt Chocolate";
       %end;
   run;
%mend;

%means()

%means(B_N=Sprungli,Stats=mean)
```

An alternative to storing the macro as a compiled macro, is to store the macro in a AUTOCALL library. An AUTOCALL library is a collection of external files or SAS catalog SOURCE entries that contain macro definition source code. To define an AUTOCALL library, specify the MAUTOSOURCE SAS system option, use the SASAUTOS= SAS system option to identify AUTOCALL library locations, and store the macro code in the location specified in the SASAUTOS= option.

```
options mautosource
        sasautos=('C:\SAS Talks\Macro Nov 2012',sasautos);
```

Instead of the previous OPTION statement, you could utilize one of the following:

```
/*appends the new location to the end of the SASAUTOS option */
options append=(sasautos=("new-location"));

/*inserts the new location at the beginning of the SASAUTOS
option */
options insert=(sasautos=("new-location"));
```

Step 5

As an alternative to the %IF statement, use the IFC data step function to create a macro variable that contains TITLE2 based on the condition &B_N=Sprungli.

```
libname Candy 'C:\SAS Talks\Macro Nov 2012';
options mstored sasmstore=Candy;
```

```
%macro means(B_N=Lindt,Stats=mean median)/store source;
   %let title2=%sysfunc(IFC(&B_N=Sprungli,
               title2 "Sprungli is a division of Lindt"
                      " Chocolate",
               title2));
   proc means data=chocolate &stats;
      where Brand_Name="&B_N";
      title "&B_N Chocolate from Switzerland";
      &title2;
   run;
%mend;

%means()

%means(B_N=Sprungli,Stats=mean)
```

General form for the IFC data step function:

IFC(logical-expression, value-returned-when-true, value-returned-when-false <,value-returned-when-missing>)

Required Arguments

logical-expression	specifies a numeric constant, variable, or expression.
value-returned-when-true	specifies a character constant, variable, or expression that is returned when the value of logical-expression is true.
value-returned-when-false	specifies a character constant, variable, or expression that is returned when the value of logical-expression is false.

Optional Argument

value-returned-when-missing	specifies a character constant, variable, or expression that is returned when the value of logical-expression is missing.

Simple Example:

```
%let x=2;
%let x=%sysfunc(IFC(&x>5, True, False));
%put &x;
```

Use the SYSFUNC function to execute the IFC data step function in a %LET statement.

%SYSFUNC (*function*(*argument-1 <...argument-n>*)<, *format*>)

function	is the name of the function to execute
argument-1 <...argument-n>	is one or more arguments used by *function*.
format	is an optional format to apply to the result of *function*

Step 6

To create a report for both brands of chocolate, edit the program to create macro variables, &B_N1 and &B_N2 with values of Lindt and Spungli. Creation can be accomplished either by using the SQL procedure with the INTO clause or the DATA step with the CALL SYMPUTX routine. Using %DO group, you can execute the PROC MEANS step for both values of &B_N1 and &B_N2.

```
libname Candy 'C:\ SAS Talks\Macro Nov 2012';
options mstored sasmstore=Candy;
%macro means(Stats=mean median)/store source;

   proc sql;
 ①      select distinct Brand_Name, count(distinct Brand_Name)
            into :B_N1-:B_N2,:num
            from chocolae;
   quit;

   /* Alternative Method. */
/*
   proc sort data=chocolate(keep=Brand_Name) out=c nodupkey;
     by Brand_Name;
   run;
   data _null_;
 ②     set c end=last;
 ③     call symputx(cats('B_N',_n_),Brand_Name);
 ④     if last then call symputx('num',_n_);
   run;
*/

⑤    %do i=1 %to &num;
⑥       %let title2=%sysfunc(IFC(&&B_N&i=Sprungli,
                title2 "Sprungli is a division of Lindt
                        Chocolate",
                title2));
      proc means data=chocolate &stats;
         where Brand_Name="&&B_N&i";
          title "&&B_N&i Chocolate from Switzerland";
         &title2;
       run;
   %end;
%mend;

%means()
```

① The SELECT statement creates three macro variables, B_N1, B_N2 and NUM containing the unique values of **Brand_Name** and the number of unique values of **Brand_Name**.

② The END= option identifies a DATA step variable named LAST that will have a value of 1 for the last observation of the SAS data set, c. Otherwise, LAST has a value of 1.

③ The SYMPUTX creates the macro variables.

CALL SYMPUTX(*macro-variable, text***);**

Where the name of the macro variable is created by utilizing the CATT function to concatenate the text value of 'B_N' to the value of _N_.

The SYMPUTX routine assigns a maximum of 32K characters to the macro variables. Numeric variables are automatically converted to character using the BEST. format, with a field width up to 32 characters, and leading and trailing blanks are removed from both arguments.

④ For the last observation of the SAS data set, c, the SYMPUTX creates the macro variable NUM containing the number of observations in c.

⑤ The %DO loop will execute for each value of I from 1 to the value of &NUM (the number of observations in the CHOCOLATE SAS data set.

⑥ Using indirect references to macro variables, the values of B_N1 and B_N2 are resolved. Indirect references are used when there are multiple ampersands preceding a name. Two ampersands (&&) resolve to one ampersand (&) and the macro processor rescans an indirect reference, left to right, from the point where multiple ampersands begin until no more references can be resolved.

For example, when i=1, the macro reference &&B_N&I first evaluates to &B_N1. It is then re-evaluated as Lindt.

Step 7

Finally, if you are going to allow a user to execute this macro routine, you might want to do some error checking of the parameter STATS.

