Get Started Writing SAS® Macros
Luisa Hartman, Jane Liao, Merck Sharp & Dohme Corp.

ABSTRACT

The SAS Macro Facility is a tool which lends flexibility to your SAS code and promotes easier maintenance. It is composed of macro variables and macro statements. When incorporated into your SAS code, macro variables and macros facilitate text substitution, reusability, customization, and standardization. This paper will provide basic examples on creating and using macro variables and macros. It presents the why, when and how one should use macros demonstrating macro code versus the same in open SAS code.

INTRODUCTION

It can be overwhelming to think of writing a macro. One can take small steps to build up confidence to write one. First, start by using macro variables in open SAS code. Then, once you are comfortable using macro variables, try building a simple macro. The simple macro can be first written as a program in standard SAS code. Then, %MACRO and %MEND statements can be added to make programs reusable. Finally, add macro parameters one at a time.

The following topics will be discussed:

- The Macro Facility
- Definition of macro variables (&)
  - Three ways to create macro variables
  - Scope or Referencing Environments
  - Other considerations
- Macros (%)
  - Macro Syntax
  - Passing Information into a Macro Using Parameters
  - Conditionally Generating SAS Code
  - Do Loop in Macro
  - Inserting Comments in Macro
  - How to use a Macro

THE MACRO FACILITY

The macro facility, which is part of base SAS software, is a tool for facilitating the use of text in a SAS program. It allows you to assign a name to character strings or groups of SAS programming statements. The macro facility is composed of two components: macro variables which are preceded by an ‘&’ and macro statements which are named with the starting character, '%'.

Let’s take a detailed look at macro variables beginning with a definition, and then three ways to create a macro variable, %LET, CALL SYMPUT, and Proc SQL Select INTO.

MACRO VARIABLES: Definition

As previously stated, macro variables provide an efficient way to represent text strings in SAS code. They are part of the SAS macro language. Another name is a symbolic variable. Macro variables are different than data step variables. The main differences are summarized in Table 1 for ease of comparison:

<table>
<thead>
<tr>
<th>Table 1: Data Step Variables</th>
<th>Macro Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associated with Data Step</td>
<td>Can be defined and used anywhere in SAS program</td>
</tr>
<tr>
<td>Associated with SAS dataset</td>
<td>Independent of SAS dataset</td>
</tr>
<tr>
<td>Value depends on current observation</td>
<td>One value until explicitly changed</td>
</tr>
<tr>
<td>Name is column or variable name</td>
<td>Name starts with ‘&amp;’</td>
</tr>
<tr>
<td>Store numeric or text information</td>
<td>Always stored as text</td>
</tr>
<tr>
<td>Stored as a column in a SAS dataset</td>
<td>Stored in a macro symbol table during a SAS session.</td>
</tr>
</tbody>
</table>

There are two types of macro variables: system or automatic and user-defined. System macro variables like SYSDATE, SYSUSERID, and SYSTIME are “automatically” available when your SAS session begins whereas user-defined are established by your code.
It is also important to understand the scope or referencing environments of a macro variable. A macro variable defined within a macro is by default a local macro variable. It is only available during macro execution. A macro variable defined outside of a macro is a global macro variable. To create a global macro variable within a macro, the `%GLOBAL` statement can be used. Likewise, the `%LOCAL` statement is used to create a macro variable of specific scope.

**MACRO VARIABLES: Ways to Create a macro variable**

What if we wanted to create a user-defined macro variable? How would we do that? The first way is the `%LET` statement which assigns a value directly to a macro variable. In this case, we know the value ahead of time. The second method, CALL SYMPUT, assigns a value produced in the DATA step to a macro variable. The value is based on the dataset. Thirdly, PROC SQL with the SELECT INTO clause assigns a value produced during PROC SQL processing to a macro variable, especially summary statistics. The value, as with CALL SYMPUT, is based on the dataset. Next let's examine each method in detail.

