SAS® Software Macros- You're Only Limited by Your Imagination

Peter Parker, US Dept. of Commerce, Washington, DC
Peter_parker@ita.doc.gov

Abstract

SAS Macros have many uses besides passing parameter values through programs. You can craft elaborate automated queries, write more user-friendly applications, and create maintainable code. I am going to demonstrate four advanced uses for SAS Macros:

1. User Interface- Macro Windows
Using simple window forms, you can create queries to produce user-customized results.

2. Code libraries- Using existing common code stored in library files, you can "include" (i.e., share) standardized code.

3. Top-Down Structured Programming- Using the best conceptual feature from COBOL, a programmer can selectively run parts of a program, by dividing up the code into macro routines. Also, you can read large SAS programs easier, since the code can be divided up into module-like routines, with the last one being the "driver" of the other modules. You
can then selectively run them, similar to the sequential methodology used in well-written COBOL code.

4. Batch Processing- You can create customized SAS desktop icons to run specialized tasks.

1. What's a SAS Macro and why make your code more complex? (Functionality vs. Complexity)

SAS macros are short-cut code. Instead of explicitly writing code, one can use the Macro "sub-language" to generate code. In this paper, programs will be examined that produce the simple report shown in **Figure1**.

Figure 1- Sample Output Generated with Data _NULL_

```
Keep up the great work !                    Keep up the great work!
 Keep up the great work !                    Keep up the great work!
  Keep up the great work !                    Keep up the great work!
   Keep up the great work !                    Keep up the great work!
    Keep up the great work !                    Keep up the great work!
     Keep up the great work !                    Keep up the great work!
      Keep up the great work !                    Keep up the great work!
       Keep up the great work !                    Keep up the great work!
        Keep up the great work !                    Keep up the great work!
         Keep up the great work !                    Keep up the great work!
          keep up the great work !
```

To produce the preceding report that repeats a message in a "v-shaped" pattern, one could write each line explicitly in a Data _*Null*_ data step, generating several lines of code (unmanageable, if one decides to repeat this printing pattern for several hundred lines). However, by using do-loops, one could create it using only a few lines of code:

Figure 2. SAS Code for Generating Sample Output of Figure 1.

```
*HARD-CODING EACH LINE;
DATA _NULL_;
FILE  PRINT  NOTITLES;

PUT / @5 "KEEP UP THE
GREAT WORK!" @80 "KEEP
UP THE GREAT WORK!"
/ @5 "KEEP UP THE GREAT
WORK!" @80 "KEEP UP THE
GREAT WORK!"
/ @6 "KEEP UP THE GREAT
WORK!" @79 "KEEP UP THE
GREAT WORK!"
/ @7 "KEEP UP THE GREAT
WORK!" @78 "KEEP UP THE
GREAT WORK!"
/ @8 "KEEP UP THE GREAT
WORK!" @77 "KEEP UP THE
GREAT WORK!"
/ @9 "KEEP UP THE GREAT
```

```
*SHORTCUT USING DO-
LOOPS;
DATA _NULL_;
FILE  PRINT  NOTITLES;
   DO I=5 TO 30;
      J= 85-I;
      PUT / @I "KEEP UP
THE GREAT WORK!" @J
"KEEP UP THE GREAT
WORK!";
   END;
RETURN;
RUN;
```

```
WORK!" @76 "KEEP UP THE
GREAT WORK!"
/ @10 "KEEP UP THE
GREAT WORK!" @75 "KEEP
UP THE GREAT WORK!"
/ @11 "KEEP UP THE
GREAT WORK!" @74 "KEEP
UP THE GREAT WORK!"
/ @12 "KEEP UP THE
GREAT WORK!" @73 "KEEP
UP THE GREAT WORK!"
.
.<one line of code per
line of desired output>
.

RETURN;
RUN;
```

This simple report example is a fine metaphor for SAS Macros. SAS Macros provides tools for writing more maintainable code. For repetitive coding, one can set macro variables at the beginning of a program that would pass parameters throughout the program. Instead of having to edit several lines throughout the program, one only would have to do light editing at the beginning.

For the previous example, one could pass the message "Keep up the great work!" to the report using macros that can be edited at the beginning of the program, rather than throughout the program. This shortcut is especially useful if the macro variables are scattered among several places within the program.

