The SAS® Macro: An Aid to the User
Robert E. Johnson
Department of Mathematical Sciences, Virginia Commonwealth University, Richmond, VA 23284-2014

This paper is presented as a beginning tutorial on the SAS® macro facility. If you write your own code, either simple or complex then you may find the macro facility useful. The purpose here is to show you some of the fundamentals, share some examples, and start you on the path of learning more about the macro facility. It should be understood that this tutorial does not present an exhaustive coverage of macros. Indeed, to fully appreciate and utilize the power of macros, one should have mastered the art of structured programming. But the rest of us can still find useful applications.

Version 5, 5.16 or later, of the SAS® System allows full use of macros under the CMS, OS, VM/PC, and VSE operating systems. You will have limited use under the VMS operating system. Version 6.06 should bring VMS into the fold. The macros presented here were tested on version 5.18, version 6.03 (for the PC), and version 6.06 (beta).

• Applications

Several macro statements and their applications will be presented in this section. The examples are small so to keep them manageable, but you should think big - imagine them in larger applications.

The %LET Statement

The %let statement is a macro statement which assigns a character string to a macro variable. (This works under VMS.) Notice the "%" symbol. This distinguishes macro statements from the usual code. The %let statement has the form:

\[ \%let macrovariable = string ; \]

The name of the macro variable must conform to the usual naming rules.

Here is an example:

• Use a Macro Variable;

\[ \%let procopt = n mean stderr t; \]
\[ data one; \]
\[ input x @@; cards; \]
\[ 2 4 3 5 7 5 6 3 4 5 6 8 7 9 \]
\[ proc means &procopt; \]

A macro variable procopt receives a null string via a %let statement. The macro variable is used by placing it in the code where you want the string to appear. In order to differentiate between regular variables and macro variables, a & symbol is placed before a macro variable. In this example, procopt is to contain the desired options for the means procedure.

A macro is resolved by substituting the string for the symbolic representation. For example, the above code is resolved to:

\[ \begin{array}{l}
\text{data one;}
\text{input x @@; cards;}
\text{2 4 3 5 7 5 6 3 4 5 6 8 7 9}
\text{proc means;}
\end{array} \]

\[ \text{RESOLVED} \]

Here is the same example with a non-null string assigned:

• Use a Macro Variable;

\[ \%let procopt = n mean stderr t; \]
\[ data one; \]
\[ input x @@; cards; \]
\[ 2 4 3 5 7 5 6 3 4 5 6 8 7 9 \]
\[ proc means &procopt; \]

\[ \begin{array}{l}
\text{data one;}
\text{input x @@; cards;}
\text{2 4 3 5 7 5 6 3 4 5 6 8 7 9}
\text{proc means n mean stderr t;}
\end{array} \]

\[ \text{RESOLVED} \]

The MACRO Statement

The macro statement is used in conjunction with a mend statement to create a SAS macro. The form is:
The macro is placed in the code preceded by a % and the is then resolved. Here is an example:

```
* Create a Macro;
%macro means;
  proc means &procopt;
%mend means;
```

In order to use %means we must first assign a value to procopt. The following application will resolve to code similar to the previous examples:

```
data one;
  input x @@; cards;
  2 4 3 5 7 5 6 3 4 5 6 8 7 9 ;
%let procopt = n mean stderr t;
%mend procopt;
%mend means;
```

Suppose we invoke the macro with the command

```
%means(n mean stderr t)
```

The string n mean stderr t will be assigned to the macro variable procopt and the result will resolve as before. Invoking the command

```
%means
```

will replace procopt with a null string and proc means will display the default statistics.

How much can one parameter do? Consider the following example:

```
data one;
  input x @@; cards;
  2 4 3 5 7 5 6 3 4 5 6 8 7 9 ;
data two;
  input y @@; cards;
  5 1 4 3 5 6 5 4 5 6 5 7 6 ;
%mend means(data = one mean t prt)
```

The desired result is

```
proc means data = one mean t prt;
```

But you would see an error here:

```
keyword parameter DATA was not defined with the macro
```

Keyword parameters are those which start with the name of the macro variable followed by a = symbol and optionally followed by an initial value. The general form is

```
%macro name-of-macro;
  - SAS statements
  - Macro statements
%mend name-of-macro;
```

Macro Parameters

Macro parameters provide a way to pass values to macros variables used within the macro. There are two kinds of parameters: positional and keyword. Keyword parameters will be demonstrated later.

Let's revisit one of the earlier examples and add a parameter to it.
Positional parameters, if any, must appear first in the parameter list. Positional parameters receive their values in the order (position) in which they are specified in the macro invocation. Keyword parameters may appear in any order after positional parameters. Parameters are separated by commas.