**METHOD 1: %LET statement**

The easiest way to create a macro variable is to use the macro language statement, `%LET`. The macro language syntax follows:

```sas
%LET macro-variable-name=<value>;
```

The `macro-variable-name` follows the rules for SAS names. In general, the maximum length of a macro variable name is 32 characters, it must begin with a letter or underscore and the remainder of the name can only be letters, numbers, and underscores [1]. The value is any string or macro expression. The value can represent text to be generated or to be used by the macro processor. Table 2 gives some examples of user defined macro variables and what each `%LET` statement establishes:

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Let mv1=Total; Assigns Total to macro variable mv1</td>
</tr>
<tr>
<td>%Let mv2=&quot;Total&quot;; Assigns “Total” to macro variable mv2</td>
</tr>
<tr>
<td>%Let a=5; Assigns 5 as string to macro variable a</td>
</tr>
<tr>
<td>%Let b=7-2; Assigns string 7-2 to macro variable b</td>
</tr>
</tbody>
</table>

Notice in the second example that the quotes are included in the value of `mv2`. Additionally, the last example illustrates that a calculation is not performed. The macro variable, `b` is equal to the string, 7-2.

The Example 1 code below shows an application of a user defined macro variable, `DEBUG`, compared to code without the use of a macro variable. When developing programs, it is useful to set up code such that you can turn the execution of Proc Contents and Proc Freq on or off. However, once you know your data, it is not necessary to execute them. The macro variable, `DEBUG`, provides that mechanism. We only need to fill in a blank or an "*" into the value for the macro variable, `DEBUG` once. When `DEBUG = *`, each statement will be treated as comments, and the code will not be executed. We can modify 9 lines of code with one `%LET` statement.

**Example 1 Open SAS Code**

```sas
proc contents data=datadir.demog;
title1 "ABxxxx Protocol nnn";
title2 "contents of demog";
run;

proc freq data=datadir.demog;
tables gender race;
title1 "ABxxxx Protocol nnn";
title2 "counts of gender and race";
run;
```

**Example 1 Using Macro Variable**

```sas
%let DEBUG = *;
&DEBUG proc contents data=datadir.demog;
&DEBUG title1 "ABxxxx Protocol nnn";
&DEBUG title2 "contents of demog";
&DEBUG run;

&DEBUG proc freq data=datadir.demog;
&DEBUG tables gender race;
&DEBUG title1 "ABxxxx Protocol nnn";
&DEBUG title2 "counts of gender and race";
&DEBUG run;
```
A simple and practical use of a macro variable is to substitute a dataset name. In Example 2, four coding changes would need to be made for the file name but instead we can just change the value of the macro variable, DSN.

**Example 2 Open SAS Code**

```sas
proc contents data=datadir.demog;
   title1 'ABxxxx Protocol nnn';
   title2 'contents of demog';
run;

proc freq data=datadir.demog;
   tables gender race;
   title1 'ABxxxx Protocol nnn';
   title2 'demog counts of gender and race';
run;
```

**Example 2 Using Macro Variables**

```sas
%let DEBUG = ;  %let dsn=demog;
&DEBUG proc contents data=datadir.&dsn;
&DEBUG title1 "ABxxxx Protocol nnn";
&DEBUG title2 "contents of &dsn"
&DEBUG run;

&DEBUG proc freq data=datadir.&dsn;
&DEBUG tables gender race;
&DEBUG title1 "ABxxxx Protocol nnn";
&DEBUG title2 "&dsn counts of gender and race";
&DEBUG run;
```

Notice the single quote (in the box on the left) is changed to a double quote (in the box on the right) in order for SAS to resolve the macro variable in the Title statement.

**METHOD 2: CALL SYMPUT**

Now we'll look at the second method for creating macro variables...CALL SYMPUT. Here the value is assigned to the macro variable in the data step and is based on the data. The syntax of the CALL SYMPUT statement is:

```
CALL SYMPUT ('macro_varname', value);
```

CALL SYMPUT has two arguments either of which may be a variable, a constant or combination of the two. It is a DATA step call routine, not a macro language statement. Values are assigned to the macro variable through SYMPUT during DATA step execution.

In Example 3, the macro variable, total is created, and is assigned the total number of observations in the 'demog' dataset. A subset of the 'demog' dataset can be found in Table 3.

