Macro = Magic; Where Number of Graphics = Many; Customizing Your Graphics Code with Macros

David Mintz, U.S. EPA, Research Triangle Park, NC
Nicole G. Mintz, Analytical Sciences, Inc., Durham, NC

Abstract

Macros can make rerunning SAS® code almost effortless. Once a macro is set up properly, all the arguments can be changed in a single line of code. This is especially helpful when generating graphics. When applied to graphics code, macros can generate the same graphic multiple times with different titles, axis labels, input data sets, etc. just by changing these attributes in the macro statement. This paper will show you, step by step, how to create a macro for graphics use.

Introduction

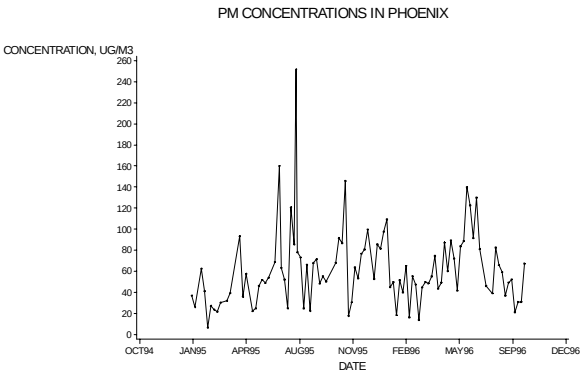
The ability to display information effectively has become an important tool in today's business world. As a result, we SAS programmers often spend hours perfecting code that generates the optimum graphic. Once our code is finished, we put it away. And almost always, we return to that code to generate another graphic of similar nature. If our code has not made use of macros, we have to manually scan the code and change variables that affect titles, axis labels, input data sets, and so on. Not only can this process be time consuming, it also opens the door to human error. The best solution is to convert the code into a SAS macro. The macro can then be called with a single line of code which identifies the macro and the parameter values to be used upon execution. Using a simple example, we will demonstrate, step by step, how to convert a simple graphics program into a macro.

The SAS code

Given a SAS data set containing a SAS date and its corresponding air pollutant concentration for particulate matter (PM) from January 1995 to December 1996, the following program generates the graphic in Figure 1.

```
proc gplot data=pm;
     format date monyy5.;
     plot pm*date /  vaxis=axis1
                     haxis=axis2
                     skipmiss;
     axis1
       label=(f=swiss h=1.75
              'CONCENTRATION, UG/M3')
       value=(f=swiss h=1.5)
       order=(0 to 260 by 20)
       minor=none;
     axis2
       label=(f=swiss h=1.75)
       value=(f=swiss h=1.5)
       minor=none;
symbol1 I=join c=black v=dot h=.25;
title 'PM CONCENTRATIONS IN PHOENIX';
run;
```

Figure 1



In a few easy steps, we will convert this program into a macro and along the way demonstrate some key features of the SAS Macro Facility.

From SAS code to SAS macro

Step 1

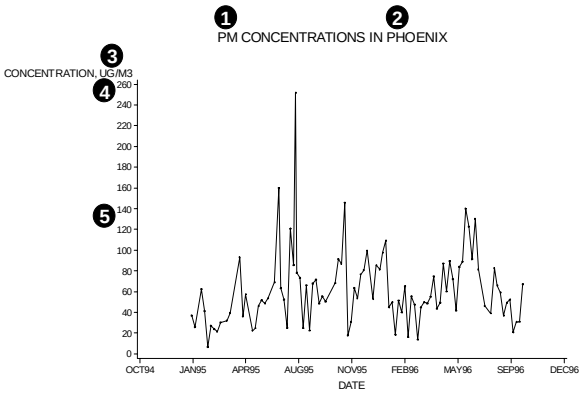
Before converting your program into a macro, **make sure the code works**. It is a lot easier to debug your code without having to debug the macro too. In fact, it is always good practice to design your code and make sure it works before introducing macro variables.

Step 2

Look at your graphic and **determine which attributes change**. In the example program, we would like to be able to change the input data set, the pollutant in the title, the city in the title, the y-axis label, the height of the y-axis, and the y-axis increment. There are many other attributes we *could* change, but we will stick with these to keep the example simple. These attributes are

identified in Figure 2.

Figure 2



Step 3

Choose parameter variable names for the attributes and determine their values. In our example, we would have the following list.