Going back to our working example, here is an illustration of a positional and a keyword parameter used together:

```
* Create a Macro (parameters);
%macro means(procopt, dsn=last);
   proc means &procopt data=&dsn;
%mend means;
```

The parameter dsn will be equal to _last_, the symbolic name for the last SAS data set created, unless a different value is passed to it. The command

```
%means(mean t prt, dsn=one)
```

will yield

```
proc means mean t prt data=one;
```

and the command

```
%means(n mean var)
```

will yield

```
proc means n mean var data=_last_;
```

The last procedure will operate on the data set two since it was the last SAS data set created.

---

Using Macros in TITLE Statements

It is possible to customize title statements by placing macro variables within the text of the title. For a macro to be resolved within a quoted text, the double quotes (") must be used. Here is an example:

```
%macro means(procopt, dsn=_last_);
   proc means &procopt data=&dsn;
   title "Statistics for Dataset &dsn";
%mend means;
```

If the command

```
%means(mean t prt, data=one)
```

is submitted, the result will be

```
proc means mean t prt data=one;
```

and the output will be similar to

```
Statistics for Dataset one
Analysis Variable : X
     N  Obs  Mean    T  Prob>|T|
---------  ------  ------  ------  -------
    14  5.286  9.81  0.0001
```

If the command

```
%means(n mean var)
```

is submitted, then the result will be

```
proc means n mean var data=_last_;
```
proc means n mean var data=last;
title "Statistics for Dataset _last_";

RESOLVED

and the output will be similar to

<table>
<thead>
<tr>
<th>Statistics for Dataset <em>last</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Variable : Y</td>
</tr>
<tr>
<td>N Observations N Mean Variance</td>
</tr>
<tr>
<td>7 7 56.0 99.67</td>
</tr>
</tbody>
</table>

The parameter dsn = last may be replaced with dsn=syslast, where syslast is an automatic macro variable supplied by SAS whose value is the two-level name of the last created SAS data set. For example, if syslast were used in the previous example, then the resolved title would read

Statistics for Dataset WORK TWO

%IF - %THEN - %ELSE Statement

It is possible to execute macro statements conditionally or generate SAS code conditionally. One mechanism is the %IF-%THEN-%ELSE structure. Suppose that in our previous example we would like to use one title for specified data sets and another title for default data sets. Consider this macro:

```sas
%macro means(procopt,dsn=last);
proc means &procopt data=&dsn;
title "Statistics for Dataset &dsn";
%if &dsn=last %then
  "Statistics for Last Created Dataset";
%else
  "Statistics for Dataset &dsn";
%mend means;
```

Otherwise, the generic title statement is used. Note again the % which precedes each macro statement. Were this left out, the data step if-then-else would be generated.

We have covered a few of the basic statements in the application section. There will more statements and applications presented in later sections as well.

- Data Step Variables - Macro Variables

It is possible to assign values of data step variables to macros and values of macros to data step variables using the %symput routine and the %symget function. Both of these functions are available in the VMS operating system.

SYMPUT Routine

Consider the following data which is stored on an external file.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BOB</td>
<td>ANNE</td>
<td>RICHARD</td>
</tr>
<tr>
<td>41</td>
<td>30</td>
<td>42</td>
</tr>
<tr>
<td>36</td>
<td>45</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>38</td>
<td>40</td>
</tr>
</tbody>
</table>

File reference name: DATSET

The three alpha-numeric fields on the first line of the data file are converted to values of the macro variables A, B, and C in the first data step via the

If the condition &dsn=last is true, then a data set name must have been specified. In this case the title statement revealing the data set name is used.
The symput routine. In the second data step these macros are used to represent variable names in the input statement. The macros are used again in the procedure step.

SYMGET Function

The symget function may be used to extract values from a macro variable which may in turn, for example, be assigned to a data step variable. The following example shows the usage.

```
data one; input a b c;
call symput('carry',sum(of a b c));
cards;
100 203
;  
data two;
thesum=symget('carry');
```

The summation of the values of variables A, B, and C is assigned to the macro variable carry in the first data step. In the second data step, the value of the macro variable is obtained and assigned to the data step variable thesum.

• %DO Loops

The macro statement %DO allows another way to conditionally execute macro statements or generate SAS code. The general form is:

```
%do macrovar = value %to value;
  SAS Statements
  Macro Statements
%end;
```

In the following example we wish to generate code which will read values from data lines, which follow a cards statement, in such a manner that the resulting data set is in a form needed for the ANOVA procedure. For instance, these data need to be analyzed.