**Example 3**

```sas
data _null_;  set datadir.demog end=eof;
   if eof then  call symput('total',_n_);
run;
```

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Sample Dataset 'demog' Used for Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>SITEID</td>
<td>SUBJID</td>
</tr>
<tr>
<td>001</td>
<td>000005</td>
</tr>
<tr>
<td>001</td>
<td>000010</td>
</tr>
<tr>
<td>001</td>
<td>000015</td>
</tr>
</tbody>
</table>

It's important to remember that you cannot use a macro variable until the DATA step in which SYMPUT assigns the value has completed. The macro variable's value will not be available until after completion. The reason is that SAS does not see the data to assign the value of the macro variable until the final, program execution stage.

Example 4 shows an application of CALL SYMPUT. Using the 'demog' dataset, we want our SAS code to determine (1) How many treatment groups the study has? And (2) What is the decoding text for each TREATCD?
Example 4

```
proc sort data=datadir.demog out=sorteddemog;
  by treatcd;
run;

data _null_;  
  set sorteddemog end=eof;
  by treatcd;
  retain trtcnt 0;
  if first.treatcd then do;
    trtcnt=trtcnt+1;
    call symput('trt'||strip(put(trtcnt,3.)), treatment);
  end;
  if eof then do;
    call symput('trt_num', strip(put(trtcnt,3.)));  
  end;
run;
%mput trt_num = &trt_num;
%mput trt1 = &trt1;
%mput trt2 = &trt2;
```

First, sort the data by TREATCD. Then on each value change in the variable TREATCD, add 1 to a counter (TRTCNT) and CALL SYMPUT creating a macro variable 'TRT' concatenated with the value of TRTCNT. The three %PUT statements display the macro variables and their values in the SAS LOG window. The values for the 3 macro variables created with our CALL SYMPUT code are included in the output for Example 4 below:

SAS LOG: Output from Example 4

```
%put trt_num = &trt_num;
trt_num = 2
%put trt1 = &trt1;
trt1 = Placebo
%put trt2 = &trt2;
trt2 = AB 50 mg
```

You can also use %PUT _user_; to view all the user-defined macro variables in your current SAS session in the SAS LOG. Additionally, you can use %PUT _all_; to view all macro variables, both system and user-defined in your current SAS session in the SAS LOG.

**METHOD 3: PROC SQL SELECT INTO**

PROC SQL is another powerful and efficient way to create macro variables by taking advantage of the INTO clause. Each macro variable must be preceded with a ‘:’. With the INTO clause of the SELECT statement, you can establish a macro variable’s value by assigning it the result of a calculation or the value of a data column or variable. The INTO clause can create a macro variable with all values of a column by using the SEPARATED BY clause. If the macro variable was not previously defined, the INTO clause creates it.

```
PROC SQL;
  SELECT column-1 <, column-2....>
  INTO :macro-variable name <, :macro-variable name, >
  FROM dataset_name
    <WHERE expression>
    <SEPARATED BY "",">
    <HAVING expression>
    <ORDER BY column-1 <, column-2... <DESC> >;
QUIT;
```

In Example 5, the macro variable, total is created, and is assigned the total number of observations in the 'demog' dataset just as we did with CALL SYMPUT.
Using our ‘demog’ dataset, we want our SAS code to print out the oldest patient(s) in the study. In Example 6, we are using the summary statistic, MAX and creating the macro variable, max_age. The macro variable, max_age, will be written to the SAS LOG by the %PUT statement. We will also print those observations where the patient age is equal to the value of max_age. The SAS LOG and OUTPUT windows show the results. Two patients have a maximum age of 68.

Example 6

```sas
proc sql;
    select max(age) into :max_age
    from datadir.demog;
quit;
%put max_age = &max_age;
proc print data=datadir.demog;
    where age eq &max_age;
    title "Oldest Patients in the Study";
run;
```

SAS LOG: Output from Example 6

```
%put max_age = &max_age;
max_age = 68
```

NOTE: There were 2 observations read from the data set DATADIR.DEMOG. WHERE age=68;

SAS OUTPUT: Output from Example 6

```
Oldest Patients in the Study
        SITEID SUBJID TREATCD TREATMENT AGE AGE_GRP RACE GENDER HT_CM WT_KLG
004 000383 1 AB 50 mg 68 4 white M 175 74.5
005 000094 0 Placebo 68 4 white M 166 69.8
```

SCOPE OR REFERENCING ENVIRONMENT

As previously mentioned, a macro variable defined within a macro is by default a local macro variable only available during macro execution. In the first case for Example 7, since TSTMVAR is defined inside the macro, TEST, it is a local macro variable. However, in the second case for Example 7, the %PUT issued after the macro ends produces a WARNING because the local macro variable is no longer available.

