

Introduction to the SAS® Macro Language

Tom Winn

Texas State Auditor's Office
Austin, Texas

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

Why bother learning about macros?

- Have you ever wanted to “package” blocks of SAS code into components which could be invoked with a single command?
 - » *This would be desirable, for instance, in repetitive coding situations involving the processing of numerous, similar data sets in identical ways.*

Why ... macros? (continued)

- Have you ever wanted to write a SAS program which modifies itself while it is running?
 - » *Perhaps you wanted to write flexible SAS code with the ability to accommodate itself to certain changeable details -- for instance, to check for data requiring special processing, to dynamically generate data-dependent lines in a report, or to communicate particular information between the steps in a program.*

Why ... macros? (continued)

- Have you ever wanted to be able to build and utilize a customized "toolbox" of frequently-used programming modules, without having to explicitly list them in each program in which they are used?

Why ... macros? (continued)

- For these situations, and for many others, use of the SAS macro language can produce the desired results.
- The SAS macro language is a powerful tool for producing flexible or repetitive code by using a modularized approach. It is particularly useful in applications development.

What are Macros?

- *Macros* are stored text which contain entire blocks of SAS code, and which are identified by a name. The stored text can include SAS statements, literals, numbers, macro variables, macro functions, macro expressions, or calls to other macros.
- *Macro variables* are used to facilitate symbolic substitution of strings of text, whereas *macros* can be used to manipulate SAS source statements.

What are Macros? (continued)

- Macro information can be inserted at any point in a SAS program simply by referring to the macro entity by name, preceded by a special character, which distinguishes macro statements from ordinary SAS code.

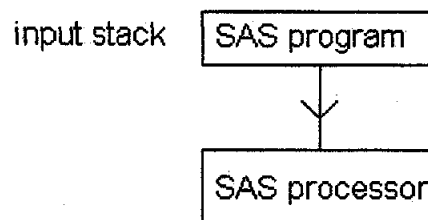
Macro variables are identified using the ampersand (&), and **macros** are identified using the percent sign (%).

What are Macros? (continued)

- These special characters (& and %) are “macro triggers.” If either of these triggers are followed by a non-blank character, the macro facility takes control over processing.
- The macro facility constructs and edits SAS source statements by substituting the currently-defined values of macro variables, and also by replacing macro names with the stored text which is associated with each of them.

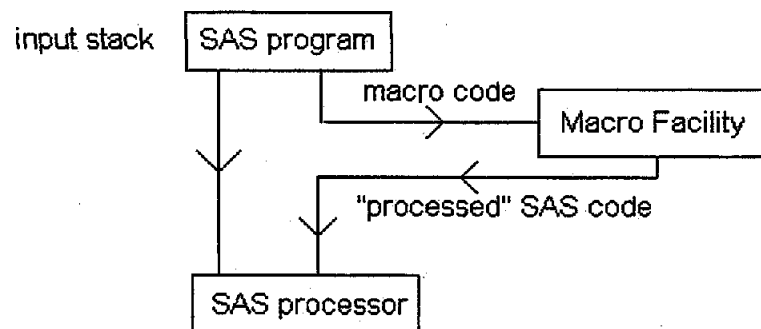
So, what does “macro” do?

Standard SAS Code:



So, what does “macro” do? -- continued

SAS Code With Macro Content:



The SAS Macro Language is a programming language.

- The macro facility follows instructions which are written using a special language.
- The SAS macro language has variables, statements, functions, expressions, and syntax. It works in conjunction with the SAS programming language.

Macro Variables

- Macro variables are used to facilitate symbolic substitution of strings of text.
- Macro variables are named following the usual rules for SAS names.
- A macro variable can be referenced by placing an ampersand immediately before the macro variable name and, optionally, by placing a period after the macro variable name.

Macro Variables, continued

- A macro variable reference causes the macro facility to substitute the current value of the macro variable for the reference.
- There are two types of macro variables: automatic macro variables, which are defined by the SAS Supervisor, and those which are defined by the programmer.

Macro Variables, continued

- Once it has been appropriately defined, a macro variable may be used repeatedly in the SAS job.
- Automatic macro variables are created by the SAS Supervisor when the SAS System is invoked. They include information such as the time, day, and date on which the job began executing, and the name of the most recently created SAS data set. Automatic macro variables are available for the duration of a SAS job and can be referenced anywhere in any SAS program.

Macro Variables, continued

- The **%LET** statement is the most common way to create user-defined macro variables, and to assign values to those variables.
- The value assigned to a macro variable is treated *as a character string* by the macro facility, but it isn't necessary to enclose the string in quotes. If quotes are used, the quotes become part of the string.
- There is no such thing as a *numeric* macro variable. In the macro language, everything is stored text.

