AN INTRODUCTION TO THE SAS® MACRO LANGUAGE

Gary L. Katsanis, Strong Memorial Hospital

This is an introduction to SAS macros, geared to the new macro programmer who may or may not be fluent in use of the SAS System. It will present general information and specific techniques for the novice to begin using macros in SAS applications. The following topics will be covered:

• a general introduction to the macro system and why it is useful;
• macros and macro variables: their similarities and differences, and when one or the other should be used;
• an introduction to macro expressions and functions;
• tips and techniques for using macros.

In each topic, real-world examples will be provided to illustrate how SAS macros may be used to deal with difficult or awkward programming situations.

Introduction

In the language of the SAS Guide to Macro Processing, Version 6, Second Edition, the macro facility is "a tool for extending and customizing the SAS System and for reducing the amount of text you must enter to do common tasks." This is perfectly true, but it is so general that those who are not familiar with macro processing may not understand how macros can help them. This paper will provide an introduction to macro processing based on uses and real-world applications with the goal of demonstrating how macros can help you to produce better, more efficient, and more effective applications.

The macro facility can be thought of as a programming system. You are all familiar with the DATA Step: a tool for manipulating data files. In an analogous way, the macro facility is a tool for manipulating something: SAS source statements. The macro facility provides you with a flexible and powerful programming language for manipulating the text of your SAS programs - your SAS source statements. The macro facility allows you to work with two new objects: macro variables (identified by the ampersand & ) and macros (identified by the percent sign %).

Macro Variables

A macro variable is a very simple thing. It is a string of text identified by a name. You create a macro variable by using a command and you use it by typing its name where you want the string of text to appear. The same macro variable can be used over and over again once it is defined.

For example, lets say you have a master file called MASTER containing information on clients that is updated periodically and that you keep a copy of the data prior to the last update in a file called OLDMSTR. You may want to run a report program against either the new file or the old file. You might approach this by creating a macro variable called dset and coding the following statement:

%let dset = MASTER ;

Now, you can refer to the macro variable in your code, instead of using the actual data set name; that is, you can code:

PROC PRINT DATA = &dset ;

The macro processor will substitute MASTER for &dset before it runs the source statement, so your program will behave as if you coded:

PROC PRINT DATA = MASTER ;

At this point, we don't have any significant savings. As a matter of fact, its taken MORE statements to perform a simple PROC PRINT. The efficiency of using macros in this sense is that you can use the macro variable &dset in many places, and you can change your source code in ALL of those places by changing the value of &dset in one place; see Figure 1. This is a very common use of macro variables by themselves: writing an extensive program and then customizing it for use on various data files by use
of macro variables. Macro variables can be very long and are limited only by memory considerations. For a general discussion of these techniques, see Septoff, 88.

A macro is a more complicated object than a macro variable. Like a macro variable, the end result of a macro is text that is included as a part of your SAS program, but unlike a macro variable, macros include executable statements that can manipulate information and perform logical control. Macros and executable statements that are a part of the macro processing language are all identified by the percent character %. The %let statement mentioned above is actually an executable macro statement that can appear any place in a SAS program, not just in a macro.

Defining and Calling Macros

You define a macro by coding %macro as the first statement and %mend as the last statement. You give the macro a name on the first statement, and call it later by using that name. For example, you could define a macro called %dset that does the same thing as the &dset variable mentioned above. One way you could do this is as follows:

\[
\begin{align*}
&\text{\%macro dset;} \\
&\text{\hspace{1cm} MASTER;} \\
&\text{\%mend;} \\
\end{align*}
\]

A statement using this macro to identify the file for a PROC PRINT could be:

\[
\begin{align*}
&\text{PROC PRINT DATA = \%dset;} \\
&\text{\%dset(MASTER);} \\
\end{align*}
\]

and the result would be the same as the previous example. You can add flexibility to this by defining macro parameters. Macro parameters are simply macro variables that are available within the macro and nowhere else. For instance, you could define %dset as follows:

\[
\begin{align*}
&\text{\%macro dset(file);} \\
&\text{\hspace{1cm} \&file;} \\
&\text{\%mend;} \\
\end{align*}
\]