Parameter Variable	Values
❶ poll	pm, co, ozone
❷ city	Phoenix, Boston, etc.
❸ ylabel	ug/m3, ppm
❹ yheight	(depends on data)
❺ yincrmt	(depends on data)
❻ datain	pm, co, ozone
❼ plotvar	pm, co, ozone

Notice that the values are the same for the variables ❶,❻, and ❼ named *poll* (the pollutant in the title), *datain* (the input data set), and *plotvar* (the variable being plotted). If you name your data sets appropriately, you can use the same parameter variable to title and label various parts of your graph. Here, we will use one variable (*poll*) to identify the input data set, to title the graphic, and to plot along the y-axis. Therefore, we will not need to use additional variables for *datain* and *plotvar*. *Poll* will reference the values for those variables.

It is a good idea to put this list in a comment at the beginning of the macro. This will help you remember the fixed parameters, and can also help co-workers who may use this macro. When commenting inside a macro program, use

```
%*comment;
```

instead of

```
/* comment */.
```

You can use the latter, but the former is sent directly to the macro processor, which helps your program run slightly more efficiently.

Step 4

Define the macro. A macro program is defined with two statements. The %MACRO statement identifies the beginning of the macro, names the macro, and defines the parameter variables. The %MEND statement identifies the end of the macro. In our example, we will add the bolded statement to the beginning of the macro.

```
%macro timeplot(poll,city,ylabel,yheight,yincrmt)
proc gplot data=pm;
     format date monyy5.;
     plot pm*date /  vaxis=axis1
                     haxis=axis2
                     skipmiss;
     axis1
       label=(f=swiss h=1.75
              'CONCENTRATION, UG/M3')
       value=(f=swiss h=1.5)
       order=(0 to 260 by 20)
       minor=none;
     axis2
       label=(f=swiss h=1.75)
       value=(f=swiss h=1.5)
       minor=none;
symbol1 I=join c=black v=dot h=.25;
title 'PM CONCENTRATIONS IN PHOENIX';
run;
```

We have named our macro *timeplot* and have listed the parameter variables inside the parentheses following the name. There are reserved words which you cannot use to name your macro because the macro facility would recognize them as macro statements. Here is a list of the reserved words.

```
ABEND   ABORT   ACT     ACTIVATE  BQUOTE   BY
CLEAR   CLOSE   CMS     COMANDR   COPY     DEACT
DEL     DELETE  DISPLAY DMIDSPLY  DMISPLIT DO
EDIT    ELSE    END     EVAL      FILE     GLOBAL
GO      GOTO    IF      INC       INCLUDE  INDEX
INFILE  INPUT   KEYDEF  LENGTH    LET      LIST
LISTM   LOCAL   MACRO   MEND      METASYM  NRBQUOTE
NRQUOTE NRSTR   ON      OPEN      PAUSE    PUT
QSCAN   QSUBSTR QUOTE   QSYSFUNC  QUPCASE  RESOLVE
RETURN  RUN     SAVE    SCAN      STOP     STR
SUBSTR  SUPERQ  SYSCALL SYSEVALF  SYSEXEC  SYSFUNC
SYSGET  SYSRPUT THEN    TO        TSO      UNQUOTE
UNSTR   UNTIL   UPCASE  WHILE     WINDOW
```

Next, we will add the bolded statement to the end of the macro.

```
%macro timeplot(poll,city,ylabel,yheight,yincrmt)
proc gplot data=pm;
     format date monyy5.;
     plot pm*date /  vaxis=axis1
                     haxis=axis2
                     skipmiss;
     axis1
       label=(f=swiss h=1.75
              'CONCENTRATION, UG/M3')
       value=(f=swiss h=1.5)
       order=(0 to 260 by 20)
       minor=none;
     axis2
       label=(f=swiss h=1.75)
       value=(f=swiss h=1.5)
       minor=none;
symbol1 I=join c=black v=dot h=.25;
title 'PM CONCENTRATIONS IN PHOENIX';
run;
%mend timeplot;
```

This line indicates the end of the macro named *timeplot*. Using the name of the macro in this statement is optional. If you have multiple macros in the same program, naming them might help you stay organized.

Step 5

Convert attributes to appropriate macro variables. Go through your code and convert the appropriate attributes. In our program, we will insert the bolded macro variables in place of the attributes.

```
%macro timeplot(poll,city,ylabel,yheight,yincrmt)
proc gplot data=&poll;
     format date monyy5.;
     plot &poll*date /  vaxis=axis1
                        haxis=axis2
                        skipmiss;
     axis1
       label=(f=swiss h=1.75
              "CONCENTRATION, &ylabel")
       value=(f=swiss h=1.5)
       order=(0 to &yheight by &yincrmt)
       minor=none;
     axis2
       label=(f=swiss h=1.75)
       value=(f=swiss h=1.5)
       minor=none;
symbol1 I=join c=black v=dot h=.25;
title "&poll CONCENTRATIONS IN &city";
run;
%mend timeplot;
```

Double quotes are necessary in the title and label statements. If single quotes are used, the macro variables inside the quotes will not be resolved, and the title will print the text exactly as it appears, ampersands and all.

