Paper #HW03

An Introduction to Macros

Deb Cassidy

Abstract
A search in the proceedings for SUGI 24-28 for the word "macro" had over 1,000 hits. Why are macros so popular? A quick glance through the papers showed many were titled in the form "A Macro to ... ". These macros were developed to write complex code once and then use it over and over again with different datasets. Macros are also useful for writing code conditionally. For example, if your data meets Condition X, then run Part A of the program. Otherwise, run Part B of the program. Without macros, you'll need to submit a program, manually check the results and then submit the correct second program.

However, macro code often looks like hieroglyphics to many people. Others have trouble understanding the difference between macro code and other SAS code. This hands-on workshop is intended to give you the basics of macros so you can figure out what &&&i&j really does and why %IF and IF are not the same.

Why Use Macros?

SAS-L is an electronic user group. As I was working on this paper, there was an intense discussion about whether macros were necessary. Of course, there were differing opinions. My opinion, macros are extremely useful but they are also easy to overuse or use in situations where they make the code more complex.

The online help states:
The macro facility is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do common tasks. The macro facility enables you to assign a name to character strings or groups of SAS programming statements. From that point on, you can work with the names rather than with the text itself.

When you use a macro facility name in a SAS program or from a command prompt, the macro facility generates SAS statements and commands as needed. The rest of SAS receives those statements and uses them in the same way it uses the ones you enter in the standard manner.

The above statements from the online help contain an important factor in deciding to use a macro – does it reduce the amount of text you must enter? You should also consider if there are other ways to do this such as BY-statement processing on procedures, using other procedures, using CALL EXECUTE or changing the structure of your data.

The other use where I find a macro often beneficial is writing conditional code. The rest of this paper assumes you have considered the options and macros are the best option for accomplishing your task.

A Simple Example

The following is a very simple example to help you understand that macros are just text substitution - a key fact that many people either never knew or seem to forget. As we cover more complex examples, you'll see there are better ways to do this to have more robust code.

Two things to remember are that &name refers to a macro variable and %name refers to a macro.

Determine what you want for a title and replace “This is my title” in the following code.

```
%let title=This is my title;
 data one;
 set sasuser.houses;
 run;
```

```
proc print data=one;
title "Proc print -- &title";
run;

proc tabulate data=one;
  class style bedrooms;
  tables style*bedrooms,n;
title "My Tabulate results for -- &title - this run";
run;

proc means max data=one;
 title '&title - Results of means';
run;
```

Submit the above code and you'll get three pages of output. The titles will show up as

Output Section	**Title**
proc print	Proc print -- This is my title
proc tabulate	My Tabulate results for -- This is my title - this run
proc means	&title - Results of means

For the first two procs, the text you specified appeared in the title. However, it didn't work for the last proc. What's the difference? Someone might say the difference was because the &title was at the beginning for the last title statement. While that was a difference, the macro variable could be anywhere in the title. You could even have "&title"; and it would work. Look carefully and you'll see the first two titles use double-quotes and the last one does not. This is a key point to remember. Macro variables will not be resolved in single quotes. If you specify single quotes, the macro processor assumes you really want the string containing the ampersand symbol itself.

A Simple Macro
Let's make the above example to the next step. You have the following code that you submit quite often. You just change the data set name and the title.

```
proc print data=put-my-data-set-name-here;
title "PROC PRINT - put-my-title-here from put-my-data-set-name-here";

proc means data=put-my-data-set-name-here;
title "put-my-title-here - Results of means for put-my-data-set-name-here";
```

The general form of a macro is:
%macro macro_name(parameters);
… your code….
%mend macro_name;

The above code would be changed to:

```
%macro print_means(mydsn=,title=);
proc print data=&mydsn;
 title "Proc print -- &title from &mydsn";
run;
proc means max data=&mydsn;
 title "&title - Results of means for &mydsn";
run;
%mend print_means;
```

The parts that you change each time are all now identified as macro variables. Everything else will be the same every time you run the code. There are actually two ways to specify the parameters. The other way is simply

```
%macro print_means(mydsn,title);
```

The disadvantage to this method is that you must always list all the parameters in the correct order. While simple with only two parameters, most macros will be far more complex. The first method also lets you specify a default value for each parameter. I actually have a default of blank in the example.

To run the macro, you write it in the form:

```
%macro_name(parameters);
```

Many new users get confused and try to use

```
%macro macro_name(parameters);
```

The %macro and %mend statements are used in creating the macro. When you need to use the macro, you simply use the macro name itself. The following shows some example code using the macro.

```
%print_means(mydsn=sasuser.houses,title=this is some title);

%print_means(mydsn=sasuser.jobs,title=this is a second test);

data one;
 set sasuser.fitness;
 run;
%print_means(title=this is another order,mydsn=sasuser.fitness);
```

Notice than on the last macro call, I switched the order of the parameters. As noted above, this is an advantage to this style of passing parameters. The other advantage I noted is shown in the following example. I have set the default value of mydsn to be _last_. This is an automatic variable that SAS sets to the name of the last data set created.

```
%macro print_means2(mydsn=_last_,title=);
proc print data=&mydsn;
title "Proc print -- &title from &mydsn";
run;
proc means max data=&mydsn;
 title "&title - Results of means for &mydsn";
run;
%mend print_means2;
```

You can then use the following macro call:

```
%print_means2(title=this is leaving mydsn out);
```

Basic Debugging
Before we move to more complex macros, let's learn a bit about debugging macros.

