

SAS Macro Language Quick Study

Clarence Jackson, CSQA, City of Dallas, Texas

ABSTRACT:

The purpose of this paper is to briefly describe the Macro Facility in BASE SAS. The information presented in this handout is based on information found in BASE SAS Users Guide and the SAS Macro Facility Guide. The Macro Facility was introduced with the 82 Release of BASE SAS. The outline of the paper is as follows:

- I. Introduction
 - A. What is the Macro Facility?
- II. SAS and Macro Facility Processes
- III. Macro Facility Overview
 - A. Components of Macro Facility
 - 1. Macro Processor
 - 2. Macro Language
 - B. Macro Language Components
 - 1. Variables
 - 2. Statements
 - 3. Functions
 - 4. Expressions and Text
 - C. Macro Definitions and Referencing
 - D. Referencing Environments
 - E. Programming Statements
 - F. Functions
 - G. Auto Call Facility
 - H. System Options
- IV. Quick Rules

I. INTRO:

The purpose of this handout is to briefly describe the Macro Facility in BASE SAS. The information presented in this handout is based on information found in SAS BASICS users guide and the SAS Macro Facility Guide.

A. WHAT IS MACRO FACILITY

There are two components in the Macro Facility:

- The Macro processor is the portion of SAS that does the work.
- The Macro language is the language used to communicate with the Macro processor

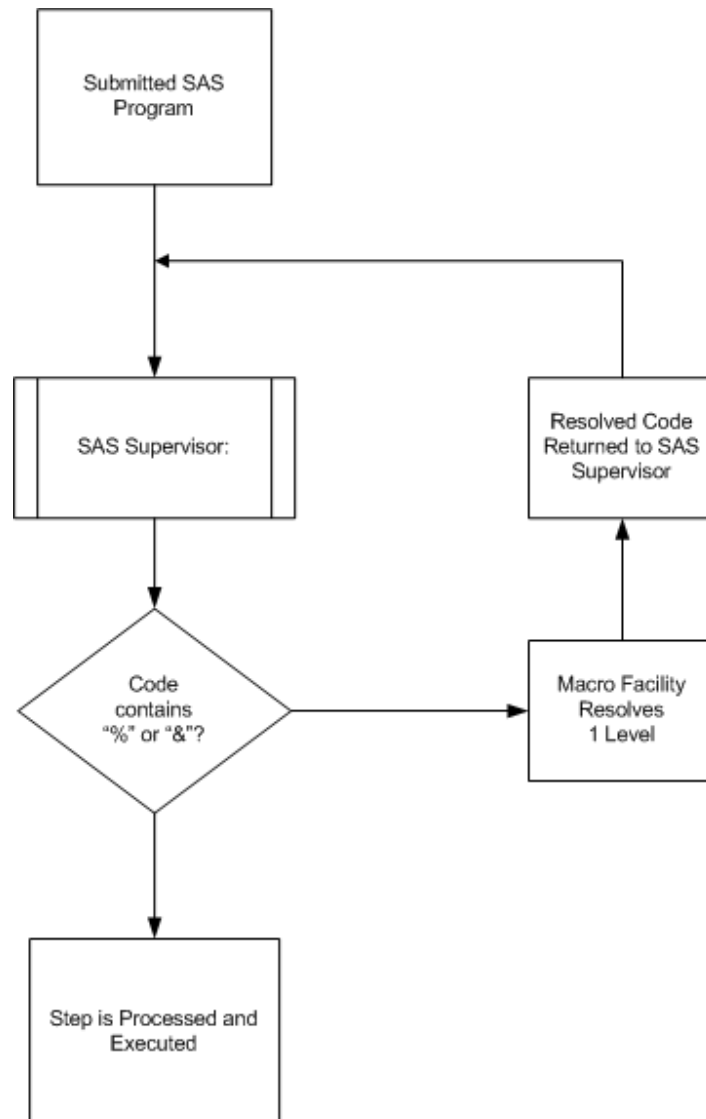
The Macro language has variables, program statements, expressions, and functions - the same pieces that make up the SAS DATA step language. However, the Macro language uses the patterns %name and &name to trigger Macro processing activities.

II. PROCESS

To understand how SAS uses Macro's, we must look at how SAS processes any statement or job. When we execute jobs in SAS, all SAS statements are read into the input stack, and passed thru the SAS supervisor one token at a time. Tokens are strings separated by blanks or operators. The SAS supervisor reads tokens until it receives a semi-colon, which completes a SAS statement. Once the supervisor reads a complete statement, it checks it for syntax, and passes the statement to the word scanner section of the supervisor. Here, the tokens are examined for their value, i.e. if the token is a SAS reserve word, user name and type, and passes the tokens on to the compiler. This process continues until a step boundary word is read. Step boundary keywords are DATA, PROC, RUN, ENDSAS, CARDS (and its other formats). Then the SAS

compiler compiles the code and executes statements for that step. In other words, only one step is compiled and executed independently.

