

SAS™ Macro Basics and Fundamentals

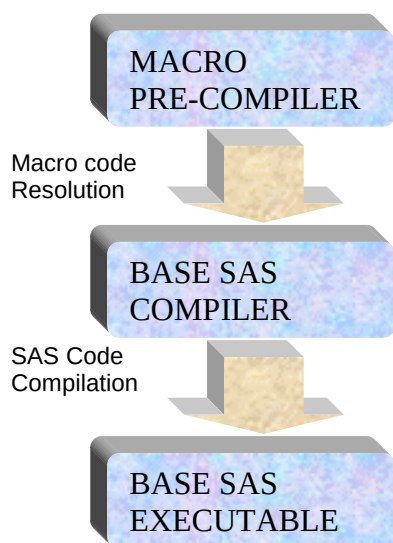
Timothy J Harrington, Trilogy Consulting Corporation, Waukegan, IL

Abstract

This paper introduces and discusses the SAS MACRO language for new SAS Programmers, or those with some prior SAS MACRO experience. This paper also explains the role of SAS Macro code in the SAS System compilation and execution phases and how SAS MACRO and the Base SAS System may be used together to write more efficient code and tackle specific programming tasks. The major SAS MACRO topics and common functions are illustrated, with examples, as well as common problems encountered, and their solutions, when using the SAS MACRO language.

The role of SAS MACRO

The SAS Macro compiler is a pre-compiler, which, with only a few exceptions, resolves SAS Macro code before the Base SAS (or SAS SCL) code is compiled. The basic sequence of events is thus:



The SAS MACRO language has two types of code (1) Invocation Statements, preceded by a percent (%) sign and (2) Macro Variables, preceded by an ampersand (&). The % and & symbols define whatever follows them as Macro code and the Macro preprocessor is invoked accordingly. The resolved contents of the Macro code are then handled by the SAS (or SCL) compiler.

Why Use SAS Macro

The SAS Macro language has many uses, the most significant being a convenient way of 'modularizing'

code into portable and reusable program units. Such program units can be stored in Macro libraries. Macro variables allow substitution of SAS variable names and even Base SAS (or SCL) code in a SAS program and such items may be passed as parameters between program modules. Macro code also allows the use of conditional compilation and execution of the underlying SAS code

The Macro Block

A SAS Macro block is a section of code (often called a 'Macro') delimited by a %MACRO and a %MEND statement like this:

%Macro My_Macro(Parameters – if any);

<SAS Code (and other MACRO Statements)>

%Mend My_Macro;

The code within such a %MACRO - %MEND block is called, compiled, and executed by specifying the Macro name:

%My_Macro (parameter list);

Macro blocks may pass Macro variables as parameters and may be nested. Below is an example, in which the requirement is to print lists of patients in a Clinical Trials data set like this one by Treatment Group (TG) showing different patient data items:

TG	PATIENT	DATEBRTH	AGE	SEX	RACE	HT	WT
01	1001	09/12/69	29	M	White	71	188
01	1002	11/23/76	22	F	White	66	164
01	1003	01/11/72	27	F	Black	67	162
01	1004	12/11/74	24	M	White	72	203
02	1101	08/28/76	22	M	Black	73	215
02	1102	07/22/73	25	F	White	65	144
02	1103	06/29/79	19	M	Asian	70	190
02	1104	01/12/80	19	M	White	69	184

The data set is called CLINTEST and the Macro shown overleaf, called PRINTPAT, is used to achieve this. The parameters DOSEGRP and ITEMS are used to substitute the appropriate SAS code within the Macro block. The value of TG is assigned to DOSEGRP, and the items required for printing, used in the PROC PRINT VAR statement, are assigned to ITEMS. ITEMS has a default value of 'ALL', the %IF statement tests for this and when true

the VAR statement is omitted from the PROC PRINT, thereby allowing all the variables to be listed. In this example the Patient number, Date of Birth, Age, and Sex are listed.

```
%Macro printpat(dosegrp=, items=ALL);
  proc print data = clintest
    (where=(trtgrp=&dosegrp));
  %if &items ne ALL %then %do;
    var patient &items;
  %end;
  run;
%mend printpat;

%printpat(dosegrp=01, items=DATEBRTH
AGE SEX);
```

Macro Variables

A Macro variable name, like a Base SAS variable can be up to eight characters long (32 in version 7) and may contain numeric digits, underscores, and hyphens. A Macro variable is assigned a value with the %LET statement, for example:

```
%LET MACVAR1 = VA21;
```

```
%LET MLOOP = 1;
```

```
%LET DRUGDATA = Clinical Trial Data
Set 2;
```

```
%LET P_VALUE = ;
```