In this example, "file" on the command line is a macro parameter: a variable that exists inside the macro only and is deleted when the macro ends. To run this macro, you would code:

\[
\begin{align*}
&\text{PROC PRINT DATA = \%dset(MASTER);} \\
\end{align*}
\]

The macro processor would call up the %dset

---

**Figure 1 - Macro Variable Substitution**

![Macro Variable Substitution Diagram](image-url)
macro, define the &file parameter within it, and substitute the value MASTER. In this case, the &dset macro variable and the %dset macro give identical values when you use them. Moreover, the second example is much longer to type in than just typing "MASTER". Given this, you may ask why you should use macros rather than just macro variables.

Functions and Other Macro Statements

One thing that you can do in a macro that you cannot do with a macro variable is to perform character manipulation and calculation. For instance, titles are often centered on a page; while a TITLE statement will center text, a PUT statement will not. You can get around this by creating a macro. For example:

```sas
%macro center(text) ;
   %let space = %eval(132-%length(text));
   %let indent = %eval(&space/2);
   @&indent n&text"
%mend ;
```

This example does several things. First of all, the %macro statement creates a temporary macro variable called &text; next it calculates values for &space using the %eval function (which treats its arguments as numbers - unless you use %eval, the macro processor will treat all information as character data). Finally, it calculates &indent and then uses &indent and &text to place the parameter in the center of a page using PUT statement syntax. If you called this by coding:

```sas
PUT %center(This is a Title) ;
```

the result would be:

```sas
PUT @58 "This is a Title" ;
```

which would be centered on a 132-column page. You could generalize this by using a second parameter for the page width as well.

Macro Processing

Macro processing is fairly involved; good references exist in the SAS Guide, and in such places as Hendren (89). Some aspects should receive special notice, though.

- Macro processing is triggered by the characters & and %.
- When a & is encountered, the macro processor will search its macro variable tables for the value of the variable in the string immediately following. That value is substituted for the reference to the macro variable.
- When a % is encountered, the macro processor will interpret the following characters as a macro you defined or as a macro function like %let or %eval. Don't name your macros with the same name as a macro function!! When a reference to a macro is found, the macro will be run and the result will be inserted in place of the reference. When a macro function is called, the function will be executed, and the result (if any) substituted for the original reference.
- After the macro processor has interpreted the value of a macro variable, a macro, or a macro function, the result will be scanned again until no more references with & or % exist. The final result is treated as if it were a part of your original SAS program.

Some macros and macro functions may leave code to be executed by the SAS compiler, others like %let leave nothing to be executed but rather perform some utility function. The only line in the %center macro that leaves anything for the SAS compiler is the fourth line, which uses &indent and &text to place the title you specified in the center of a page. For a detailed discussion, please refer to the references mentioned above.

Other Concepts

Two other major concepts in macro processing are use of the %if statement and the %do statement. These are often used together. The %if statement requires a condition and an action. If the condition is true, the action will be processed as macro code, otherwise the action will be skipped. An %else statement exists as well.
For example, you might create the following macro:

```sas
%macro varlist(oldnew);
  %if %upcase(&oldnew) = MASTER
    %then VAR1 - VAR3;
  %else OLDVAR1 - OLDVAR3;
%mend;
```

Now, `%varlist(MASTER)` would resolve to "VAR1 - VAR3" and `%varlist` with any value other than `MASTER` would resolve to "OLDVAR1 - OLDVAR3".

Now, if you coded `%printn(MASTER)` , the macro would resolve to:

```sas
PROC PRINT DATA=MASTER;
  VAR VAR1 - VAR3;
```

Note the reference to another macro, `%varlist`. When the macro processor reaches references like `%varlist`, it will put the first macro on hold, process the new macro in its entirety, and use the result as part of the original macro.