Submit `%print_means(title=The one without the default);`

Submitting the macro with the default for the data set name will give you errors. The log shows:

```
%print_means(title=The one without the default);
NOTE: Line generated by the invoked macro "PRINT_MEANS".
1      proc print data=&mydsn; title "Proc print -- &title from &mydsn"; run;
proc means max data=&mydsn;  title "&title -
                             -
                             22
1  ! Results of means for &mydsn"; run;
ERROR: File WORK.NAME.DATA does not exist.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds



22: LINE and COLUMN cannot be determined.
NOTE 242-205: NOSPOOL is on. Rerunning with OPTION SPOOL may allow recovery
of the LINE and COLUMN where the error has
              occurred.
ERROR 22-322: Expecting a name.
ERROR: File WORK.NAME.DATA does not exist.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


ERROR 22-322: Expecting a name.
```

Does the log look a bit confusing? Although some people will be able to figure out the problem, several options can help.

Symbolgen will show you how the macro variable resolves.

If you submit your code as

```
Options symbolgen;
%print_means(title=The one without the default);
```

you'll see the following lines now appear in the log.

```
SYMBOLGEN:  Macro variable MYDSN resolves to
SYMBOLGEN:  Macro variable TITLE resolves to The one without the default
```

You are told explicitly that the line in the log came from the SYMOLGEN option. You see that MYDSN resolves to nothing. It other cases people will discover the macro variable resolves but just not how it was expected to resolve. In this case, it was rather obvious that substituting "nothing" in the line

proc print data= ;

was a problem.

Let's assume you had more complicated code and didn't catch that SAS was trying to use the line

proc print data= ;

Options MPRINT will give you the following in the log.

```
MPRINT(PRINT_MEANS):   proc print data=;
ERROR: File WORK.NAME.DATA does not exist.
MPRINT(PRINT_MEANS):   title "Proc print -- The one without the default from
";
MPRINT(PRINT_MEANS):   run;
```

MPRINT write the SAS code for you based on how the variables resolved.

MLOGIC is another option and it controls whether macro execution is traced for debugging. Lines such as the following will now appear in the log.

```
MLOGIC(PRINT_MEANS):  Beginning execution.
MLOGIC(PRINT_MEANS):  Parameter TITLE has value The one without the default
MLOGIC(PRINT_MEANS):  Parameter MYDSN has value
```

Based on this example, you may wonder what the difference is between symbolgen and mlogic since you found out through both methods that MYDSN was blank. Actually, you didn't. MLOGIC told you the value passed to the parameter was "nothing". SYMBOLGEN told you the value of the macro variable resolved to "nothing". In more complex examples, MLOGIC provides lots more detail.

Notice that the log gets longer with more options. You turn off the macro options with a "NO" before the option name as in:

```
Options nosymbolgen nomprint nomlogic;
```

Once you debug your macro, you should turn off the options. Many people like to include a parameter within each macro to make it easy to turn the options on and off as you do your testing. The following method for executing code conditionally can be used to do so.

Executing Code Conditionally
The following is a simple example showing how to execute code conditionally. You want to have proc means run if you specify "means" and proc print run if you specify "print". Pretty simple example, isn't it?

```
%macro condition_ex(condition);

 %if &condition="means" %then %do;
  proc means data=one;
  title "Results of condition &condition";
  %end;

  %else %if &condition="print" %then %do;
    proc print data=one;
     title "Results of condition &condition";
      %end;

%mend condition_ex;

%condition_ex(means);
```

When you run it, you get the following in the log.

```
116  %macro condition_ex(condition);
117
118   %if &condition="means" %then %do;
```

```
  proc means data=one;
  title "Results of condition &condition";
  %end;

  %else %if &condition="print" %then %do;
    proc print data=one;
     title "Results of condition &condition";
    %end;
    run;

%mend condition_ex;

%condition_ex(means);
```

There are no error messages yet there is nothing in the output window. Let's look at the code. %IF-%THEN-%DO-%ELSE-%END are the macro equivalents of IF-THEN-DO-ELSE-END. The difference that many people don't understand is that the macro portion of the code is saying to process the non-macro code when the condition is met. %IF statements have nothing to do with your data. Likewise, IF statements only deal with your data and not the code that is being passed to SAS from the macro language.