Anything which is to the right of the equals sign (=) and precedes the next semicolon is stored in the Macro variable. In the above examples the value 'VA21' is stored in MACVAR1, the value '1' in MLOOP, and the character string (including any embedded spaces) 'Clinical Trial Dataset 2' is stored in DRUGDATA. In the fourth example the Macro variable P_VALUE is defined but left empty, the equivalent of missing. If a previously undefined Macro variable is referenced a WARNING is written to the log file to that effect. Some further points to note about Macro variables are (1) Macro variables can be blank or can contain any number of alpha, numeric, or other characters (2) Leading or trailing spaces are not stored using a %LET statement, but embedded spaces are (3) SAS Macro regards all Macro variables as unquoted character strings.

To access the contents of a Macro variable the variable name must be 'resolved' using the ampersand (&). To display the contents of a Macro variable in the log file %PUT is used. %PUT may be used with other (unquoted) text, for example:

```
%PUT &MACVAR1;
%PUT The values of MLOOP and
DRUGDATA are &MLOOP and &DRUGDATA;
```

These statements will result in the following being written to the SASLOG file:

VA21

**The values of MLOOP and DRUGDATA
are 1 and Clinical Trial Data Set 2**

If the & symbols were omitted the Macro variables would not be resolved and this output would be:

MACVAR1

**The values of MLOOP and DRUGDATA
are MLOOP and DRUGDATA**

The & symbol tells the Macro compiler where to begin resolving, normally resolution stops at the space or other delimiter such as a semicolon or comma, but a problem arises when a Macro variable is embedded within a character string. For example, if the intention is to resolve two data set names JANSALES and FEBSALES and the first three characters of the data set name are stored in the Macro variable MONTH, if MONTH contains the value JAN, using &MONTHSALES or &MONTH SALES will result in an error. Clearly, a terminator is needed to indicate which part of 'MONTHSALES' is the Macro variable and which part is regular SAS. This terminator is the period (.), hence the correct coding is &MONTH.SALES.

Another issue when resolving a Macro variable is when quotation marks are needed, for example when assigning a character SAS variable with the contents of a Macro variable. For example, if the Macro variable CUSTNAME contains the (unquoted) character string 'JAMES SMITH' and the intention is to store that value in the SAS character variable PURCHSR at runtime, this would result in an error:

```
PURCHSR = &CUSTNAME;
```

The resolved value of the Macro variable is a character string literal, which must be in quotation marks, but if single quotation marks are used:

```
PURCHSR = '&CUSTNAME';
```

the value of PURCHSR is '&CUSTNAME' because the Macro pre-compiler does not recognize single quotation marks. For resolution to take place double quotation marks must be used:

```
PURCHSR = "&CUSTNAME";
```

this resolves to:

```
PURCHSR = "James Smith";
```

Conversely, if an ampersand is being used in a literal string single quotation marks must be used, if double quotation marks are used the Macro pre-compiler will try to resolve the text. For example, 'Smith & Jones' is handled as is, the ampersand is a literal character, but "Smith & Jones" will result in an error because the Macro pre-compiler will be activated by the ampersand. The same is true for the percent sign (%).

There are also certain *quoting functions* available in SAS Macro, which are discussed later.

Macro Parameters, Local, and Global Macro Variables

Macro variables may be passed as parameters to a %MACRO %MEND block. Such parameters are placed in parenthesis following the %MACRO statement. The example below is a Macro named COUNTOBS, which returns the numbers of observations in the data set specified by the first parameter INDS containing the value of PATIENT where that patient number is included in the list in the second parameter PATIENTS. The third parameter contains the name of the output data set, which, for each patient will have the name of the Input data set, the Patient Number, and the number of observations found for that patient.

```
%Macro countobs(inds=,patients=,
  outds= patcount);
proc sql;
  create table &outds as
    select "&inds" as tname
      length = 17, patient,
      count(*) as n
    from &inds
      where(patient in
        (&patients))
      group by patient;
quit;
%mend countobs;

%countobs(inds=DRUGLIB.CLINTEST,
  patients=2089 2090 2092,
  outds=numobs);
```

PROC SQL is used with the COUNT function to return the number of observations grouped by patient for each of the patient numbers (patient must be numeric) in the parameter PATIENTS. In this example the data set NUMOBS will have three observations, one for each patient. The value of TNAME will in all three observations be "DRUGLIB.CLINTEST", the three values of PATIENT will be 2089, 2090, and 2092, and the values of N will be the respective observation counts for each of the three patients.