Macro compilation and execution are separate processes from Data or PROC compilation and execution. Understanding the difference will help you to understand Macro expression evaluation and how to make SAS work better for you.



III. MACRO FACILITY OVERVIEW:

Now, why do we use Macros? Using the Macro Facility allows you to do very interesting things with SAS that is not possible without it. The SAS Macro Facility allows us to:

- Retrieve system information from the SAS Supervisor
- Perform conditional execution of SAS Data and Proc Step
- Generate data-dependent SAS statements (SAS Code)
- Communicate information between SAS steps
- Generate repetitive SAS code

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. The Facility allows you to package small or large amounts of text into units or modules that have names. From that point on, you can work with the names rather than the text itself. When you use a Macro Facility name in a SAS program, the Macro Facility generates SAS statements and commands as needed. The rest of the SAS system uses those statements and commands just like the ones you enter.

A. COMPONENTS OF MACRO FACILITY

There are two components in the Macro Facility:

1. The Macro processor is the portion of SAS that does the work.
2. The Macro language is the language used to communicate with the Macro processor

The Macro language has variables, program statements, expressions, and functions - the same pieces that make up the SAS DATA step language. However, the Macro language uses the patterns **%name** and **&name** to trigger Macro processing activities.

B. MACRO LANGUAGE COMPONENTS

The components of the Macro Language are:

- Macro Variables
- Macro Statements
- Macro Functions
- Macro expressions
- Constant Text

Macro variables (or symbolic variables) are different from DATA step variables and can be defined anywhere in a SAS job, except after a CARDS, CARDS4, DATALINES, PARMCARDS, or PARMCARDS4 statement. Whereas a Data step variable is assigned within that step, and must be carried within a SAS data file past the step, a Macro variables' value can remain constant until changed or the end of the SAS session.

C. MACRO DEFINITIONS AND REFERENCING

The simplest way to define and assign a value to a macro variable is to use %LET statement, as in:

```
%LET DSN=NEWDATA;
```

DSN is the name of the Macro variable, and NEWDATE is its value. To reference the value of a Macro variable, place an "&" in front of its name, as in &DSN. The pattern &DSN is called a "Macro variable reference". After creation of the Macro variable, the reference is resolved by the Macro processor using the &DSN. Changing the Macro is as easy as another "%LET" definition of the variable name. Using the %LET statement, you can create variables values that contain entire sections of a SAS program:

```
%LET PLOT=%STR(PROC PLOT; PLOT INCOME + AGE;);
```

In this case, the value of the macro variable is enclosed in the %STR Macro function so that the semicolons within the value are part of the TEXT, not end of the %LET statement.

A % MACRO statement defines stored text by name. The simplest type of Macro works much like a Macro variable, but more complex code can be included. %Macros can do many things that Macro variables cannot.

To define a %MACRO, a simple example is:

```
%MACRO DSN; PROC PLOT; PLOT INCOME + AGE; RUN %MEND DSN;
```

The "%MACRO" statement must begin every Macro and must contain a name for the Macro (DSN), text (NEWDATA) and end with the %MEND statement. To invoke a Macro, place a "%" in front of its name, as in "%DSN". The pattern "%MACRO-NAME" is called a "Macro invocation" or a "Macro Call". After definition of the Macro, the call is resolved by the Macro processor by replacing the Macro name with the text from the Macro. As with the %LET statement, you can create Macros that contain entire sections of SAS program:

```
%MACRO PLOT; PROC PLOT; PLOT INCOME * AGE; %MEND PLOT;
```

Suppose that the plotting variables in the PROC PLOT step can change. Rewrite the names of the PLOT variables with Macro variable references, and supply values in %LET statements before the Macro is called:

```
%MACRO PLOT; PROC PLOT; PLOT &YVAR * &XVAR; %MEND PLOT;
    %LET XVAR = INCOME;
    %LET YVAR=AGE;
    %PLOT
    %LET XVAR = SAVINGS;
%PLOT
```

In many cases, it becomes inconvenient to keep redefining variables using Macro %LET statements, so define the Macro variables as part of the %MACRO statement. This can be done using either positional or keyword parameters. The following illustrates both styles:

Positional Parameters -

Definition - %MACRO PLOT(XVAR,XVAR);
 PROC PLOT; PLOT &YVAR * &XVAR;
 %MEND PLOT;

Call - %PLOT(INCOME,AGE)

Keyword Parameters -

Definition - %MACRO PLOT(YVAR=,XVAR=);
 PROC PLOT; PLOT &YVAR &VAR;
 %MEND PLOT;

Call - %PLOT(YVAR=INCOME,XVAR=AGE)

I prefer using keyword parameters because variables defined can be assigned default values such as:

```
%MACRO PLOT(YVAR=INCOME,XVAR=AGE);
    PROC PLOT&YVAR*&XVAR;
    %MEND PLOT
    %PLOT(XVAR=SAVINGS)
%PLOT
```

(Two calls are issued here. The first will produce a plot for income by 'savings * age'. The second call will use the default value of &XVAR, 'age'.)

