

MACROS MADE SIMPLE

Eric Webster, Bradford Exchange USA Ltd.

What about these macro things, anyway? Aren't they very complicated?

Like any program in SAS*, macros and elements of the macro language are as simple or as complicated as you make them. They can be hundreds of lines long and control complex interactive systems, or can be as simple as one extra line in a program that simplifies maintenance and control of a program. This presentation is an overview of the most simple aspects of the macro language, but these easy techniques can save you a great deal of time and trouble.

I. MACRO SUBSTITUTION VARIABLES

Macro substitution variables are pieces of text stored by SAS and are referred to by a valid SAS variable name. When the SAS system sees an ampersand "&" placed next to a variable name, it substitutes the text before running the DATA step or PROC.

For example, consider a program you have written that accesses your companies' SAS dataset of sales information, and prints a report of the sales made in a particular state. One approach would be:

```
DATA SALES;  
  SET DATALIB.STATESET;  
  IF STATE = 'MI';  
RUN;  
  
TITLE1 "SALES REPORT FOR STATE: MI";  
  
PROC PRINT NOOBS DATA=SALES;  
RUN;
```

This technique works well, but can be prone to error: the state name must be changed in both the DATA step (or WHERE statement if used instead of a data step) and TITLE line. This can be done with a global change in the editor, but consider if the last state was Indiana (IN) and a global change is executed to Michigan (MI). Any IN statement becomes MI, and the program bombs.

* SAS is the registered trademark of SAS Institute, Inc., Cary NC, USA.

A better way is to use the %LET function to define a macro variable:

```
%LET MYSTATE = MI;

DATA SALES;
  SET DATALIB.STATESET;
  IF STATE = "&MYSTATE";
RUN;

TITLE1 "SALES REPORT FOR STATE: &MYSTATE";

PROC PRINT NOOBS DATA=SALES;
RUN;
```

Now all you need to do to run a different report is to change the %LET statement and resubmit. Note that statements such as %LET do not have to be used inside a DATA step or PROC; they are like TITLE statements in that they can be used in "open code".

One of the nicest uses of the %LET is to define a dataset name. Consider a SAS data library SALES90, which contains the sales by state for your company for each month, e.g., SALES90.JANUARY, SALES90.FEBRUARY, etc. For every year you have libraries such as SALES85, SALES86, etc. You have written a program to print out the sales for a particular state for a particular month for a particular year, and after attending the WISAS meeting, you included the %LET to determine the dataset library and name:

```
%LET MYSTATE = WI;
%LET LIB      = SALES90;
%LET MONTH    = MARCH;

DATA SALES;
  SET &LIB.&MONTH;           <----- NO!
  IF STATE = "&MYSTATE";
RUN;
```

Unfortunately, this resulted in an error: "INVALID DATASET NAME SALES90MARCH". What went wrong? The period between %LIB and %MONTH is taken by SAS as meaning "This is the end of the name of the macro variable LIB", LIB is resolved into SALES90, and the period is ignored (in much the same way as the ampersand is ignored). So, to correct the situation, add a second dot. Rule to remember: when placing two macro variables together to build a two-level macro variable name, put two periods between the variable names.

```
DATA SALES;
  SET &LIB..&MONTH;          <----- YES!
  IF STATE = "&MYSTATE";
RUN;
```

Debugging tip: to test how a macro variable will turn out, use %PUT:

```
%PUT Using one period gives: %LIB.&MONTH;  
%PUT Using two periods gives: %LIB..&MONTH;
```

Returns in the SAS log:

```
Using one period gives: SALES90MARCH  
Using two periods gives: SALES90.MARCH
```

Like %LET, %PUT can be put anywhere (open code).

II. BASIC MACROS

Macros themselves are simply chunks of SAS code that are given a name; when SAS encounters the name, the SAS code is substituted. These can be especially effective when combined with %LET:

```
%MACRO STATERPT;  
  DATA SALES;  
    SET DATALIB.STATES;  
    IF STATE = "&MYSTATE";  
  RUN;  
  
  TITLE1 "REPORT FOR STATE: &MYSTATE";  
  
  PROC PRINT NOOBS DATA=SALES;  
  RUN;  
%MEND STATERPT;
```

Now, whenever SAS sees the invocation %STATERPT, the above code is substituted.

WHY?