Example 7 Case 1

```sas
%macro test;
    %let tstmvar=15;
    %put tstmvar = &tstmvar;
%mend test;
%test;
Log:
tstmvar = 15
```

Example 7 Case 2

```sas
%macro test;
    %let tstmvar=15;
%mend test;
%test;
%put tstmvar = &tstmvar;
Log:
WARNING: Apparent symbolic reference TSTMVAR not resolved.
```

A global macro variable is available during your entire SAS session. In the first case for Example 8, since TSTMV is defined outside the macro, TEST, it is a global macro variable. SAS uses the global macro variable within the macro, TEST, since it is available. The value is changed to 15.
In the second case for Example 8, the %GLOBAL statement issued inside the macro produces a global macro variable, TSTMV which is available for the entire SAS session.

In Example 9, it is easy to confuse a macro variable's scope if we are not careful in defining it. In the first case, since TSTMVAR is defined outside the macro, TEST, initially, it is a global macro variable. SAS uses the global macro variable within the macro, TEST, since it is available. The value is changed to 15.

However, in the second case, TSTMVAR is BOTH a global macro variable and a local macro variable due to the %local definition. A macro variable defined outside of a macro is a global macro variable. This coding practice would not be recommended. Use a different macro variable name to avoid confusion.

**OTHER CONSIDERATIONS**

In addition to the direct macro variable references (&name) discussed so far, our code can also benefit from indirect macro variable references. In the macro in Example 14, there are user-defined macro variables which are referenced indirectly: %LET VAR1=age; , %let VAR2=ht_cm; , and %LET VAR3=wt_klg; . Let's examine the use of VAR1, VAR2, and VAR3. Assume &i is equal to 1. To reference them in a %PUT, which of the following two statements is correct?

```
Statement 1: %put &VAR&i;        /* incorrect - tries to resolve &VAR   */
Statement 2: %put &&VAR&i;       /* correct – resolves to VAR1   */
```

When the macro processor receives the %PUT &VAR&i; statement (statement 1), it produces a warning message saying that there is no macro variable VAR because the macro facility has tried to resolve &VAR and then &i and concatenate those values [1].

When the macro processor receives the %PUT &&VAR&i; statement (statement 2), it is processed as follows:
1. The double ampersand (&&) resolves to a single ampersand (&).
2. VAR is evaluated as text.
3. &i is resolved to 1.
4. Now the reference is &VAR1 so the macro processor starts from the beginning again, and prints the value of VAR1 in the SAS LOG. The value of VAR1 is age.
With a double ampersand, the macro processor scans the reference twice, and is able to resolve the macro variable reference. The double ampersand is needed to force the macro processor to do an additional scan.

Another consideration is the use of prefixes and suffixes with macro variables. Within your code, you may want to generate macro variables based on a certain condition and name them accordingly. For example, if macro variables indicate the sales for each month of the year, a numeric suffix can be appended.

In the following reference, `%let name = sales;`, the rules for usage of prefixes and suffixes are:
1. Use a period (.) as the delimiter between the prefix and &name. In the example, save.&name, the macro variable resolves to `save.sales`.
2. Use a period (.) as the delimiter between &name and the suffix. In the example, &name.1, the macro variable resolves to `sales1`.
3. Use two periods (..) to produce a period in the text. In the example, &name..1, the macro variable resolves to `sales.1`.

**MACROS: MACRO SYNTAX**

The syntax for a macro is simple. It starts with the key word `%MACRO` followed with a macro name; parameters are optional. If there are multiple parameters, separate them by a comma. A macro ends with the key word `%MEND`. The macro name is optional at the `%MEND` statement, but it is strongly recommended for debugging purposes. Refer to Table 4 for macro syntax.