To restrict the scope of Macro variables to the Macro block itself only, %LOCAL is used. For example

```
%LOCAL COUNT01 COUNT02;
```

This statement defines the Macro variables COUNT01 and COUNT02 as local to the %MACRO %MEND block they are defined in. Any reference to either of these variables outside that Macro block will be undefined or they will have the values of any previous definition of COUNT01 or COUNT02 prior to entering the local Macro block. To define Macro variables as %LOCAL to the Macro block in question is always good practice, this prevents a possible clash of Macro variables with the same name outside the block. If a parameter defined in the %MACRO statement is also defined as %LOCAL the input value to that parameter is unchanged on return, even if it is changed within the %MACRO block.

%GLOBAL defines a Macro variable as common to a whole application. %GLOBAL should be used with care, since it is not the 'opposite' of %LOCAL and Macro variables with the same names as those defined as %GLOBAL will be overwritten by the Global values (This is a portability problem). The use of %GLOBAL should be restricted to the application or whole system level and all %GLOBAL variables should be defined in one place at the start of the whole application. %GLOBAL Macro variables should, ideally, be constants that will never change such as:

```
%GLOBAL G_PI G_EXP;
%LET G_PI = 3.142;
%LET G_EXP = 2.718;
```

Macro Variable Functions

Several Macro variable functions exist which have 'sister functions' in the Base SAS language. These functions are listed below and use the Macro variables SUGI_24 and SUGI_25 as examples following the assignments below:

```
%LET SUGI_24=Miami April 11 to 14
1999;
%LET SUGI_25 = Indianapolis April 9
to 12 2000;
```

%UPCASE converts all alphabetic characters to upper case. **%LOWCASE** converts all alphabetic characters to lower case:

```
%LET BIG_SUGI = %UPCASE(&SUGI_25);
%PUT &BIG_SUGI;
```

```
INDIANAPOLIS APRIL 9 TO 12 2000
```

%LENGTH returns the length (number of characters including embedded spaces) of the contents of a Macro variable:

```
%LET X = %LENGTH(&SUGI_24);  
%LET Y = %LENGTH(&SUGI_25);  
%PUT X= &X Y= &Y;
```

X= 25 Y=31

%LEFT, **%COMPRESS**, and **%TRIM** left justifies (a Macro variable when defined with a **%LET** statement is always left justified), removes multiple blanks, and removes trailing blanks respectively.

%INDEX(<Macro variable>,<substring>) returns the location of the first character of the first occurrence of a sub-string within a Macro variable. If the sub-string is not found the result is 0. To search for a comma (,) or other 'problem' characters in a Macro string quoting functions must be used. In the first example the position of the 'A' in 'April' in the SUGI_24 Macro variable is returned in X. The second performs exactly the same test on SUGI_25, but with the string 'April' stored in the Macro variable MONTH. In the third example the returned value, Z, is zero because the string 'XYZ' is not present in SUGI_25.

```
%LET X = %INDEX(&SUGI_24,April);  
%LET MONTH = April;  
%LET Y = %INDEX(&SUGI_25,&MONTH);  
%LET Z = %INDEX(&SUGI_25,XYZ);  
%PUT X= &X Y= &Y Z= &Z
```

X= 7 Y= 14 Z= 0

%SUBSTR(<Macro variable>,<starting point>,<sub-string length>) returns a sub-string from within a Macro variable. The second argument, Starting Point, is the position of the first character of the sub-string. The third argument is the length, or number of characters, which make up the sub-string. If this third argument is omitted the whole of the remainder of the Macro variable is returned.

```
%LET MONTH= %SUBSTR(&SUGI_25,14,5);  
%PUT &MONTH;
```

April

```
%LET FULLDATE=%SUBSTR(&SUGI_25,14);  
%PUT FULLDATE;
```

April 9 to 12 2000

%EVAL is used to 'evaluate', that is perform calculations using Macro variables which resolve to numeric values or arithmetic and logical operators. In this example the **%SUBSTR** function is used to extract the day numbers, which are then subtracted.

Adding 1 results in the duration as the number of days.

```
%LET NUMDAYS=  
    %EVAL(%SUBSTR(&SUGI_25,25,2) -  
        %SUBSTR(&SUGI_25,20,1)+ 1);  
%PUT Duration in Days = &NUMDAYS;
```

Duration in Days = 4

A few points to note about the **%EVAL** function are: (1) Arithmetic and logical operators have the same meaning and same operator precedence as in the Base SAS Language, (2) Division (uses the slash (/)) is integer division only. Division by zero returns a blank, and (3) The **%EVAL** function is implied in most circumstances. Problems can arise with implied **%EVAL** when a text string in a Macro variable contains a dash (-) due to confusion with a minus sign.