Consider:

```
%LET MYSTATE = WI;  
%STATERPT;  
  
%LET MYSTATE = IN;  
%STATERPT;  
  
%LET MYSTATE = IL;  
%STATERPT;
```

This series of invocations reduces the total number of lines in your program file to one macro and three invocations; this version is easier to maintain and run.

The macro language is a full programming language itself, complete with control statements such as %IF, %DO-%END, %DO %WHILE-%END, etc. What would you do with control statements in a macro? Consider a program that needs to either run a daily report or, on Fridays, a weekly summary report:

```
%MACRO REPORTS;
  %IF &SYSDAY = Friday %THEN
    %DO;
      <code for weekly report>
    %END;
  %ELSE
    %DO;
      <code for daily report>
    %END;
%MEND REPORTS;

%REPORTS;
```

When submitted, this macro automatically determines if it is Friday or not and runs the proper code.

III. SYMPUT

One of the most useful systems in SAS is the CALL function SYMPUT, which can actually let PROC's share information with DATA steps. Normally in SAS all PROC's and DATA steps are independent, relating to each other only through creating and reading datasets. Sometimes, however, information generated by a PROC or determined through a DATA step would be of use in the actual code of a subsequent calculation, and SYMPUT can be used to communicate these.

Consider a dataset of sales by salesman by state. Since not all states are similar in terms of sales, salesmen are evaluated differently depending on the overall sales level of the state they work in - salesmen in high sales states must meet a different level of sales than those in low sales states. Management determines that all salesmen in a given state should be evaluated on an index relating their sales to the average sales for that state, with 100 being average, 75 being below average, and 125 being above average. You need to determine the average for the state and divide the sales for each salesman by the state average, e.g., if the state average sales was \$25,000 and salesman A had \$27,500 in sales, their index would be $100 * 27,500 / 25000 = 110$. You need to be able to generate each report separately for a state on demand.

```
%LET MYSTATE = WI;

PROC MEANS DATA=DATALIB.STATES;
  WHERE STATE="&MYSTATE";
  VAR SALES;
  OUTPUT
    OUT = MEANSET
    MEAN = AVGSALES;
  RUN;

DATA _NULL_ ;          <----- ONLY EXECUTING A DATA STEP TO RUN SYMPUT
  SET MEANSET;
  CALL SYMPUT('AVERAGE',AVGSALES);
  RUN;

DATA SALES;
  SET DATALIB.STATES;
  IF STATE="&MYSTATE";
  INDEX = 100 * SALES / &AVERAGE;
  RUN;

PROC PRINT NOOBS DATA=SALES;
  VAR SALESMAN SALES INDEX;
  RUN;
```

The PROC MEANS computes the average sales for the selected state, and writes it out to a one-variable, one-state dataset called MEANSET. A _NULL_ DATA step is executed which calls SYMPUT; the first argument to SYMPUT is the new macro variable name, passed as a character, and the second argument is the variable or value to pass to this macro variable. After the end of the DATA step from which SYMPUT is called (and not before!), the macro variable AVERAGE contains the average sales for that salesman (check with %PUT). The DATA step is then executed to compute the index for each salesman, and the PROC PRINT displays the information.

The above example could be made even more useful by enclosing all code in a %MACRO, and executing like:

```
%MACRO INDEXRPT;  
    <Above code for report>  
%MEND INDEXRPT;  
  
%LET MYSTATE = WI;  
%INDEXRPT;  
  
%LET MYSTATE = IL;  
%INDEXRPT;  
  
%LET MYSTATE = IN;  
%INDEXRPT;
```

Note that SYMPUT cannot be both called and used inside the same DATA step - the values are only assigned to the macro variables at the end of the DATA step execution. Finally, since all macro variables are stored as text, each use in a numeric computation such as $INDEX = 100 * SALES / \&AVERAGE$ involves converting the character value into numeric each time the statement is executed; if there are a large number of observations, this can be very time consuming. This can be avoided by assigning the value into a numeric variable at the beginning of the DATA step:

```
DATA _NULL_;  
    CALL SYMPUT('AVERAGE',AVGSALES);  
    RUN;  
  
DATA SALES;  
    RETAIN AVERAGE;  
    IF _N_ = 1 THEN AVERAGE = &AVERAGE;  
    INDEX = 100 * SALES / AVERAGE;  
    RUN;
```

Here the character value is converted only on the first pass, and stored in a numeric variable.