Figure 3- Example of using Macros to pass parameters in code

```
%LET MESSAGE="KEEP UP THE GREAT WORK!";

DATA _NULL_;
FILE  PRINT  NOTITLES;
           DO I=5 TO 30;
              J= 85-I;
              PUT / @I &MESSAGE @J &MESSAGE;
          END;
RETURN;
RUN;
```

In the first line in **Figure 3**, the macro variable "Message" is defined with the value of "KEEP UP THE GREAT WORK!", using the macro function of %let. This value will be substituted in the Put statement, since an "&" in front of a macro variable causes the value of that variable to be used in the SAS code when it is running.

The downside of macros is that they make your program more complicated. You can over-macro a program, and turn it into a nightmare program, especially when you need to tweak it a year later. One must use common sense, or rather, restraint. I have rejected some code solutions, purely on that realization, knowing that a multi-page macro-based solution will become a future problem to laboriously decipher. See **Appendices I** and **II** for examples of using macros both poorly and properly.

However, SAS macros do more than just pass parameters through a program. They can also be used as a user interface to pass parameters through a program interactively. They can share code from other programs for code writing standardization. They can be used to write more logical, easier to read code, using the techniques of Top-down Structured programming. Finally, they can be used to batch process SAS programs by clicking on a customized desktop icon.

2. Using Macro Windows as a simple User Interface.

Figure 4. Example of Macro Window to pass values to macro variable

```
%WINDOW SELMESS COLOR=YELLOW
  #3 @5 "ENTER YOUR MESSAGE" @30 MESSAGE 25
      PROTECT=NO ATTR=HIGHLIGHT COLOR=BLACK
REQUIRED=YES
;
%DISPLAY SELMESS;
RUN;
DATA _NULL_;
```

```
FILE PRINT NOTITLES;
           DO I=5 TO 30;
              J= 85-I;
              PUT / @I "&MESSAGE" @J "&MESSAGE";
          END;
RETURN;
RUN;
```

In this example (**Figure 4**), instead of hard-coding a value for the macro variable "message", one can run the SAS program and have it prompt you for that value. The Macro Window "Selmess" is made up several components:

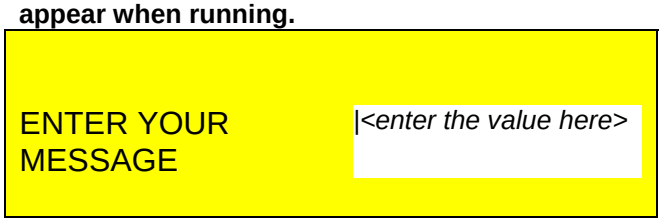
a. %WINDOW SELMESS COLOR=YELLOW-
<The Macro Window is called Selmess and the background color is yellow>

b. #3 @5 "ENTER YOUR MESSAGE"-
< starting at column 5 of line 3 of the screen, the text "Enter YOUR MESSAGE" will appear.>

c. @30 Message 25 PROTECT=NO ATTR=HIGHLIGHT COLOR=BLACK REQUIRED=YES
<at column 30 of that same line, one will be prompted for a value for the Macro variable "Message" which has the following attributes:
i. 25 characters length
ii. PROTECT=NO (value of "Message" can be changed, the default)
iii. ATTR=HIGHLIGHT (displays the field at high intensity)
iv. COLOR=BLACK (specifies that field's color will be black)
vi. REQUIRED=YES (one must enter a value for this field)>

d. %DISPLAY SELMESS;-
<cause the Macro Window Selmess to run>

Figure 5- How the Macro Window example will appear when running.



Here are some other uses for Macro Windows:

1. ask who the user is (based on your name, the program can do some
internal housekeeping, such as deciding how to map the network drives to the libname and filename statements)
2. ask which reports to run
3. ask which variables to print, to sort and to break by in those reports
4. give directions on running production (e.g., where the output will be stored, or when to run certain parts of production)

3. Using Macros to share code (Code Libraries)

Code sharing is another way of recycling code. A good programmer will re-use code whenever possible. Rather than write a program from scratch, I prefer to find a similar program that I've written and then modify it for my new application. SAS macros can enhance this ability to share. Code that will be used for multiple applications can be stored as a file with any extension and stored in a library folder. Using the macro %include statements, this code can be placed in several programs. Here are some of the advantages of using this method of sharing code:

1. If a paragraph of code needs to be changed, one only needs to change it in one place, rather than in all of the programs (such as the record layout in an infile statement).
2. One could save time from copying and pasting code, by putting this "link" to the coding needed.