%SCAN(<Macro variable>,n) returns the nth 'word' in a Macro variable:

```
%LET CITY = %SCAN(&SUGI_25,1)  
%LET YEAR = %SCAN(&SUGI_25,6)  
%PUT The SUGI Location is &CITY in  
    the Year &YEAR;
```

**The SUGI Location is Indianapolis
in the Year 2000**

In the preceding NUMDAYS example **%SCAN** is a better function to use than **%SUBSTR**:

```
%LET NUMDAYS=  
    %EVAL(%SCAN(&SUGI_25,3) -  
        %SCAN(&SUGI_25, 5) + 1);  
%PUT Duration in Days = &NUMDAYS;
```

Duration in Days = 4

The **%SCAN** function can also take a third argument, which is a word delimiter list. The (documented) default delimiters are (in ASCII systems) <space>, underscore (_), period(.), and < (+ & ! \$ *) ; ^ - / , % |. If a delimiter list is specified these defaults are all overwritten. Any such new list applies only to the current **%SCAN** call.

The **%STR** function is used to handle commas, semicolons, and certain other 'problem' characters. **%STR** takes literally everything inside a string, except the apostrophe, and the % and & characters, which are evaluated accordingly. **%STR** is particularly useful when the intention is to store SAS code in a Macro variable, for example:

```
%LET DOREPORT =
  proc print data = allsales; var
    date custno itemcode qty value;
run;;
```

This would result in DOREPORT having the value 'PROC PRINT DATA = ALLSALES' and then an error occurring as the SAS compiler then tries to evaluate the VAR statement with no relevant preceding statement. The solution to this problem is to use %STR:

```
%LET DOREPORT =
  %STR(proc print data=allsales;
var date custno itemcode qty value;
run;);
```

The semicolons inside the parenthesis are not interpreted by the Macro pre-compiler but are stored as literal characters. The final semicolon, after the right parenthesis, tells the Macro pre-compiler the end of the required variable contents has been reached.

A couple of points to note here are %STR includes all spaces, including leading, trailing, and multiple spaces, and %STR will not work for apostrophes, for example:

```
%LET TITLE = %STR
  (SUGI 25 at America's Crossroads);
```

This would result in an 'Unmatched Quote' error. The way to prevent this is to have a percent (%) sign immediately preceding the quote so it is not interpreted

```
%LET TITLE = %STR
  (SUGI 25 at America%'s Crossroads);
```

%VERIFY(<mvar1>,<mvar2>) is used to compare one Macro variable with another, that is the function returns the position of the first character found in MVAR1 that is *not* found in MVAR2.

```
%VERIFY(&SUGI_24, &SUGI_25)
```

returns a 1 because the first character in SUGI_24 is a 'M' and there is no 'M' anywhere in SUGI_25. When each of the characters in the first string occurs at least once in the second string %VERIFY returns a 0.

%QUOTE, like %STR, is used to handle 'problem' characters, which the Macro pre-compiler itself uses. Examples are the semicolon (;), single or double quotation marks, and the dash (-), which is interpreted as a minus sign, for example if the Macro variable TABNUM is assigned the value 'PT01-21A' this code would cause an error:

```
%IF &TABNUM = &TABLE %THEN %DO;
```

The reason is the dash in the Macro variable TABNUM is interpreted as a minus sign because the %EVAL function is implied. Hence, the pre-compiler tries to 'subtract' '21A' from 'PT01', which results in the error 'A character operand was found in %EVAL where a numeric operand is expected'. The way to prevent this from happening is to use the %QUOTE function, which interprets everything as literal characters:

```
%IF %QUOTE(&TABNUM)= %QUOTE(&TABLE)
  %THEN %DO;
```

When using %QUOTE, if there is an unmatched quotation mark (apostrophe) in a character string a percent sign (%) is needed immediately preceding the apostrophe, as with the %STR function. For example:

```
%LET TITLE =
  %QUOTE(Summer 1999 Sales Results
(Director's Copy), September 1999);
```

```
%PUT &TITLE;
```

would result in an 'Unmatched Quote' error, or everything (all the following code) until a second apostrophe is encountered being stored in TITLE because of the apostrophe in the word 'Directors'. Placing a % sign immediately in front of the apostrophe (Director%'s) solves the problem:

```
Summer 1999 Sales Results
(Director's Copy), September 1999
```

For convenience, many other Macro functions have 'quotable versions' so using %QUOTE for each function argument is not necessary. Examples are:

```
%QUPCASE, %QSUBSTR, and %QSCAN.
```