<table>
<thead>
<tr>
<th>Table 4: Macro Syntax</th>
</tr>
</thead>
</table>
| `%MACRO` macro-name `<(macro-parameters)>;  
  
  macro text  
  
  `%MEND` `<macro-name>`;  |

**Invoking the macro:** `%macro-name `<(macro-parameters)>`;

**Note:** Reserved words should not be used as a macro name. For example, if “RUN” is used as a SAS macro name, the following error messages will appear on SAS LOG window:
- ERROR: Macro RUN has been given a reserved name.
- ERROR: A dummy macro will be compiled

Below is an example of starting to write a macro. On the left side, it shows the open SAS code with the use of a macro variable. To write a macro, first add the key word, `%MACRO` and give the name of the macro as `pat_list`; then put the key word `%MEND` to end the macro. Don’t forget to include the macro name, `pat_list` with the `%MEND` statement.

<table>
<thead>
<tr>
<th>Open SAS code for example 10</th>
<th>Macro approach: Example 10</th>
</tr>
</thead>
</table>
| **proc sql** noprint;  
  select mean(age) into :avg  
  from demog;  
  quit;  
  proc print data=demog noobs;  
  where age > &avg;  
  title “Patients with age > Average &avg”;  
  run;  
| **%MACRO** pat_list;  
  `%* begin macro define.;  
  proc sql noprint;  
    select mean(age) into :AVG  
    from demog;  
  quit;  
  proc print data=demog noobs;  
    where age > &AVG;  
    title “Patients with age > Average &AVG”;  
  run;  
  `%MEND` pat_list;  
  `%* end macro define.;  
  %pat_list;  
  `%* macro call.;  |

**PASSING INFORMATION INTO A MACRO USING PARAMETERS**

There is not much benefit to writing a macro when there is no macro parameter. To make a macro reusable, flexible and powerful, macro parameters are a must. From our first example, we can add a macro parameter to print a listing of patients with AGE greater than average age; HEIGHT greater than average height; WEIGHT greater than average...
weight etc. Refer to Example 11. Since we have the macro with the parameter MVAR, we can use macro calls to generate patient listings for HEIGHT and WEIGHT while for open SAS, redundant SAS statements were needed.

Open SAS code for Example 11

```sas
*** age ***;
proc sql noprint;
   select mean(age) into :avg
        from demog;
quit;
proc print data=demog noobs;
   where age > &avg;
   title "Patients with age > average &avg";
run;

*** height ***;
proc sql noprint;
   select mean(HT_CM) into :avg
        from demog;
quit;
proc print data=demog noobs;
   where ht_cm > &avg;
   title "Patients with height > average &avg";
run;

*** weight ***;
proc sql noprint;
   select mean(WT_KLG) into :avg
        from demog;
quit;
proc print data=demog noobs;
   where wt_klg > &avg;
   title "Patients with weight > average &avg";
run;
```

We can make a macro more useful by adding more macro parameters. In Example 12, we can add another macro parameter MBY to have the macro print patient listings with a by group.

**Example 12 Adding more macro parameters**

```sas
%MACRO pat_list (MVAR=age, MBY=gender);
   proc sql noprint;
      select mean(&MVAR) into :AVG
           from demog;
   quit;
   proc sort data=demog out=sorted;
      by &MBY;
   run;
   proc print data=sorted noobs;
      by &MBY;
      where &MVAR > &AVG;
      title "Patients with &MVAR > Average &AVG by &MBY";
   run;
%MEND pat_list;
```

Macro calls to generate listing with the by group.

```sas
%pat_list;
%pat_list(MVAR=age, MBY=RACE);
%pat_list(MVAR=age, MBY=TREATMENT);
```
CONDITIONALLY GENERATING SAS CODE

One of the advantages of Macros is their flexibility. From our previous two macro examples (11 & 12), we see that we can print different listings: a whole listing of patients and a listing of patients by groups. But we don't have the flexibility to execute the statement depending on the value of the macro parameter that is conditionally generating SAS code. In the example below, we will be able to process the SAS code depending on the macro parameter MBY. If &MBY is given a value, the macro will sort the data then print the list grouped by the value assigned by &MBY. Otherwise, if &MBY is NULL then we don't need to sort the data just print the list as a whole without grouping.