Macro Variables, continued

- Here is the general syntax for **%LET**:
%LET *macrovariablename* = *value* ;
The name of a macro variable must conform to the customary rules for names in SAS.
- Here is a particular example:
%LET OPT = N OBS UNIFORM;
- Then, the statement
PROC PRINT &OPT;
would resolve as (that is, would be replaced by)
PROC PRINT N OBS UNIFORM;

Macros

- Macros are used to manipulate SAS source statements.
- A macro is defined by enclosing its text between a **%MACRO statement** and a **%MEND statement**.
- The %MACRO statement may include a parameter list (either *positional* or *keyword type*) to define special macro variables that can be referenced within the macro.

Macros, continued

- Here is the general syntax which is most commonly used:

```
%MACRO macroname(parameters);  
    . . . macrotext . . .  
%MEND macroname;
```

Macros, continued

- Macro variables which are defined within a macro generally cannot be resolved outside of the macro.
- It is possible to invoke a macro from within another macro. When this happens, the macros are said to be *"nested."*
- In general, a macro variable can be resolved in the macro within which it is defined, and in all other macros that are nested within that macro.

Macro Programming Statements and Macro Functions

- Most macro programming statements appear similarly, and are used in the same way, as corresponding ordinary-SAS programming statements, except that they begin with a percent sign (%).

Macro Programming Statements and Macro Functions, continued

- For example, the **%IF-%THEN-%ELSE**, **%DO**, **%DO %WHILE**, **%DO %UNTIL**, **%END**, and **%GOTO** statements are used in precisely the same manner as the familiar **IF-THEN-ELSE**, **DO**, **DO WHILE**, **DO UNTIL**, **END**, and **GOTO** statements.
- These statements facilitate conditional code generation, iterative processing, and branching within a macro.

Macro Programming Statements and Macro Functions, continued

- The SAS macro language includes several functions for manipulating strings which are similar to character string functions which might be used in a DATA step in ordinary SAS.
- **%LENGTH**, **%SUBSTR**, **%INDEX**, and **%SCAN** work just as you might suppose.

Macro Programming Statements and Macro Functions, continued

- Single quotes (‘ ’) and double quotes (“ ”) don’t have the same effect in the SAS macro language as in ordinary SAS.
- If a macro reference is placed within single quotes, it *will not* resolve (that is, it will be treated as constant text, and no symbolic substitution will occur).

However, macro variable references enclosed in double quotes *will* resolve.

Macro Programming Statements and Macro Functions, continued

- The SAS language uses quotes to indicate character constants. However, since quotes are considered part of the text in the SAS macro language, *quoting functions* take the place of quotes in macro expressions.
- Quoting functions are very useful for removing the customary syntactical meanings of special characters (like semicolons, unmatched parentheses, apostrophes, operators, and mnemonics) in character strings. Consult the SAS documentation for further information.

A Few Examples of the Use of Macros

- Here is an example of a macro, named REPET, which uses keyword-type parameters in an iterative loop to generate multiple DATA and PROC steps:

```
%MACRO REPET(FIRST=, LAST=);  
  %LOCAL I;  
  %DO I=&FIRST %TO &LAST;  
    DATA NET&I;  
      INFILE IN&I;  
      INPUT DEPT $ INCOME EXPENSE;  
      PROFIT = INCOME - EXPENSE;  
    PROC PRINT;  
      TITLE "REPORT OF INCOME & EXPENSE FOR &";  
    %END;  
  RUN;  
%MEND REPET;
```

A Few Examples of the Use of Macros, continued

- Simply defining a macro does not cause it to execute.
- Here is how to invoke the previously-defined macro:

```
%REPET(FIRST=1990, LAST=2000)
```

- ✓ Notice that the null default values of the parameters are replaced with desired values.
- ✓ Also notice that the statement does not end with a semicolon.

A Few Examples of the Use of Macros, continued

- Following is an example in which macro variables are used for the start date of the time period under consideration in a batch reporting job which normally would be run each Monday morning.
 - » Before submitting the program, the appropriate date elements are inserted as arguments in the MDY function for a variable named START.
 - » If the program is run on a Monday, and if the desired time period is the preceding week, then it isn't necessary to do anything to START.
 - » This program is easy to maintain and use, since the start date is only entered once at the beginning of the program, and the macro facility automatically substitutes values for two different forms of the date as often as they are required.