Concatenating Macro Variables

Macro variables are concatenated by literally joining them together, no double bars (||) are needed as when joining Base SAS variables, but each variable to be joined must be preceded by an ampersand (&):

```
%LET SUGINO =
  %STR(SUGI 24 was held in );
%LET CITY = MIAMI;
%LET YEAR = 1999;
%LET LASTSUGI =
  &SUGINO&CITY%STR(, Florida, in
)&YEAR;
```

The value of LASTSUGI is now:

```
SUGI 24 was held in MIAMI, Florida,
in 1999
```

Note the use of the %STR function to preserve leading and trailing spaces. Resolution of a preceding Macro variable stops when another ampersand (&) or a percent sign (%) is encountered, as with SUGINO and CITY, however, if a Macro variable is to be concatenated to a non Macro item such as a literal string or a Base SAS variable a period (.) must be used to break the resolution. For example if MX is a Macro variable and SX is a Base SAS variable:

&MXSX

would result in a WARNING that the Macro variable MXSX is undefined, (or an ERROR if the resulting 'variable' was more than eight characters long in SAS Version 6 or earlier)

A period must be used as a delimiter. &MX.SX is needed to combine the value of &MX and the runtime value of SX. A point to note is that when a Macro variable is used to store, say, a library name, when a period is used as part of the Base SAS code *two* periods are needed to delimit the library name from the member name. For example if the library name DRUGTEST is stored in the Macro variable CLINLIB and a data set name STUDY025 is stored in the Macro variable DATASET the data set must be referred to using two periods between CLINLIB and DATASET:

```
%LET CLINLIB = drugtest;
%LET DATASET = study025;

data test01;
    set &CLINLIB..&DATASET;
run;
```

The first period tells the Macro pre-compiler when to stop resolving the Macro variable CLINLIB and the second period is the SAS delimiter between the library name and the data set (member) name.

&CLINLIB.&DATASET

would result in an error because the SAS compiler would try to resolve 'DRUGTESTSTUDY025'. This would happen regardless of whether the data set member name was a resolved Macro variable or a SAS data set name. The same rule applies to format references when the format name is stored in a Macro variable:

```
%LET FORMAT = BEST12;
```

To resolve the format name as a SAS format two periods must be used, for example:

```
chnum = put(x,&format..);
```

The first period tells the Macro pre-compiler to end resolving FORMAT and the second period is the

period needed for the SAS system to recognize BEST12 as a format, that is 'BEST12.'

Macro Control Structures

The SAS Macro language has the same control structures as the Base SAS System. The following example uses an %IF %THEN %ELSE construct to create a data set containing clinical trials data by sub-setting the data set DRUGDATA by one of three groups of patients depending on which treatment group, 1, 2, or 3, stored in the Macro variable TRTCODE is selected. The patient groups are 1 = Patient Numbers up to and including 999, 2 = Patient numbers 1000 to 1999, and 3 = Patient Numbers of 2000 and over. The name of the data set and the contents of the WHERE clause are determined by the value of TRTCOMP. Any number of Base SAS or SAS Macro statements may be placed between each %IF %THEN %DO and the corresponding %END statements. The use of a final %ELSE %DO...%END construct is to allow for the possibility of TRTCODE having an invalid value. For maximum efficiency the %IF %THEN %ELSE construct should be in order of decreasing likelihood of each condition being true. In this example if the value of TRTCODE was usually 3 the %IF &TRTCODE = 3 should be the first condition tested.

```
DATA GROUP&TRTCODE;
    SET DRUGDATA(WHERE=(
%IF &TRTCODE = 1 %THEN %DO;
        PATIENT LT 1000 AND PATIENT NE.
%END;
%ELSE %IF &TRTCODE = 2 %THEN %DO;
        1000 LE PATIENT LT 2000
%END;
%ELSE %IF &TRTCODE = 3 %THEN %DO;
        PATIENT GE 2000
%END;
%ELSE %DO;
        PATIENT LT 0
        %PUT ERROR IN TREATMENT CODE
TRTCOMP = &TRTCOMP;
%END;
    ));
RUN;
```

%IF %THEN %ELSE constructs may be nested and Boolean expressions may be written in SAS Macro using the %EVAL Macro function, for example:

```
%LET X = %EVAL(&MV1=2 AND &MV2=7);
%LET Y = %EVAL(&MV1*3=9 OR &MV2=6);
%LET Z = %EVAL(&PET1 EQ DOG) +
        %EVAL(&P2 EQ CAT);
```

In the first example X has the value 1 when both MV1 is 2 and MV2 is 7. If either or both of these expressions are false X has the value 0. In the second example if either or both expressions are true Y has the value 1, if both are false Y is 0. In the third example, when PET1 has the value DOG and PET2 has the value CAT Z is assigned the value 2. When PET1 is DOG and PET2 is not CAT Z is 1. When PET1 is not DOG and PET2 is CAT Z is also 1. When PET1 is not DOG and PET2 is not CAT Z is 0.