Example 13 Conditionally Generating SAS Code

```
%MACRO pat_list(MVAR=age, MBY=gender);
  proc sql noprint;
    select mean(&MVAR) into :AVG
    from demog;
  quit;
  %if "&MBY" ne "" %then %do;
    proc sort data=demog out=sorted;
    by &MBY;
    run;
    proc print data=sorted noobs;
      by &MBY;
      where &MVAR > &AVG;
      title "Patients with &MVAR > Average &AVG by &MBY";
    run;
  %end;
  %else %do;
    proc print data=demog noobs;
      where &MVAR > &AVG;
      title "Patients with &MVAR > Average &AVG";
    run;
  %end;
%MEND pat_list;
```

DO LOOP IN MACRO

A macro can be more complex and powerful when using a Do Loop. Refer to Table 5 for syntax using a Do Loop in a macro.

<table>
<thead>
<tr>
<th>Table 5: Do Loop in Macro Syntax</th>
</tr>
</thead>
</table>
| %DO macro-variable=start %TO stop <BY increment>;
| text and macro language statements|
| %END;                            |
| %DO %UNTIL (expression);
| text and macro language statements|
| %END;                            |
| %DO %WHILE (expression);
| text and macro program statements|
| %END;                            |

In Example 14 below, the left side is the listing of 12 macro calls. The right side is the approach using a Do Loop. We define a new macro called loop_list. For some complex tasks such as a Do Loop, we have to define a macro. For our new macro, we define some new macro variables, &var1, &var2 and &var3. With these variables, we are going to get the average; the macro variables &byvar1 through &byvar4 are the variables for the by group to print patient listings. When the byvar is NULL, we will have our patient listing as a whole. When the byvar is Gender, we will print the patient listing by gender and so on. The outside loop i is from 1 to 3 for the 3 average variables. For each of the average variables, we can have the inside loop j from 1 to 4 to print 4 listings, that is, whole listing, by gender, by race and by treatment.
When calling our macro `%pat_list`, we are using the indirect way to reference our macro variables &MVAR and &MBY. For MVAR, at the start of the i loop, we have resolved &&var&i to &var1 then, we resolve &var1 as age.

<table>
<thead>
<tr>
<th>Macro calls for Example 14</th>
<th>Example 14 Using DO LOOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Average age</td>
<td>%MACRO loop_list;</td>
</tr>
<tr>
<td>%pat_list(mby=);</td>
<td>%let var1=age;</td>
</tr>
<tr>
<td>%pat_list;</td>
<td>%let var2=ht_cm;</td>
</tr>
<tr>
<td>%pat_list(mby=race);</td>
<td>%let var3=wt_klg;</td>
</tr>
<tr>
<td>%pat_list(mby=treatment);</td>
<td>%let byvar1=;</td>
</tr>
<tr>
<td></td>
<td>%let byvar2=gender;</td>
</tr>
<tr>
<td></td>
<td>%let byvar3=race;</td>
</tr>
<tr>
<td></td>
<td>%let byvar4=treatment;</td>
</tr>
<tr>
<td>2) Average height</td>
<td>%do i=1 %to 3;</td>
</tr>
<tr>
<td>%pat_list(mvar=HT_CM, mby=);</td>
<td>%do j=1 %to 4;</td>
</tr>
<tr>
<td>%pat_list(mvar=HT_CM);</td>
<td>%pat_list(mvar=&amp;&amp;var&amp;i, mby=&amp;&amp;byvar&amp;j);</td>
</tr>
<tr>
<td>%pat_list(mvar=HT_CM, mby=race);</td>
<td>%end;</td>
</tr>
<tr>
<td>%pat_list(mvar=HT_CM, mby=treatment);</td>
<td>%end;</td>
</tr>
<tr>
<td>3) Average weight</td>
<td>%MEND loop_list;</td>
</tr>
<tr>
<td>%pat_list(mvar=WT_KLG, mby=);</td>
<td>%loop_list;</td>
</tr>
<tr>
<td>%pat_list(mvar=WT_KLG);</td>
<td></td>
</tr>
<tr>
<td>%pat_list(mvar=WT_KLG, mby=race);</td>
<td></td>
</tr>
<tr>
<td>%pat_list(mvar=WT_KLG, mby=treatment);</td>
<td></td>
</tr>
</tbody>
</table>

**INSERTING COMMENTS IN MACRO**

All code benefits from commenting, and macro code is no exception. There are two forms of comments for macro code (please see Table 6). There is no restriction on the form of commenting to use, but there are some recommendations. Refer to Example 15.