3. One could enforce standardization of coding by requiring that certain code be shared.

There is a disadvantage to using %include. It will be harder to read and to debug a program, since some of the coding is in another file.

Figure 6. Using shared code with %Include Macro Function

* LIBRARY PROGRAM: (h:\sugicode\prntnull.sas); DATA _NULL_; FILE PRINT NOTITLES; DO I=5 TO 30; J= 85-I; PUT / @I "&MESSAGE" @J "&MESSAGE"; END; RETURN; RUN;	*MAIN PROGRAM:; %WINDOW SELMESS COLOR=YELLOW #5 @5 "ENTER YOUR MESSAGE" @30 MESSAGE 25 PROTECT=NO ATTR=HIGHLIGHT COLOR=BLACK REQUIRED=YES ; %DISPLAY SELMESS; RUN; * prints the message entered in the Windows menu; %INCLUDE "H:\SUGICODE\PRNTNULL.SAS"; RUN;

In **Figure 6**, the Main program will use the code from the program, prntnull.sas, to print the message entered from the Windows Macro. The code in prntnull.sas can be used in any other program, by inserting the code, `%INCLUDE "H:\SUGICODE\PRNTNULL.SAS;".`

4. Writing Top-Down Structured Programming, using the SAS macro language (more advanced code)

Writing a program is only the start of the implementation of a computer-based solution. Maintaining the solution may require effort over many years. Most of us don't write programs that will be easy to figure out years later. We write "coffee-break" programs, something we can churn out quickly (ideally in the time it takes to have a cup of coffee) to do the crucial task at hand. Even if we commented every other line, it still wouldn't be an easy task to read a program for comprehension. That's where top-down structured programming comes into play. A program is broken down into modules, such as "read data", "manipulate data" and "print data". These modules should be placed in a linear order, for ease of reading. A final module, the "driver," would be used to run these modules selectively, based on parameters passed through the program.

Base SAS does not default to such a modular logical design. It's purely linear code. You start at the top and execute code line by line until you reach the last line. In that sense, it is "top-down" but it's not inherently "structured." However, with SAS macros, one could develop this modular structure. You can create a macro for each module, including the final driver module. Every line of code could be in a macro. For instance, all code to read in the data could be contained in a macro called "Readdata". For example:

Figure 7. Sample of code using Macros for Top Down Structured Design

```
%macro readdata;
* this macro is used to read in SAS datasets;
data  testfile;
set  test1;
.
.
%mend readdata;
*;
%macro sortdata;
* this macro is used to sort SAS datasets;
Proc sort;
By   value1;
.
.
%mend sortdata;
*;
%macro printdata;
proc print;
var…..;
.
```

```
.
%mend printdata;
* this macro is used to print SAS reports;
*;
%macro driver;
* this macro is used to decide which macros to run;

* this is the driver of the program;
%readdata;
run;
%sortdata;
run;
%printdata;
run;
%mend driver;
%driver;
run;
```

The preceding example in **figure 7** shows a simple case of running a program. There are four macros defined here, readdata, sortdata, printdata and driver. While the first three are defined in the code, they are not executed until the "driver" macro is run. Within the "driver" macro, are "%"s followed by the name of the macro (e.g., %readdata). These macros will not run until the driver macro is run, %driver, which is not until the next to last line in this example.

Using Macro "IF" statements, one could selectively run macros, based on parameters passed through the macro windows. For instance, one could have three reports- detail, summary, and forecasts, where the execution of each macro runs that particular report. In a macro window program, one could setup a menu where a user can decide which report to run:

Figure 8. Sample of code to selectively run macros

```
     *here is the macro window to be used to query
     the user for which report to run;
QUIT;
RUN;
     * initial settings;

%GLOBAL USEDET USESUM USEFORE;
%WINDOW SELECT COLOR=BLUE
 #5 @5 "OTEXA SPECIAL REQUEST- REPORT"
@60 "&sysday, &sysdate.."
 #12 @20 "TYPE OF REQUEST (mark only one with
X)"
    PROTECT=NO ATTR=HIGHLIGH
COLOR=WHITE REQUIRED=NO
 #15 @15 "DETAIL" @55 USEDET 1
    PROTECT=NO ATTR=UNDERLINE
COLOR=WHITE REQUIRED=NO
 #17 @15 "SUMMARY" @55 USESUM 1
    PROTECT=NO ATTR=UNDERLINE
COLOR=WHITE REQUIRED=NO
 #19 @15 "FORECAST" @55 USEFORE 1
    PROTECT=NO ATTR=UNDERLINE
COLOR=WHITE REQUIRED=NO

 #22 @15 "PRESS ENTER TO CONTINUE"  @55
CONT 1
     PROTECT=NO ATTR=UNDERLINE
COLOR=WHITE REQUIRED=NO

;
RUN;
.
.
%MACRO DETREP;
  *this macro will print the detailed report;
  PROC SORT; BY …..
  .
  .
  %MEND DETREP;
  *;
  *;
%MACRO SUMREP;
  *this macro will print the summary report;
  PROC SORT; BY …..
  .
  .
  %MEND SUMREP;
  *;
  *;
%MACRO FORCREP;
  *this macro will print the forecast report;
  PROC SORT; BY …..
  .
  .
  %MEND FORECREP;


  *'
  * driver to take choices in windows macro and
  determine which macro to execute;

     %MACRO DRIVER;
     * display selection menu;
```

```
%DISPLAY  SELECT;
%IF "&USEDET" = "X" %THEN %DO;
      %DETREP;
      RUN;
%END;
%IF "&USESUM" = "X" %THEN %DO;
      %SUMREP;
      RUN;
%END;
%IF "&USEFORE" = "X" %THEN %DO;
      %FORECREP;
      RUN;
%END;
%MEND DRIVER;
%DRIVER;
      RUN;
```

Here is a detailed explanation of the program example in **figure 8**:

a. USEDET, USESUM AND USEFORE are defined as global macro variables so that the values of these macro variables can be referenced in all defined macros.

b. The macro window, SELECT, creates a screen upon execution where an user can select which report to run, by marking the values of the macro variables, USEDET, USESUM, and USEFORE with an "X"

c. Three macros are defined for running the detailed report (DETREP), summary report (SUMREP), and the forecast report (FORCREP). Note that these programs are only defined and will not run until later in the code.

d. The final macro, DRIVER, determines which of the three macros mentioned in c. will run, depending on the values of the macro variable USEDET, USESUM, AND USEFORE which received user values when the macro window SELECT (see b.) was displayed.

5. Putting it all together (Creating desktop icons for running SAS programs)

Now that you've created a program using SAS macros, SAS macro windows, shared code, and a top-down structured programming methodology, you can simplify running the program by creating an icon on your computer desktop. Rather than clicking on the SAS icon to open SAS software, and then browsing and clicking through menus to find your program, you can streamline the process to the clicking of only one icon.

Suppose you've created a program called test.sas that you store in the following folder- c:\SAS Programs. Here's how to create the icon (assuming a Windows NT, Windows95 or Windows98 operating system):

a. Right click on the SAS program icon and then left click on the "create shortcut" item (this will create another copy of the SAS program icon on your desktop).

b. Right click on this new icon and then left click on the "rename" item. Rename it to "Test."

c. Right click again on this new icon and then left click on the "properties" item. Click on the "shortcut" tab. On the "target" item, you'll see a reference to running SAS- <driveletter>:\SAS\SAS.exe. Append to it a blank space and then add c:\SAS Programs\test.sas. If your SAS software is installed on your C: drive, this code will look like this-
C:\SAS\SAS.EXE C:\SAS Programs\test.sas
(note that these three Microsoft operating systems are not case sensitive)

d. Click on "OK" to save these new settings.

e. When you double-click on this new icon on your desktop, your new SAS job will run, including the SAS macro windows. The SAS log will be stored in a file called <name of program>.log and the SAS output will be stored in <name of program>.lst. In this example where SAS is installed on the C: drive, the two files generated for the test.sas program will be called c:\sas\test.log and c:\sas\test.lst.

5. Concluding Remarks

Good code writing is more than correct syntax, just as good fiction writing is more than error-free typing. One needs to write with obvious logic that can be readily picked up by other programmers for maintenance and also can be transmitted with a minimal amount of coding lines. The point of using computers is to save time. Whether using a do-loop algorithm or a Macro Windows user interface, a well-written program should be created from a minimalist artist's point of view; less is better. Little as possible is best. SAS software's appeal is in its minimalist approach in basic programming functionality. Rather than write a detailed bubble-sort program, you only need a "PROC SORT" statement with a list of variables to sort by. SAS macros take this generated-code concept a step further, and simplifies the running of a SAS job. A SAS macro, properly used, can be the polish to a well-crafted SAS program.