Step 6

Call the macro. Once the macro has been defined and submitted, you can call it using various parameter values. Use the following coding structure to call your macro and pass along parameter values.

```
%macro-name(parameter values)
```

Because this is not a SAS statement, a semicolon is not required. In fact, placing a semicolon after a macro call may insert an inappropriate semicolon into the resulting program and cause errors during compilation or execution. In our example, we could call our macro with following bolded line.

```
%macro timeplot(poll,city,ylabel,yheight,yincrmt)
proc gplot data=&poll;
     format date monyy5.;
     plot &poll*date /  vaxis=axis1
                        haxis=axis2
                        skipmiss;
     axis1
       label=(f=swiss h=1.75
              "CONCENTRATION, &ylabel")
       value=(f=swiss h=1.5)
       order=(0 to &yheight by &yincrmt)
       minor=none;
     axis2
       label=(f=swiss h=1.75)
       value=(f=swiss h=1.5)
       minor=none;
symbol1 I=join c=black v=dot h=.25;
title "&poll CONCENTRATIONS IN &city";
run;
%mend timeplot;
%timeplot(ozone,Boston,ppm,0.15,0.05)
```

We have listed the values to be assigned to each parameter variable inside the parentheses following the name. Specifically, *poll*=ozone, *city*=Boston, *ylabel*=ppm, *yheight*=0.15, and *yincrmt*=0.05. It is important to list the values in the same order as their respective parameter variables appear in the %MACRO statement. Because the order is important when listing the parameter values like this, they are called *positional* parameters. If you wish to omit a positional parameter value, be sure to leave it blank in the list. For example, if we wanted to omit the second parameter value, we would use the following line to call the macro.

```
%timeplot(ozone,,ppm,0.15,0.05)
```

Optional: Using Keyword Parameters

Alternatively, you can use *keyword* parameters when defining your macro to assign default values. Keyword parameters use an = sign to directly assign values. If we wanted to make Boston the default city for all our graphics, we would define our macro as follows.

```
%macro timeplot(poll=,city=Boston,ylabel=,
                yheight=,yincrmt=);
```

Since we do not want default values for the other variables, null values are assigned. We could call this macro as follows. Notice that the order is not important because the values are directly assigned with an = sign. Boston is the value for *city* unless another value is

assigned.

```
%timeplot(ylabel=ppm,yheight=0.15,
          yincrmt=0.05,poll=ozone)
```

Additionally, you can define your macro using both positional and keyword parameters. However, all positional parameter variables must be listed before any keyword parameter variable. Our macro definition, then, would be the following.

```
%macro timeplot(poll,ylabel,yheight,yincrmt,
                city=Boston);
```

When calling this macro, again it is important to list all positional values before any keyword parameter. In this case, our macro would be called as follows.

```
%timeplot(ozone,ppm,0.15,0.05,city=Detroit)
```

Notice that we have overridden the default value for city with Detroit. If we wanted to leave the default, we would have just omitted the *city*= assignment.

Summary

Effective macro programming reduces human error, eliminates redundant code, allows easily understood programs to be shared among co-workers, and ultimately saves you time. This paper demonstrates how to transform a simple program into a macro which can be called with a single line of code. In the process, it examines several features of the SAS Macro Facility. If you would like to better understand how the macro facility works, we recommend attending the SAS Macro Language Training Course offered by SAS Institute.

References

SAS Institute Inc. (1996), *SAS Macro Language Course Notes*, Cary, NC: SAS Institute Inc.

Acknowledgments

We would like to thank Michelle Wayland (U.S. EPA) for reviewing this paper and providing excellent feedback, and Amy Peters (SAS Institute, Inc.) for answering questions concerning the SAS Macro Facility.

Further Information

If you have questions or comments, please contact David Mintz at mintz.david@epamail.epa.gov or the following mailing address:
AQTAG (MD-14)
U.S. EPA
Research Triangle Park, NC 27711

or

Nicole Mintz at ngm@asciences.com or the following mailing address:
Analytical Sciences, Inc.
2605 Meridian Parkway, Suite 200
Durham, NC 27713