<table>
<thead>
<tr>
<th>Table 6: Inserting Comments in Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro comment:</td>
</tr>
<tr>
<td>▪ First form: beginning with /* and ending with */</td>
</tr>
<tr>
<td>□ Recommended when explaining the definition of the macro parameters.</td>
</tr>
<tr>
<td>□ Not recommended in the macro since it can be hard for debugging due to the inconvenience when you try to comment out code you don’t want to run.</td>
</tr>
<tr>
<td>▪ Second form: begins with a %* and ends with a ;</td>
</tr>
<tr>
<td>□ Recommended and should be the first choice when inserting a comment in a macro.</td>
</tr>
<tr>
<td>▪ The comment will not be seen in the log when macro is executed.</td>
</tr>
<tr>
<td>▪ The asterisk-style comment ( * commentary ; ) used in SAS code is not recommended within a macro.</td>
</tr>
</tbody>
</table>
HOW TO USE A MACRO

So far we have talked about defining a macro. But what about how to use a macro? In the examples that we had discussed in the paper, we first define a macro then we invoke the macro by macro calls within the same program (file). If we want, our macro can be used repeatedly whenever we need to, and it can be shared by other programs and other programmers. We can achieve that by saving the macro to a permanent SAS program file then make it available either by using a %include statement or OPTIONS SASAUTOS.

• %INCLUDE
  1. Save macro as a SAS program: macro_file_name.sas
  2. Reuse the macro by including the SAS file
  3. Syntax: %INCLUDE "path\macro_file_name.sas"
  4. A very important note: When you make your macro call (invoke the macro), you need to use the macro name, not the SAS file name you created when you saved your macro. In order not to be confused, it is better to save the macro to a SAS file with the same name as your macro name.

• OPTIONS SASAUTOS
  1. While an include statement can only include one file at a time, the SASAUTOS makes all macros available from the folder.
  2. Syntax: %let search1= c:\search1\macrolib;
     %let search2= c:\search2;
     Options sasautos="(&search1", "&search2", sasautos);
  3. Important notes:
     a. The macro name and the SAS file name have to be the same.
     b. When multiple paths are defined in the SASAUTOS, i.e. Options sasautos="(&search1", "&search2", sasautos); the order of the searching will be &search1 and search2 and sasautos. i.e. if we have saved the same macro name at search1 and search2, the macro name at search1 will be compiled.
     c. If we update a macro in the middle of the SAS session, we will need to compile the updated macro first or restart the SAS session in order to invoke the change.
CONCLUSION

This paper presented ways to create a macro variable, define a macro, and when and how to use a macro variable and a macro. With proper application, we learned that there are many benefits to using macros. We encourage you to begin incorporating macro variables into your SAS code and try writing macros if you have not done so already.

REFERENCES


TRADEMARK INFORMATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

ACKNOWLEDGMENTS

We would like to thank Amy Gillespie, Mary Anne Rutkowski, Lei Sun, and Shuping Zhang from Merck Sharp & Dohme Corp. for reviewing the paper and providing valuable feedback.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Luisa Hartman
Company: Merck Sharp & Dohme Corp.
Address: 770 Sumneytown Pike, P.O. Box 4
City, State ZIP: West Point, PA 19486
Work Phone: 215-993-1567
Fax:
Email: luisa_hartman@merck.com

Author Name: Jane Liao
Company: Merck Sharp & Dohme Corp.
Address: 351 North Sumneytown Pike
City, State ZIP: North Wales, PA 19454-1099
Work Phone: 267-305-7514
Fax:
Email: jane_liu2@merck.com