At first glance, you might think you misspelled the parameter. You might also think it had to do with the case. The code is case-sensitive and that is often the problem but in this case, everything was spelled correctly and everything used lower case. Let's turn on the option SYMBOLGEN to see if we can figure out why it didn't work. The following lines now appear in the log.

```
SYMBOLGEN:  Macro variable CONDITION resolves to means
SYMBOLGEN:  Macro variable CONDITION resolves to means
```

Condition resolves correctly so that is not the problem. You try MPRINT and get nothing added to the code since MPRINT shows you the code being passed to SAS. So, for some reason there is no code being generated from the macro processor. Let's try MLOGIC.

```
MLOGIC(CONDITION_EX):  Beginning execution.
MLOGIC(CONDITION_EX):  Parameter CONDITION has value means
MLOGIC(CONDITION_EX):  %IF condition &condition="means" is FALSE
MLOGIC(CONDITION_EX):  %IF condition &condition="print" is FALSE
MLOGIC(CONDITION_EX):  Ending execution.
```

Well, this makes it quite clear that both %IF conditions were not met. But why? Do your own substitution and you see the line is:

```
%IF means=”means” %THEN %DO;
```

Those little quotes have caused problems again. You know you need to change the code so it will resolve to one the following. But which one or will both work?

```
%IF “means”=”means” %THEN %DO;
```

or

```
%IF means=means %THEN %DO;
```

If you run the code both ways, you'll discover both work. Remember, the macro processor is dealing with text. However, many people have problems because they get parameter values confused with variable names. While “means” is not a good variable name so it is a bit more obvious, you need to be sure if you are dealing with a variable name or simply a text string as a parameter. Even when you are dealing with variable names, it can be easy to get macro variables confused with your data set variables.

A Looping Example

This is another simple example to show another aspect of the macro language that confuses some people. Let's assume you have many flat files in the same format. Each is identified by a unique number ranging from 1 to the number of files you have. You want to read each one and create a data set with the same number.

Let's assume this is what the code looks like before you turn it into a macro.

```
data file1;
   infile "c:\mypath\mydata1.txt";
   input myfield;
run;

data file2;
   infile "c:\mypath\mydata2.txt";
   input myfield;
run;

data file3;
   infile "c:\mypath\mydata3.xt";
   input myfield;
run;
```

Everything is identical except the 1,2, and 3 used to identify the different files. You look in the manual and find a %DO-%END with looks like DO-END in regular code. You write the following.

```
%macro readdata(howmany);
   %do i=1 %to &howmany;
      data file&i;
         infile "c:\mypath\mydata&i.txt";
         input myfield;
      run;
   %end;
%mend readdata;

%readdata(3)
```

You look in your log and see, among other things, lines containing:

```
ERROR: Physical file does not exist, c:\mypath\mydata1txt.
ERROR: Physical file does not exist, c:\mypath\mydata2txt.
ERROR: Physical file does not exist, c:\mypath\mydata3txt.
```

Well, the looping worked fine but what happened to the period before txt in your file name? It is definitely there in the macro. The macro processor needs a way to know when the macro variable name ends and when the next piece of text begins. In the earlier examples, a blank, quote or semi-colon following the macro variable. These characters can not be part of a variable name so the macro processor knew the macro variable name had finished. But what happens if you want some other character that can be part of a variable name to immediately follow your macro variable? A period is used to end the macro variable name. Thus, what you really wrote was the text string "c:\mypath\mydata" followed by the macro variable &i. followed by the text string "txt".   If you really want to include a period following a macro variable, you need to use 2 periods. The line should have been written as:

```
infile "c:\mypath\mydata&i..txt";
```

You can also do incremental values such as

```
%do i=1 %to &howmany %by 2;
```

However, unlike the data step, the following is not acceptable:

```
%do i=1,5,10;
```

Macro do loops are not as flexible as data step do loops. Should you need code such as the above example which does not work, there are ways to accomplish it but they more complex than the basics this paper covers.

Summary
Do you know everything to write great macros. NO!!! This workshop should be viewed as "getting your feet wet in 6 inches of water." You aren't ready to go swimming in the macro ocean just yet. However, you need to be comfortable with the simple examples and understand how they work before tackling the more complex examples. Odds are that you will use the functionality shown in this paper in most of your macros. You'll also be using a lot of other functionality as well. There are many macro functions which you'll need to learn as well. If the material in this workshop covered everything you needed, then your code could be a good example of the situation where a macro isn't what you really need.

Key points to remember:
Macros process CHARACTER STRINGS
&xxxx is a macro variable while %yyyy is a macro
Macro variables are resolved within double-quotes but not single-quotes
Symbolgen, mlogic and mprint are useful options for debugging macros
Macro language is similar to Base SAS in many ways but there are differences

Recommended References
The online help provides additional details and examples for learning more about macros.

The SAS Books By Users program has recently published the second edition of Art Carpenter's book "Carpent*er's Complete Guide to the SAS Macro Language".* This book contains over 400 pages of material about macros.

Previous SUGI and regional conference presentations also provide a wealth of information about macros. Lex Jansen has created a search engine for the online papers. The search engine can be accessed at www.lexjansen.com/sugi. A different search method for SUGI papers can be accessed at support.sas.com/sugi.

SAS-L is an electronic newsgroup that is also useful for answering questions about your specific macro. There are several ways to access SAS-L. One if through http://www.listserv.uga.edu/archives/sas-l.html It is recommended that you search previous messages before posting a question since a similar question may already have been asked.

Contact info
Deb Cassidy
Debc_sugstuff@usa.com