Macro variables can also be created during data step execution based on data values within the DATA step by using the "SYMPUT" routine, and passed to the Macro Facility to be used in other steps. By the same token, values in the Macro Facility can be passed to the data step by using the "SYMGET" routine.

D. REFERENCING ENVIRONMENT:

When the System Option MACRO is in effect, SAS programs have one or more referencing environments: a referencing environment is the area in which variable is available for use. Referencing environments are important because they determine whether the MACRO processor can locate a Macro variable that is referenced. The two types of Referencing Environments are "GLOBAL" and "LOCAL".

The Global environment is the entire SAS job, while local environments are within Macros. The Macro processor uses referencing environments according to the following rules:

1. When executing Macro program statements that can create Macro variables, the Macro processor attempts to change the value of an existing Macro variable regardless of environment, rather than creating a new Macro variable.
2. When executing Macro program statements that can create Macro variables and no Macro variable of the name requested exists, the Macro processor creates a variable in the current environment.
3. The %GLOBAL statement causes the Macro processor to create a variable in the global environment, regardless of the current environment.
4. %LOCAL statement and Macro parameter creates Macro variable in the current local environment even if a variable of that name exists in another environment.

E. PROGRAMMING STATEMENTS

Program statements for the Macro Language can be described as "open code" and "closed code" statements. Open code statements can be used outside any definition as well as inside. Closed code can only be used inside a Macro definition. All previously discussed Macro statements are open code statements. Others include %CMS, %INPUT, %PUT, %TSO, and all Macro functions and routines. Examples of closed code, or Inside-Macros-Only statements are %*Comment; %WHILE, %END, %GOTO, %IF-%THEN/%ELSE, and %LABEL: All of these statements are the same as SAS Data Step statements except that these statements program the Macro Processor. Using these statements allows for:

- Conditional execution of Macro statements and the generation of SAS code
- Branching within a Macro
- Supplies values for Macro variables during Macro execution and
- Creation of Macro variables and assigning values.

F. FUNCTIONS

Macro Functions process one or more Macro expressions, called arguments, produce a new character string, the result or returned value. They can be either inside or outside a Macro. There are three categories: Character, Evaluation, and quoting functions.

Macro character functions either change or provide information about the character string that is their argument. Most Macro character functions have names that correspond to DATA step character functions, and are similar in the way each works, except that Macro functions executes while SAS is constructing a DATA or PROC step. The DATA step function executes when the DATA STEP executes, and is treated as constant text by the Macro Processor, and unlike DATA step functions, Macro character functions may be paired with a "Q" for returning quoted results.

The evaluation function is %EVAL. Because the Macro Processor is a character (string) handling facility, "numbers" in Macro expressions are treated as character strings. The %EVAL function assigns numeric properties to integers in expressions and performs calculations using integer arithmetic. The %EVAL function also evaluates logical expressions such as &A=&B.

The quoting functions in the Macro language perform the activity equivalent to enclosing a portion of a SAS statement in single or double quotes. That the Macro Facility treats the expression with the function as a unit and not perform any evaluations on the expression. Quoting functions are used to cause the Macro Processor

to treat special characters as text. Quoting functions are necessary because the Macro processor treats single quote (') and double quotes (") as text.

G. AUTOCALL FACILITY

The Macro Facility AUTOCALL feature allows you to store the source statements for Macros in a file, associate the file with a SAS job or session, and call or include the Macros as needed. The Macro definition does not have to be part of the current program. By specifying the MAUTOSOURCE system options and associating the file reference (DD) SASAUTOS with a library of Macros, the AUTOCALL facility will automatically define and execute a Macro when it is called into a SAS job. This facility is especially useful for defining your own Macro functions, which you can store in an AUTOCALL library. SAS provides some special Macros in the SAS AUTOCALL library supplied with Base SAS.

H. SYSTEM OPTIONS

Normally, Macro statements don't appear on the log when executed. This is a problem when complex code is involved and the Macro is not working properly. Error codes for Macros are between 1000 and 4000 indicating that problem occurred in the Macro Facility. System options such as IMPRINT, MLOGIC, SYMBOLGEN and MACROGEN are available and can be used either separately or in combination. The options display information on the SAS log giving detail about the compilation or execution of a Macro. The type of messages that is displayed in the log depends on the option chosen.