Figure 2 - Macro to List All Members in a Library

```sas
PROC CONTENTS DATA = INLIB.ALL_OUT = MEMLIST;
RUN;

DATA NULL;
LENGTH MVNAME $8;

SET MEMLIST(KEEP=MEMNAME) END = LASTONE;
BY MEMNAME;

IF LAST.MEMNAME THEN DO;
  I + 1;
  MVNAME = "ds" || LEFT(PUT(I),2));

  CALL SYMPUT(MVNAME,MEMNAME);
END;

IF LASTONE THEN CALL SYMPUT("numds",_N_);
RUN;

%macro memlist;
  %do i = 1 to &numds;
    &ds&i
  %end;
%mend;
```

The `%do` statement is coupled with an `%end` statement, and when you use them in a macro, all lines between the two statements are treated as a unit. For example, you might code:

```sas
%macro printn(oldnew);
  %if &oldnew = MASTER %then %do;
    PROC PRINT DATA = &oldnew;
    VAR %varlist(&oldnew);
  %end;
%mend;
```

Summary and Another Example

At this point, we've seen some information about macro variables, macros, and macro functions, as well as their uses and utility in some sections. These techniques can be used as a significant time saver during program development, can minimize the effort to support an application, and can provide a level of flexibility and control that is not available elsewhere in the SAS System.
Creating a List of All Files in a SAS Library

An illustration can be found in the following application. In this application, data sets are created interactively and saved in a permanent SAS Library called `INLIB`. On occasion, the data sets have to be concatenated through a `SET` statement, requiring a list of the files. For example, if there were 3 files called `DATA1`, `DS2`, and `RECORD`, we would need the `SET` statement: "SET DATA1 DS2 RECORD;".

To deal with this, we developed an application and a macro that ran `PROC CONTENTS` and saved the result in a SAS data set. A `DATA` step read the result, saving the member names in a series of macro variables. A macro, `%memlist`, was used to insert the macro variables into our SAS program. The program and macro are presented in Figure 2 on the previous page.

```
   loop. In this case, we want to repeat a statement several times, as determined by the value of the macro variable `%numds`. As in the similar `DATA` Step statement, the counter `%i` is initialized to 1, then incremented automatically; and all statements between the `%do` and the `%end` statement are executed until the counter reaches the value `%numds`.

You also see an example of building the name of a macro variable by use of the values of another macro variable. Consider the string `&&ds&i`. A rule of macro processing states that two ampersands `&&` will always resolve to a single ampersand, `&`. When `&&ds&i` is seen in the first pass through the `%do` loop, `%i` will be 1. The two leading ampersands will resolve to a single ampersand; `ds` (which is not a macro variable) will be untouched, and `%i` will resolve to 1. The result is `&ds1` which itself is a macro variable. The macro processor will recognize this and resolve `&ds1` again, substituting its value, `DATA1`; see Figure 3.

Some Other Issues

Semicolons present a problem in assigning macro variables: since a semicolon is treated as the end of a macro function line, you cannot just embed a semicolon in the middle of a string you would like to define as a macro variable.

---

Figure 3 - Resolution of `%memlist` Macro

The following macro variables exist in the macro variable table.

<table>
<thead>
<tr>
<th>Macro Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;ds1</code></td>
<td><code>DATA1</code></td>
</tr>
<tr>
<td><code>&amp;ds2</code></td>
<td><code>DS2</code></td>
</tr>
<tr>
<td><code>&amp;ds3</code></td>
<td><code>RECORD</code></td>
</tr>
<tr>
<td><code>&amp;numds</code></td>
<td>3</td>
</tr>
</tbody>
</table>

`&&ds&i` is resolved inside the macro loop in two passes by the macro processor:

<table>
<thead>
<tr>
<th>Value of</th>
<th>Value of <code>&amp;&amp;ds&amp;i</code> after 1st pass</th>
<th>Value of <code>&amp;&amp;ds&amp;i</code> after 2nd pass</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%i</code></td>
<td><code>&amp;ds1</code></td>
<td><code>DATA1</code></td>
</tr>
<tr>
<td>1</td>
<td><code>&amp;ds1</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>&amp;ds2</code></td>
<td><code>DS2</code></td>
</tr>
<tr>
<td>3</td>
<td><code>&amp;ds3</code></td>
<td><code>RECORD</code></td>
</tr>
</tbody>
</table>

Each time the macro loop runs, the fully-resolved value of `&&ds&i` is left as a result, so `%memlist` will resolve to "DATA1 DS2 RECORD".
A way to deal with this situation is to use the quoting functions. These are documented in Chapter 10 of the SAS Guide; all quoting functions share a common attribute: they treat special characters like $&, % and ; as if they had no special meaning, or they restore the special meaning to those characters. For example, you might want to assign a macro variable to several lines of SAS code. The statement:

```sas
%let printn = PROC PRINT DATA = &dset; VAR NAME VAR1-VAR3; ;
```

would create an error, since the semicolon after &dset would be considered as the end of the %let

Several SAS options can be used with the macro facility (beyond SYSPARM mentioned above). The MACRO option turns macro processing on and off; make sure you specify "MACRO" before you try to use the macro facility! The options

Figure 4: SAS Log With and Without Macro Information Options

```
10 DATA _NULL_;
11 PUT %center(This is a Title);
12 STOP ;
13 RUN ;

NOTE: The DATA statement used 0.03 CPU seconds and 3122K.

33 OPTIONS MPRINT SYMBOLGEN MLOGIC ;
34 DATA _NULL_;
35 PUT MLOGIC(CENTER): Beginning execution.
36 %center(This is a Title);
37 MLOGIC(CENTER): Parameter TEXT has value This is a Title
38 MLOGIC(CENTER): %LET (variable name is SPACE)
39 SYMBOLGEN: Macro variable TEXT resolves to This is a Title
40 SYMBOLGEN: Macro variable SPACE resolves to 117
41 SYMBOLGEN: Macro variable INDENT resolves to 58
42 SYMBOLGEN: Macro variable TEXT resolves to This is a Title
43 MPRINT(CENTER): @58 "This is a Title"
44 MLOGIC(CENTER): Ending execution.
45 STOP ;
46 RUN ;

NOTE: The DATA statement used 0.02 CPU seconds and 3122K.
```

statement. The right way to do this is to code:

```sas
%let printn = %str(PROC PRINT DATA=&dset; VAR NAME VAR1 - VAR3;);
```

The %str function removes the special significance of the semicolons and the entire line, including semicolons, is stored in the variable called &printn.

There are a series of automatic macro variables that are created for your use. These are documented at the end of Chapter 2 in the SAS Guide. The automatic variables provide information about things like the date, time, environment (interactive or not), status after the most recent command, and so forth. One variable, $&SYSPARM, can have its value set outside of SAS processing in an OPTIONS statement; useful for tailoring a noninteractive or batch program.

MPRINT (which prints the SAS code ultimately generated by a macro), MTRACE (which prints the macro statements as they execute), and SYMBOLGEN (which prints the resolved values of macro variables) are the Version 6 options for macro diagnostics. These are documented in the System Options section of the SAS Language Reference, Version 6, First Edition and also in Appendix 1 of the SAS Guide.to Macro Processing Version 6, Second Edition. See Figure 4 for the effect of these options on your macro execution. Note that messages in the log will be
marked on the left with the option that caused them to be generated.

Be careful about timing in macro applications. For example, a DATA Step is compiled before it executes. Macro variables created by SYMPUT are not available until after the DATA Step is executed, so you cannot use macro variables created in a DATA Step in the same DATA Step. Other timing issues exist as well; see Hendren (89) and Chapters 8 and 9 of the SAS Guide.

Bibliography

Note: this covers only a few of the many articles written about macros and presented in NESUG and SUGI over the years. I have limited this to references written in the last three years and referred to above or found to be particularly useful.

I would like to thank the management of the Office of Clinical Practice Evaluation for their unstinting support, and the staff, especially Daniel Gronell and Keith Skelton, for many valuable suggestions. An earlier version of this paper was presented at the NESUG '90 conference. I would like to thank those whose feedback (both positive and negative) have led to improvements in this version.


SAS is a registered trademark of SAS Institute, Inc., Cary, NC, USA.