%DO is also used in loop structures. In the following example there are eight data sets named TEST01 to TEST08, with corresponding variables and attributes, each containing clinical trials data. The variable PATIENT contains the patient number, DATEDOSE is the date when a dose of test medication was taken, DOSEAMT is the amount of the dose, and DOSEUNIT the dose unit of measurement. If the required data set, TEST50, is to contain all observations where patients were given a dose of 50 mg of the test medication, this construct, using **%DO**, would produce the required results:

```
PROC DATASETS NOLIST;
%DO I = 1 %TO 8;
  APPEND BASE=TEST50 DATA=TEST0&I
    (KEEP=PATIENT DATEDOSE
      DOSEAMT DOSEUNIT
      WHERE=(DOSEAMT=50 AND
        UPCASE(DOSEUNIT)='MG'));
%END;
RUN;
```

The Macro variable I is being used as a counter with a starting value of 1 and a finishing value of 8. A **%BY** may be used to increment the counter by a value (including a negative value) other than 1. In this example **%DO I = 1 %TO 8 %BY 2**; would append the matching observations from TEST01, TEST03, TEST05, and TEST07. **%DO** loops may be nested and any number of Base SAS or Macro statements may be placed within each **%DO** - **%END** block.

%DO may be used with **%WHILE** or **%UNTIL**. The above example could be rewritten as:

```
PROC DATASETS NOLIST;
%LET I = 1;
%DO %WHILE (&I LE 8);
  APPEND BASE=TEST50 DATA=TEST0&I
    (KEEP = PATIENT DATEDOSE
      DOSEAMT DOSEUNIT
      WHERE = (DOSEAMT = 50 AND
        UPCASE(DOSEUNIT) = 'MG'));
  %LET I = %EVAL(&I+1);
%END;
RUN;
```

The **%WHILE** condition must be in parenthesis and the **%DO** loop will execute while this statement is still true, the loop stops as soon as this condition becomes false. If, in this example, the LE operator was replaced with LT the data set TEST08 would be omitted. Notice that the counter variable I is not automatically incremented, the first **%LET** statement is needed to initialize the counter to its starting value and the second **%LET** statement with **%EVAL** is needed to increment the counter within the loop.

Using **%UNTIL**, this example would be rewritten as:

```
PROC DATASETS NOLIST;
%LET I = 1;
%DO %UNTIL (&I GT 8);
  APPEND BASE=TEST50 DATA=TEST0&I
    (KEEP = PATIENT DATEDOSE
      DOSEAMT DOSEUNIT
      WHERE = (DOSEAMT = 50 AND
        UPCASE(DOSEUNIT) = 'MG'));
  %LET I = %EVAL(&I+1);
%END;
RUN;
```

%UNTIL is very similar to **%WHILE** except the loop executes while the condition (which must be in parenthesis) is still false. Execution stops as soon as the condition becomes true.

%GOTO is used to direct control to a Macro label. The above example, if rewritten using **%GOTO**, would be:

```
PROC DATASETS NOLIST;
%LET I = 1;
%DOAGAIN:
%IF (&I GT 8) %THEN %DO;
  %GOTO FINISH;
%END;
  APPEND BASE=TEST50 DATA=TEST0&I
    (KEEP = PATIENT DATEDOSE
      DOSEAMT DOSEUNIT
      WHERE = (DOSEAMT = 50 AND
        UPCASE(DOSEUNIT)= 'MG'));
  %LET I = %EVAL(&I+1);
  %GOTO DOAGAIN;
%FINISH:
RUN;
```

In this example the counter variable I is initialized to its starting value of 1. The **%IF** statement tests to see

if the value has exceeded the finishing value of 8. If the condition is false the APPEND statement is compiled for execution by the Base SAS system and the second %LET statement then increments the counter. The second %GOTO statement returns control to the %DOAGAIN label and the value of I is tested again. This process repeats until the %IF test is true and then control jumps to the %FINISH label. The SAS Macro labels themselves must begin with a percent sign (%) and be terminated with a colon (not a semicolon).