```
libname Candy 'C:\Users\Owner\Documents\SAS Talks\Macro Nov 2012';
options mstored sasmstore=Candy;
①%macro means(Stats=mean median)/store source minoperator;
②  %let i=1;
③  %let onestat=%upcase(%scan(&Stats,&i));
④  %do %while (&onestat ne );

⑤       %if not (&onestat) in
              ALPHA CHARTYPE CLASSDATA
              CLM COMPLETETYPES CSS CV DATA DESCEND DESCENDING
              DESCENDTYPES EXCLNPWGT
              EXCLNPWGTS EXCLUSIVE FW IDMIN KURTOSIS LCLM MAX MAXDEC
              MEAN MEDIAN
              MIN MISSING MODE N NDEC NMISS NOLABELS NONOBS
              NOPRINT NOTHREADS
              NOTRAP NWAY ORDER P1 P10 P20 P25 P30 P40 P5 P50 P60
              P70 P75 P80
              P90 P95 P99 PCTLDEF PRINT PRINTALL PRINTALLTYPES
              PRINTIDS PRINTIDVARS
              PROBT Q1 Q3 QMARKERS QMETHOD QNTLDEF QRANGE RANGE
```

```
               SKEWNESS STACKODS
               STACKODSOUTPUT STDDEV STDERR SUM SUMSIZE SUMWGT T
               THREADS UCLM USS
               VAR VARDEF)
⑥        %then %do;
            %put ERROR: You must specify one of the
                         appropriate statistics.;
             %put ERROR: Look in SAS Help to see what they are.;
⑦            %return;
         %end;
⑧        %let i=%eval(&i+1);
         %let onestat=%upcase(%scan(&Stats,&i));
         %end;

   proc sql;
       select distinct Brand_Name, count(distinct Brand_Name)
                   into :B_N1-:B_N4,:num
              from chocolate;
   quit;

   /* Alternative Method. */
/*
   proc sort data=chocolate(keep=Brand_Name) out=c nodupkey;
     by Brand_Name;
   run;
   data _null_;
      set c end=last;
      call symputx(cats('B_N',_n_),Brand_Name);
      if last then call symputx('num',_n_);
   run;
*/

   %do i=1 %to &num;
      %let title2=%sysfunc(IFC(&&B_N&i=Sprungli,
               title2 "Sprungli is a division of Lindt Chocolate",
               title2));
      proc means data=chocolate &stats;
         where Brand_Name="&&B_N&i";
          title "&&B_N&i Chocolate from Switzerland";
         &title2;
      run;
   %end;
%mend;
```

① The error checking is using the macro IN operator (see step ⑤); therefore, you must specify the MINOPERATOR on the %MACRO statement or as a system option. The default delimiter for the IN list is a blank. If there is another delimiter, you must also specify the MINDELIMITER= option.

② The %LET statement is creating the macro variable I that will allow the %DO loop to extract each individual statistic for comparing against the list of appropriate statistics for the MEANS procedure.

③ The macro variable ONESSTAT is created by scanning the value of the parameter for the first 'word' where a blank is a delimiter and the upper casing it for comparison.

④ The %DO %WHILE loop will continue to process the %IF statement as long as the macro variable &ONESTAT has a value.

⑤ The %IF statement is using the IN operator to check against valid values of statistics on the MEANS statement. Notice that the IN operator uses slightly different syntax from the IN operator in the DATA step IF statement or the WHERE statement. First, there are no parenthesis around the in conditions. Second, there are no quotation marks around the individual values being compared. Third, the NOT operator is applied outside of the IN operator. For example, in the DATA step you would use "if x not in ('a','b','c);'; however in the macro facility you would use '%if not (&x in a b c).

⑥ The %PUT statements in the %DO group are writing the word ERROR followed by the text and coloring them red so they are visible.

⑦ The %RETURN statement is stopping the macro from executing if the %IF condition is true. Using this statement, you avoid having to utilize the %ELSE %DO statement.

⑧ Finally, increase the value of the macro variable I by using the %EVAL function (the macro facility does not do arithmetic) and find the next statistic in the STATS parameter before returning to check the %DO %WHILE condition.

Now test the program using the default statistics, good statistics, and bad statistics.

```
options mprint mlogic;
%means()
%means(Stats=average)
%means(Stats=n nmiss)
```

Step 8

Finally, the macro is ready for distribution. Just make sure you turn off the SYMBOLGEN, MPRINT and MLOGIC options as they are taking resources!

```
options nosymbolgen noprint nomlogic;
```

Conclusion

The macro facility is extremely powerful for creating routines that can run themselves and generate data and/or reports. Using macro variables can make your programming tasks more flexible and make you a more efficient programmer if you are executing the same program multiple times with different values for a variable. Creating a macro routine for frequent tasks also enables you to be a more efficient programmer. However, using the macro facility for every program is unnecessary. So think carefully about what your program needs to do and how often it needs to be changed. And if the answer is often, then the macro facility is the tool for you!

Contacts

Jane Stroupe
SAS Contract Instructor
jgsstroupe@gmail.com