IV. QUICK RULES

There are several sets of rules in use depending on the way the Macro Facility and SAS is used. In the batch environment, these rules for new Macro writers should be:

1. Define all Macros before use, preferably at the beginning of the SAS or AUTOCALLED.
2. Know your referencing environment, global, local, and current
3. Use systems options in debugging Macros
4. Use keyword parameters with Macro variable defaults
5. Document Macros with **%* comments**;

CONCLUSION

Using the SAS Macro Facility will save time in coding SAS programs, allow for modular and structured design of code, and a whole host of other benefits to the user.

REFERENCES

BASE SAS Users Guide and the SAS Macro Facility Guide

CONTACT INFORMATION

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

Clarence Jackson

City of Dallas

1500 Marilla 4DS

Dallas, TX 75201

E-mail: cjac@compuserve.com or clarence.jackson@dallascityhall.com

Web: <http://ourworld.compuserve.com/homepages/CJac>

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 trademarks of their respective companies.

APPENDIX

The following tables are for your reference only.

<u>Macro Language Statements Allowed in Macro Definitions and Open Code</u>	
Statement	Description
%* comment	designates comment text
%DISPLAY	displays a macro window
%GLOBAL	creates macro variables that are available during the execution of an entire SAS session
%INPUT	supplies values to macro variables during macro execution
%KEYDEF	assigns a definition to or identifies the definition of a function key
%LET	creates a macro variable and assigns it a value
%MACRO	begins a macro definition
%PUT	writes text or the values of macro variables to the SAS log
%SYSCALL	invokes a SAS call routine
%SYSEXEC	issues operating system commands
%SYSLPUT	defines a new macro variable or modifies the value of an existing macro variable on a remote host or server
%SYSRPUT	assigns the value of a macro variable on a remote host to a macro variable on the local host
%WINDOW	defines customized windows

<u>Macro Language Statements Allowed in Macro Definitions Only</u>	
Statement	Description
%DO	begins a %DO group
%DO, Iterative	executes statements repetitively, based on the value of an index variable
%DO %UNTIL	executes statements repetitively until a condition is true
%DO %WHILE	executes statements repetitively while a condition is true
%END	ends a %DO group
%GOTO	branches macro processing to the specified label
%IF- %THEN/%ELSE	conditionally processes a portion of a macro

%label:	identifies the destination of a %GOTO statement
%LOCAL	creates macro variable that are available only during the execution of the macro where they are defined
%MEND	ends a macaro definition

<u>System Options Used in the Macro Facility</u>	
Option	Description
CMDMAC	controls command-style macro invocation
IMPLMAC	controls statement-style macro invocation
MACRO	controls whether the SAS macro language is available
MAUTOSOURCE	controls whether the macro autocall feature is available
MERROR	controls whether the macro processor issues a warning message when a macro-like name (%name) does not match a compiled macro
MFILE	determines whether MPRINT output is routed to an external file
MLOGIC	controls whether macro execution is traced for debugging
MPRINT	controls whether SAS statements generated by macro execution are traced for debugging
MRECALL	controls whether the macro processor searches the autocall libraries for a member that was not found during an earlier search
MSTORED	controls whether stored compiled macros are available
MSYMTABMAX	specifies the maximum amount of memory available to the macro variable symbol tables(s)
MVARSIZE	specifies the maximum size for in-memory macro variable values
SASAUTOS	specifies one or more autocall libraries
SASMSTORE	specifies the libref of a SAS library containing a catalog of stored compiled SAS macros
SERROR	controls whether the macro processor issues a warning message when a macro variable reference does not match a macro variable
SYMBOLGEN	controls whether the results of resolving macro variable references are

	displayed for debugging
SYSPARM	controls whether the macro processor searches the autocal libraries for a member that was not found during an earlier search

<u>Automatic Macro Variables</u>	
Variable	READ/WRITE Status
SYSBUFFR	read/write
SYSCC	read/write
SYSCHARWIDTH	read only
SYSCMD	read/write
SYSDATE	read only
SYSDATE9	read only
SYSDAY	read only
SYSDEVIC	read/write
SYSDMG	read/write
SYSDSN	read/write
SYSENV	read only
SYSERR	read only
SYSFILRC	read/write
SYSINDEX	read only
SYSINFO	read only
SYSJOBID	read only
SYSLAST	read/write
SYSLCKRC	read/write
SYSLIBRC	read/write

SYSMENV	read only
SYSMMSG	read/write
SYSPARM	read/write
SYSPBUFF	read/write
SYSPROCESSID	read only
SYSPROCESSNAME	read only
SYSRC	read/write
SYSSCP	read only
SYSSCPL	read only
SYSSITE	read only
SYSSTARTID	read only
SYSSTARTNAME	read only
SYSTIME	read only
SYSUSERID	read only
SYSVER	read only
SYSVLONG	read only