A Few Examples of the Use of Macros, continued

```
DATA _NULL_ ;
  START = MDY(10,20,2000);
  %GLOBAL STRTDA1 STRTDA2;
  IF &SYSDAY=Monday THEN DO;
    START2 = &SYSDATE - 7;
    CALL SYMPUT("STRTDA1",LEFT(PUT(START2,WORDDATE.)));
    CALL SYMPUT("STRTDA2",LEFT(PUT(START2,DATE.)));
  END;
  ELSE DO;
    CALL SYMPUT("STRTDA1",LEFT(PUT(START,WORDDATE.)));
    CALL SYMPUT("STRTDA2",LEFT(PUT(START,DATE.)));
  END;

DATA A;
  INFILE IN1;
  INPUT @1 VAR1 PD6. @7 VAR2 $2.
        @9 VAR3 $4. @13 DATEVAR MMDDYY10.;
  IF VAR2 = '02;
  IF DATEVAR GE "&STRTDA2"D;
  PROC PRINT;
  VAR VAR1 VAR3;
  TITLE "Records Processed Since &STRTDA1";
```

A Few Examples of the Use of Macros, continued

- The preceding program makes use of two automatic macro variables, **SYSDAY** and **SYSDATE**, which are created when the SAS System is invoked.
- The **SYMPUT** routine is used to transfer information from the DATA step to macro variables, during the execution phase of the DATA step.

A Few Examples of the Use of Macros, continued

- Sometimes, I use SAS to perform the same kind of statistical analysis and/or reporting on data files which share the same general structure, but which differ according to industry grouping or economic region. Macros are ideal for this situation!
- Following is an outline of a typical case.

A Few Examples of the Use of Macros, continued

```
%MACRO ANLYS(REGN);  
  DATA &REGN.2;  
    SET &REGN;  
    BY INDGRP;  
    ... (SAS programming statements)  
  PROC .....;  
    ... (SAS programming statements)  
  TITLE "Analysis for &REGN Region";  
  DATA .....;  
    ... (SAS programming statements)  
  PROC .....;  
  RUN;  
%MEND ANLYS;
```

A Few Examples of the Use of Macros, continued

- Notice the use of the parameter REGN. Macro parameters are special macro variables which are defined with the macro, and which are assigned a value when the macro is invoked. Also notice the use of a period as a macro variable delimiter in the first DATA step.
- Here is how the preceding macro would be called -- notice that each invocation includes a value for the parameter REGN:

```
%ANLYS(PLAINS)  
%ANLYS(METROPL)  
%ANLYS(EASTEX)  
%ANLYS(GULFC)  
%ANLYS(CENTRAL)  
%ANLYS(BORDER)  
%ANLYS(OUTSTA)
```

Macros can be stored in, and accessed from, macro libraries.

- The **autocall facility** is a way of making one or more libraries of commonly-used SAS macros available, without having to explicitly include them in each SAS program in which they are used.
- This allows SAS programmers to create macros for customized functions, and to have access to them whenever they are needed.
- There could be *personal* macro libraries, or *departmental* macro libraries for macros to be shared by many persons.

Macro Autocall Libraries, continued

- Autocall libraries --
 - » On MVS systems, autocall libraries are partitioned data sets.
 - » On a CMS system, an autocall library is a maclib.
 - » On a PC, an autocall library is a directory consisting of files with a 'sas' extension.
- The macros are stored as members in the library, where the name of the macro is the same as the name of the member. The libraries are concatenated, using the **fileref SASAUTOS**. Then, the **autocall facility** will automatically define and execute a stored macro whenever it is called in the SAS session, provided that the **MAUTOSOURCE** system option is in effect.

Suggestions for Further Reading

- SAS Institute Inc., SAS Guide to Macro Processing, Version 6, Second Edition, Cary, NC: SAS Institute Inc., 1990.
- SAS Institute Inc., SAS Macro Facility Tips and Techniques, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1994.
- SAS Institute Inc., SAS Macro Language: Reference, First Edition, Cary, NC: SAS Institute Inc., 1997.
- Thomas J. Winn Jr., "The SAS Macro Language: An Overview," Proceedings of SCSUG '91, pp. 267-272.
- Thomas J. Winn Jr., "Debugging SAS Macros," Proceedings of SCSUG '94, pp. 212-229; and SUGI-20 Conference Proceedings, 1995, pp. 346-352.
- Thomas J. Winn Jr., "Debugging SAS Macros, Revised," Proceedings of SCSUG '99, pp. 249-255.

Conclusion

- The SAS Macro Language is a powerful tool for simplifying repetitive coding, for communicating information between program steps, for generating data-dependent SAS statements, for permitting conditional execution of SAS code, and for dynamically importing certain information from the SAS Supervisor.
- In macro programming, a modularized approach is used.
- In this presentation, we have seen, or referred to, a few of the basic uses. And we also have listed several references for further study.

Author Information

- Tom Winn
10906 Rustic Manor Lane
Austin, TX 78750
- Telephone: 512 / 250-8610
- E-Mail: TWIN504@aol.com