This paper serves only a brief introduction to some of the programming methodologies that SAS macros can enhance. Further reading of the referenced material in **section 7** below will introduce the reader to other similar concepts**. Appendix I** will show a program with SAS macros out of control. **Appendix II** shows how to gain control.

6. References

Art Carpenter, Carpenter's Complete Guide to the SAS Macro Language, Cary, NC: SAS Institute Inc., 1998 242 pp.

SAS Institute Inc., SAS Guide to Macro Processing, Version 6, Second Edition, Cary, NC: SAS Institute Inc., 1990, 319 pp.

SAS Institute Inc., SAS Macro Language: Reference, First Edition, Cary, NC: SAS Institute Inc., 1997. 304 pp.

Appendix I- Making a Macro Monster Program (some ugly programming)

Be warned that this over-macroed coding, while it will work (at least until you try to maintain it) will not be pretty.

Figure 9- Example of Ugly Programming

```
%GLOBAL DATA NULLS YES D1 D2 D3 D4 D5 D6 D7 D8
D9 D10 J85;
%LET DATA=DATA;
%LET NULLS=_NULL_;
%LET YES=RUN;
%LET J85=85-I;
&YES;
%MACRO A123;
&DATA &NULLS;
FILE  PRINT  NOTITLES;
          DO I = 1 TO 30;
             J= &J85;
             PUT / @I "&MESSAGE" @J "&MESSAGE";
         END;
```

```
RETURN;
%MEND A123;
*;
%WINDOW ABC COLOR=YELLOW
  #5 @5 "ENTER YOUR MESSAGE" @30 MESSAGE 25
      PROTECT=NO ATTR=HIGHLIGHT COLOR=BLACK
REQUIRED=YES
;
%DISPLAY ABC;
&YES;
%A123;
&YES;
```

Salient points on the Ugly program in **figure 9**:

1. %Global statement
   a. Here are non-intuitive names for macro variables that have no meaning in themselves (J85? Yes?).
   b. Why include macro variables that won't even be used (D1, D2…) or don't need to be used?
2. %let data=data; Why rename a SAS keyword with a macro variable?
3. %let Nulls=_null_ ;Why rename a SAS keyword with a similar named macro variable?
4. %let Yes=run; Why rename a SAS keyword with a totally different named macro variable?
5. %let J85=85-I; Why use a macro code to replace code unless one wants to change it later? Here, the code is not likely to be changed later.
6. &Yes; "Run;" should be used instead. An unnecessary layer of complexity has been added.
7. &MACRO A123; This is the only macro in the program. Why have it?
8. &Data &Nulls; Why not "Data _ Null _ "? This is unnecessary and confusing.
9. J=&J85; Why not "J= 85-I" ? While this code is syntactically correct, it's not how SAS macros where meant to be used.
10. %Window ABC…. This Macro window appears first in the running of the program but appears after Macro A123 in the code. This is not top-down design.

Overall, this program will run correctly, but it can be written much simpler (see **Appendix II**). As well, when a programmer tries to maintain this program a year later, he'll spend much time trying to figure out the meaning of these macro variables names and how they're used. SAS macros can be a major programming aid, if properly used.

Appendix II- Taming the beast (an alternative to some ugly programming)

Figure 10. Cleaner Example of Macro Window to pass value to macro variable

```
%WINDOW SELMESS COLOR=YELLOW
  #3 @5 "ENTER YOUR MESSAGE" @30 MESSAGE 25
      PROTECT=NO ATTR=HIGHLIGHT COLOR=BLACK
REQUIRED=YES
;
%DISPLAY SELMESS;
RUN;
DATA _NULL_;
FILE PRINT NOTITLES;
           DO I=5 TO 30;
              J= 85-I;
              PUT / @I "&MESSAGE" @J "&MESSAGE";
          END;
RETURN;
RUN;
```

The example shown in **figure 10** produces the same output as the ugly code in **figure 9**, but in fewer lines and in clearer syntax. A Macro window (selmess) will prompt the user for a message and then the Data _Null _ statement will print it out in a "V" shaped pattern.