Within loops there may be situations where Macro variables may be double resolved, using two ampersands (&&). This is best illustrated with an example. Suppose a printed report is to contain four footnotes, starting in report column 10 and written underneath each other. The text of each footnote can first be defined in sequential Macro variables as:

```
%LET FN1 = %QUOTE(H = Patients
    given High doses (Over 75mg));
```

```
%LET FN2 = %QUOTE(L = Patients
    given Low doses (Under 25mg));
```

```
%LET FN3 = %QUOTE(N = Patients
    given No Doses or Ineligible);
```

```
%LET FN4 = %QUOTE(X = Amount of
    dose Unknown);
```

The %QUOTE function is needed because of the parenthesis. An easy way to print these footnotes in the assigned order is to use this Macro code after the code which prints each page of the report:

```
%DO I = 1 %TO 4;
    put @10 "&&FN&I";
%END;
```

The Macro pre-compiler first resolves the highest 'level' of the ampersands (&), that is the outermost ampersands, in this case these are the first ampersand preceding the 'FN' and the ampersand preceding the 'I'. Effectively this first resolution results in:

```
put @10 "&FN1";
put @10 "&FN2";
put @10 "&FN3";
put @10 "&FN4";
```

Resolution of the remaining ampersand (&) now results in this Base SAS Code being passed to the SAS compiler:

```
put @10 "H = Patients given High
doses (Over 75mg)";
```

```
put @10 "L = Patients given Low
doses (Under 25mg)";
put @10 "N = Patients given No
Doses or Ineligible";
put @10 "X = Amount of dose
Unknown";
```

Passing values between Base SAS and SAS Macro

Although the SAS Macro language is handled by a pre-compiler, some interaction takes place between the SAS Macro language and the Base SAS system at run time and values can be passed between SAS variables and Macro variables. When the ampersand (&) or the percent sign (%) are used the precompiled result of the Macro code is used by the SAS compiler, for example both of these expressions have the same end result:

```
X = %SUBSTR(&SUGI_25, 20, 1);
```

```
X = SUBSTR("&SUGI_25",20,1);
```

The SAS variable X is assigned the value 9 at runtime, but the resolution of the Macro variable SUGI_25 takes place at pre-compile time. To handle runtime exchanges between the SAS and SAS Macro languages two functions CALL SYMPUT and CALL SYMGET are used in a DATA Step, or SELECT INTO is used in PROC SQL.

CALL SYMPUT places the run time value of a SAS variable inside a DATA Step into a Macro variable at run time, for example:

```
CALL SYMPUT('REPGROUP', TRTCODE);
```

This statement places the run time value of TRTCODE into the Macro variable REPGROUP. The SAS variable may be numeric or character, but the Macro variable must be in single or double quotes.

SYMGET is the 'opposite' function of SYMPUT and assigns the resolved value of a Macro variable to a SAS runtime variable in a DATA step.

```
TRTCODE = SYMGET('REPGROUP');
```

The value contained in REPGROUP is placed as the runtime value of TRTCODE. Note that an ampersand (&) is not needed to resolve the Macro variable in this case.

In PROC SQL the SELECT clause is used with INTO to place the run time values of one or more of the SAS variables (columns) into one or more MACRO variables:

```
PROC SQL NOPRINT;
  SELECT <var1>, <var2>
    INTO :<Macvar1>, :<Macvar2>
    FROM <data table or view>;
QUIT;
```

Run time values of the selected variables in the data table (data set) or view are placed into the corresponding Macro variables, each Macro variable must be preceded by a colon (:) and be separated by a comma. The NOPRINT SQL option is to disable printing to the SAS Output file.

In this example the number of observations in a clinical trials data set, CLINTEST, is being written to the Macro variable VISITS

```
PROC SQL NOPRINT;
  SELECT COUNT(*) INTO :VISITS
  FROM CLINTEST;
QUIT;
```

This next example selects the values of PATIENT, DATEVIS, and DOCTOR into the Macro variables PATNO, VISDATE, and PHYSICN according to the WHERE clause

```
PROC SQL NOPRINT;
  SELECT PATIENT, DATEVIS, DOCTOR
    INTO :PATNO, :VISDATE, :PHYSICN
    FROM CLINTEST
    WHERE (PATIENT=2004 AND
          DOCTOR='BORIS');
QUIT;

%PUT &PATNO &VISDATE &PHYSICN;

2004 01APR99 BORIS
```

If more than one observation meets the WHERE clause criteria the first matching observation has its values stored in the Macro variables. If no observations match the WHERE clause the Macro variables are returned blank.

