INTRODUCTION TO THE MACRO LANGUAGE

Sandra Hendren
Health Data Institute

The purpose of this paper is to explain the macro language at a conceptual level. It will not discuss the syntax of the language in any detail. Correct syntax can be acquired fairly easily from the manual. Instead, this paper will attempt to give an introduction to the way the macro language works internally in the SAS system in order that you can better program, debug, and design systems using it.

The first concept of importance is that this is a second language. It is similar in syntax and function to the SAS language but it is truly another language available in the SAS Basics product. It is a language whose output is SAS code. In that respect it is a code generator of the SAS language. Macro statements and variable resolutions execute prior to the compilation and execution of SAS statements within the same job stream. Therefore, the following example yields the subsequent SAS job:

PROGRAM:
options mprint symbolgen;
title "Example of Weird but Correct Code";

data one;
  input %let x=first; a b %put x=&x; c;
total=a %macro doit;
    proc means; run; %mend doit;
+ b;
cards;

10 15 20
25 30 35
40 45 50
55 60 65
70 75 80
85 90 100
; %doit
LOG:
37 options mprint symbolgen;
38 title "Example of Weird but Correct Code";
39
data one;
40 input %let x=first; a b %put x=&x; c;

SYMBOLGEN:  Macro variable X resolves to first x=first
42 total=a %macro doit;
43 proc means; run; %mend doit;
44 +b;
45 cards;
46 ;

NOTE: The data set WORK.ONE has 6 observations and 4 variables.
NOTE: The DATA statement used 7.00 seconds.

I do not recommend writing code as this example shows! I use it merely for instructional purposes to prove the point that macro statements and variable resolutions are processed before compilation (and therefore execution) of SAS code.

How is this possible? The diagram accompanying this paper shows how the SAS and macro languages work internally. The wordscanner, a part of the SAS supervisor, begins by tokenizing your submitted SAS program. A token is an English word, a punctuation mark, a number, or a quoted string. So that, as the wordscanner pulls one token at a time from the top of the input stack, it will pass that token either to the SAS language or to the macro language. Tokens are passed to the SAS language until an end of step token is encountered. (DATA, PROC, CARDS, or RUN.) When the SAS language receives such an end of step token, it will quit compiling that SAS step and begin execution of the code collected thus far. Thus, each SAS step within your program is compiled and executed, then the next SAS step is compiled and executed, and the next, and the next, until there are no further SAS steps in your job.

Notice that whenever the wordscanner pulls either of two tokens, an "&" or "%", followed by a non-blank token, these tokens are triggers that the wordscanner should pass that token (and the subsequent group of tokens) to the macro language. If the token is an "&", this is a request for the macro facility to return the current value of the variable it precedes. If the trigger is a "%" sign, the macro facility collects all tokens from the wordscanner until it gets to the end of that statement since this is a token that introduces an entire macro statement. The macro facility will then process its code and return the results of that
processing to the top of the input stack. The SAS language, in its quest for tokens from the wordscanner, has been temporarily put on hold. While the SAS language is sitting and waiting for its next token, the macro facility has been triggered and has taken control of that group of tokens in order to resolve the value of a macro variable or process a macro statement. The SAS language does not know this is happening. It is simply waiting for its next token in its attempt to compile code until it gets to an end of step token, at which time it will execute the currently compiled code.

This process allows a number of new applications in the SAS system via the macro language. The applications I feel to be most important (because it adds new functionality to the SAS system) are:

(1) You can perform conditional execution of SAS DATA and PROC steps.
(2) You can generate repetitive SAS code easily.
(3) You can pass parameters (i.e., macro variable values) from step to step within a single SAS job.

The following examples will show the three ways of creating macro variables and passing values to them. In these examples you will see some of the applications mentioned previously. There are three ways to create a macro variable:

(1) With a macro LET statement.
(2) As a parameter value on a call to a macro.
(3) By passing a data value into a macro variable in a data step (CALL SYMPUT).

(Here is the code that created the dataset used in the following examples:

```
69     data all;
70     infile cards eof=assign;
71     input id age sex $ charge area $;
72     assign: call symput('nbig',put( n ,8.));
73     cards;
82     ;
NOTE: The data set WORK.ALL has 9 observations and 5 variables. NOTE: The DATA statement used 6.00 seconds.)
```

Method #1 - the macro LET statement:

```
84     options mprint symbolgen;
85     %let city=Boston;
```
HENDREN / Introduction to Macros

86
87   data area;
88   set all;
89   if area="&city";
SYMBOLGEN: Macro variable CITY resolves to Boston
90   run;
NOTE: The data set WORK AREA has 4 observations and 5 variables. NOTE: The DATA statement used 16.00 seconds.

Method #2 - A parameter on a macro definition and its subsequent call:

91   %macro aver(dataset);
92   proc means data=&dataset;
93   var charge;
94   output out=one mean=avchg std=stdchg max=hichg; 95 run;
96   %mend aver;
97   o/oaver(all)
SYMBOLGEN: Macro variable DATASET resolves to all
MPRINT(AVER): PROC MEANS DATA=ALL;
MPRINT(AVER): VAR CHARGE;
MPRINT(AVER): OUTPUT OUT=ONE MEAN=AVCHG STD=STDCHG MAX=HICHG;
MPRINT(AVER): RUN;

NOTE: The data set WORK.ONE has 1 observations and 5 variables. NOTE: The PROCEDURE MEANS used 7.00 seconds.

Method #3 - Passing a data value into a macro variable and then using that value to generate SAS code.

116   data null ;
NOTE: The PROCEDURE PLOT used 1.37 minutes.
117   set one;
118   call symput('meanchg',put(avchg,8.2));
119   call symput('stdref',put(stdchg,8.2));
120   run;
NOTE: The DATA statement used 3.00 seconds.
121
122   proc plot data=area;
Since the macro language is indeed a second language in the SAS Basics system, that means you have a whole new set of error messages as well. Error messages returned from the macro language have codes greater than 1000, while error messages returned from the SAS language have error codes of less than 1000. Two options are available to you for assistance in tracing and debugging macro language statements - MPRINT and SYMBOLGEN.

MPRINT shows the tokens that the SAS language receives. This is probably the most useful debugging option, especially for programmer/logic errors. These are the errors where the macro language works correctly but returns tokens that, when compiled and executed in the SAS language, does not do what you intended it to do. This option is also the one you should use if you have an error message of less than 1000 (a SAS language error).

SYMBOLGEN prints on the log the resolution of macro variables. This option is most useful, therefore, for finding macro logic errors or macro syntax errors (error codes greater than 1000).

The addition of the macro language to the SAS system in 1982 made SAS an appropriate programming language for production-level business applications. The macro language can also create menu-driven systems for end users (as can the subsequent introduction of SAS/AF). But the internals of the macro language still are not well known. This paper merely introduces the most fundamental of those concepts.
HENDREN / Introduction to Macros

SAS PROGRAM (INPUT STACK)

DATA PERSON;
  INFILE RAWDATA;
  INPUT SSN AGE;

PROC PRINT;
  TITLE "R & D REPORT ON &SYSDAY"
RUN;

SAS SUPERVISOR

WORD SCANNER

MACRO FACILITY

MACRO PROCESSOR

SYMBOLIC PROCESSOR

SYMBOLIC TABLES

MACRO LANGUAGE

SAS LANGUAGE

COMPILE

EXECUTE