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Gender</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We could write code specifically for this problem, but instead we will construct a macro which will have the flexibility to work with any number of factors, any number of levels for each factor, and any number of observations per cross-classification (same number for each cell - balanced ANOVA). Consider the following macro.

```
%macro anova;
data tempanova;
%doto=1 %to &count;
do &&fact&&i = 1 to &&&limit&&i;
%end;
do i=1 to &n; input y @@;
output end;
%doto=1 %to &count;
end;
%end;
%mend anova;
```

This macro uses several macro variables:

- count: The number of factors.
- fact: The name of the first factor.
- i: The name of the ith factor.
- limit: The number of levels of the first factor.
- limiti: The number of levels of the ith factor.
- n: The number of observations in each cross-classification.
- i: The index variable for the do loop.

Each of these variables, with the exception of i, will require the assignment of values prior to invoking the macro.

The first macro do loop will create as many SAS do statements there values of the macro variable count. The second macro do loop will generate the corresponding end statements.

Here is how this macro may be used to read the data shown above.

```
%mlet count=2; %let n=2;
%mlet fact1=Treat; %let limit1=2;
%mlet fact2=Gender; %let limit2=2;
%anova
cards:
2 3 1 4
4 6 4 3
proc print;
```

248
What does the && symbol do? Take for example &&C"t&i. When &i=1, &&Cact&i is first resolved to &Cactl and is then resolved to Treat.

The generated code is:

data tempaova;
  do Treat = 1 to 2;
  do Gender = 1 to 2;
    input y @@;
    output; end;
  end;
  cards;
  2 3 1 4
  4 6 4 3
  proc print;
end;
end;
cards;

The resulting output is:

\begin{tabular}{|c|c|c|c|}
\hline
OBS & TREAT & GENDER & I Y \\
\hline
1 & 1 & 1 & 1 2 \\
2 & 1 & 1 & 2 3 \\
3 & 1 & 2 & 1 2 \\
4 & 2 & 2 & 2 4 \\
5 & 2 & 1 & 1 4 \\
6 & 2 & 1 & 2 6 \\
7 & 2 & 2 & 1 4 \\
8 & 2 & 2 & 2 3 \\
\hline
\end{tabular}

- Storing and Calling Macros

For any of the aforementioned operating systems, a macro may be stored in a file and included as needed in the SAS code. A special macro, which you may have used under other circumstances, that brings a file into the SAS code is %include. Version 5 usage is

\%include file-reference;

where the file-reference refers to the reference name assigned to the file containing the macro code. If the file is stored as a member of a partitioned file (as one might do with the OS operating system), the usage may be

\%include file-reference(membername);

Version 6 usage may be as above or the actual file name may be used:

\%include 'filename';

The autocall facility is a convenient way to use stored macros. The OS operating system method will be illustrated here. Check your basics guide for the method pertaining to your operating system.

Macros should be stored in a partitioned file. The member name should coincide with the name of the macro stored under that member name. For batch use, add these lines to your JCL:

```
// EXEC SAS,OPTIONS='MAUTOSOURCE'
//SASLIB DD DSN=partitioned_file,DISP=SHR
```

For TSO under Version 5, invoke SAS with this command

SAS AUTOS(partitioned_file) + OPTIONS(MAUTOSOURCE)

Under Version 6, invoke

SAS SASAUTOS(partitioned_file) + OPTIONS(MAUTOSOURCE)

The stored macro is then used by simply placing it in the code. When it is first encountered, it will be compiled and executed. It is then available for further use in the remainder of the program.

- Procedure-Like Macros

Macros may be written and stored for later use. These macros may have been designed for small applications, but it is possible to design them with a considerable dose of flexibility for potentially large applications. These procedure-like macros can make your customized application programs available to other users without their understanding your code. They need only know how to access the macro and how to pass data and parameter values to it.

In the appendix is an extension of the ANOVA macro we saw earlier. It makes use of many of the statements and structures introduced in this tutorial, but, obviously, it makes use of more advanced aspects of the macro facility as well. You might take it as a challenge to learn about the statements, structures, and functions that appear in the macro.

Submitting the command

\%anova(infile=DATIN,Treat 2 Gender 2);

where DATIN is the file reference name for the file containing the data to be analyzed, will yield the following SAS code:
The anova macro has the flexibility of allowing any number of factors, any number of levels for a factor, any number of observations for a cross-classification, and external or internal data. Data is entered in a natural way, cell by cell. The observations within a cell are delimited by a "."

• Summary

We have seen some of the basic uses of the SAS macro facility. In addition, we have seen a more complicated macro that can be added to a library of procedure-like macros. To learn more about how you might take advantage of this facility, consult your SAS User's Guide: Basics, Version 5 Edition (SAS Institute, 1985) or SAS Guide to Macro Processing, Version 5 Edition (SAS Institute, 1987), or SAS Guide to Macro Processing, Version 6 Edition (SAS Institute, 1987).

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