To store values from more than one observation in Macro variables a separating delimiter is specified using the SEPARATED BY clause. In this next example all the DOCTOR names are listed into the Macro variable MEDPROS, each name separated by a comma (any character may be used). The DISTINCT keyword used on DOCTOR selects the unique DOCTOR names and hence prevents duplication of DOCTOR names (in practice there

could be two or more different doctors with the same name, so an ID number is preferable). The default order in which the DOCTOR names are placed into the Macro variable is the same order as the names appear in the table. However, ORDER BY DOCTOR may be used to sort the Macro variable contents. When a character variable is written to a Macro variable, trailing spaces are eliminated unless NOTRIM is specified, when the full length of the character variable including all embedded spaces is stored.

```
PROC SQL NOPRINT;
  SELECT DISTINCT DOCTOR INTO
    :MEDPROS SEPARATED BY ','
  FROM CLINTEST;
QUIT;
```

Another way to store multiple values is to specify a range of Macro variables. In this example all the distinct patient numbers are stored in numerical order in the Macro variables PAT1 to PAT999. If there are less than 999 patients in the data set the excess Macro variables are left blank. More than one range of Macro variables may be specified at the same time, each range being delimited by a comma (,).

```
PROC SQL NOPRINT;
  SELECT DISTINCT PATIENT INTO
    :PAT1 - :PAT999
    ORDER BY PATIENT;
QUIT;
```

System Macro Variables

The SAS system provides several built in system Macro variables, some of the most frequently used are:

&SYSDATE returns the date in the format DDMMYY. (e.g. 27SEP99 - This is a Y2K problem in version 6).

&SYSDAY returns the current day of the week (e.g. MONDAY)

&SYSTIME returns the time in military HH:MM format (e.g. 14:45)

&SYSERR returns a return code from the most recently executed SAS procedure, 0 means the procedure ran successfully, 4 means a WARNING occurred, and 8 means an ERROR occurred.

&SYSSCP returns the operating system name

&SYSJOBID returns the current job id

&SQLOBS is the number of observations processed in a data set or view by the most recently executed PROC SQL code segment

&SQLRC is a return code from the most recently executed segment of PROC SQL code.:

0 = The SQL Procedure ran successfully
4 = A WARNING occurred
8 = An ERROR occurred

&SQLXRC and **&SQLXMSG** contain DBMS specific return code information when the SQL Pass Through facility is used.

SAS MACRO and the SASLOG

If MPRINT or MLOGIC or SYMBOLGEN are specified in an OPTIONS statement aspects of the SAS Macro code are 'explained' in the SASLOG. For example, if MPRINT is used, the name of each macro 'module' is listed at each line, the values of Macro variable loop counters, and whether %IF %THEN conditions are TRUE or FALSE are shown.

Macro Libraries and SASAUTOS

One of the main reasons for using %MACRO %MEND blocks of code is to allow repeated use by different applications or 'calling programs'. Typically SAS Macros are stored in libraries and are called by name, quite often passing parameters. When a Macro is successfully compiled it is stored in the default Macro Library catalog (SASMACR CATALOG in the WORK directory). A SAS Macro may be 'included' into a program specifying:

%INCLUDE "<Macro Filename>";

The Macro Filename is the host operating system filename containing the required macro(s). A more convenient method is to use the SAS Autocall facility. An option called SASAUTOS may be added to an OPTIONS statement to specify a search hierarchy for when a Macro is called (Similar to FMSTRCH):

```
OPTIONS SASAUTOS = ( 'MACLIB1',  
                    'MACLIB2', 'MACLIB3' );
```

The Macro libraries are enclosed in quotes and are separated by commas. The Macro library names are the filenames containing the Macros, their syntax is therefore dependent on the host operating system. In this example when a Macro is called the default SASMACR work library is searched first, if the Macro

is not found there MACLIB1 is searched, then MACLIB2, and then MACLIB3 until the Macro is found. If the Macro is not found in MACLIB3 a 'Statement is invalid or used out of proper order' ERROR is returned with a WARNING that the Macro invocation was not resolved. When a new SASAUTOS option is specified the previous SASAUTOS definition is lost.

Conclusions

SAS Macro is a major, and in most applications, an essential part of the SAS System. SAS Macro allows repetition and reuse of code, conditional compilation, the use of parameters, and SAS code substitution. Programmers, through the use of the SAS Macro language, are able to write more efficient, more modular, easier to test and validate, and more easily documented code. The SAS Macro language has much in common with the Base SAS language and is hence easy for beginner SAS programmers to learn and for class instructors to teach.

Acknowledgements

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. TM indicates USA registration.

Timothy J Harrington
Trilogy Consulting Corporation
Consultant Resource Center
5148 Lovers Lane
Kalamazoo
MI 49009

(616) 344 1688 extn 318

TJHARRIN@TRILOGYUSA.COM