A TUTORIAL ON THE SAS® MACRO LANGUAGE

John J. Cohen, SAS Consultant

INTRODUCTION

The SAS macro language is a "super" language that rests on top of regular SAS programming statements. It creates "spaces" into which the programmer can fit information that, if used properly, can make programming easier, less tedious, and more fun. However, not every program is improved by using macros. It is another language syntax to learn, with rules that can be confusing. It also can create problems in "debugging" programs that are even more entertaining than those offered by regular SAS. Finally, the macro language requires additional memory (which may be at a premium in some environments) and takes longer to run.

Reasons to use the macro language include to pass values through a program, thereby avoiding some of the pitfalls of "hard coded" values. It is useful as a "code generator", saving repetitive and tedious effort. Macros can allow conditional execution of parts of your program, allowing for more sophisticated systems. Passing macro code "fragments" may make certain tasks far easier than using regular SAS alone. Macros can be used as a way to create "modular programs", pieces of code which can be standardized and re-used throughout your organization. Finally, macros can be used to create interactive systems, especially for prototyping and in the absence of SAS/AF. We will explore several of these, and finish with a brief discussion on macro debugging techniques.

MACROS

Macros consist of collections of regular SAS program statements, macro variables, macro language statements, and macro functions contained within a %MACRO and a %MEND. The %MACRO statement includes a name and the macro is "called" using that name—preceded by a %.

Figure 1 A Simple Macro

%MACRO MYFIRST;
PROC PRINT DATA=CLAIMS;
TITLE1 "TESTING MY FIRST MACRO";
TITLE2 "SO WHAT DID YOU EXPECT?";
%MEND MYFIRST;

%MYFIRST;

Macro MYFIRST is not very interesting, but it is syntactically correct. SAS recognizes that this is a macro (and not regular SAS programming statements) because of the % symbol. Note that these macro statements are ended with semi-colons (;), and the selection of double quotes (" ) for the TITLE statement is deliberate.
MACRO VARIABLES

Macro variables are indicated by preceding a variable name with an ampersand (&). Following standard SAS naming conventions, we will create a macro variable called &WHEN. We can assign character string value to &WHEN, which will be substituted in place of the space taken up by the macro variable at execution time. In this example we will assign a value to &WHEN when we call the macro.

Figure 2 Passing Values With Macro Variables

```sas
%MACRO CLAIMREP(WHEN=) ;
DATA REPORT; SET CLAIMS;
  IF DATE = &WHEN THEN CURRENT=AMOUNT;
  ELSE IF DATE = (&WHEN - 1) THEN PRIOR=AMOUNT;
  ELSE DELETE;
PROC PRINT DATA=REPORT;
  VAR CURRENT PRIOR;
  TITLE1 "MONTHLY CLAIMS REPORT FOR &WHEN";
  TITLE2 "NO, YOU DEFINE ""HARD-CODED"""";
%MEND CLAIMREP ;

%CLAIMREP(WHEN=9209);  
```

When macro CLAIMREP executes, all observations from dataset CLAIMS which have a DATE value equal to the value of &WHEN (the current month of 9209) or prior month (&WHEN-1, or 9208) will be output into dataset REPORT and printed. Note that we also use &WHEN in the TITLE statement, letting us know which month is contained in the report. We can submit next month's program by simply changing the macro variable value in the call to %CLAIMREP(WHEN=9210). We can verify last year's report with %CLAIMREP(WHEN=9109).

CLAIMREP is useful because it allows us to pass a value (using &WHEN) into the program without having to check it for each instance and then changing them individually. In a complicated program, the chances of forgetting at least one are remarkably high.

MACRO STATEMENTS

Our first macro statements were the %MACRO & %MEND. Other macro statements, often similar to their regular SAS equivalents, also start with the % symbol, such as %DO, %END, %IF %THEN, %ELSE, %GLOBAL, %LOCAL, and %LET. In the following example, we will run an annualized version of %CLAIMREP, once for each of the last ten years. We will use the macro language to "generate" ten copies of the program.
Figure 3 Using Macro Statements To Generate Code

```sas
%MACRO CLAIMREP;
%LET CURYEAR=92;
%DO YEAR=83 %TO 92;
   %IF "&YEAR" = "&CURYEAR"
      %THEN %LET NOTE = DATA IS ONLY THRU &SYSDATE;
      DATA REPORT; SET CLAIMS;
      IF YEAR = &YEAR THEN OUTPUT;
      PROC PRINT DATA=REPORT;
      TITLE1 "ANNUAL CLAIMS REPORT FOR 19&YEAR";
      TITLE2 "I LIKE SHORT PROGRAMS";
      FOOTNOTE "&NOTE";
   %END;
%MEND CLAIMREP;
%CLAIMREP;

Note that we assign value to &CURYEAR and &NOTE using a %LET statement, and to &YEAR with a %DO statement (all without ampersands). &SYSDATE is an automatic macro variable created by SAS. In TITLE1 the phrase 19&YEAR will "resolve" to 1983 when &YEAR = 83, to 1984 when &YEAR = 84, etc.

MACRO FUNCTIONS

Macro functions operate much like regular SAS functions, except that the arguments are within the context of the macro language. The bulk operate on character strings (e.g., %SUBSTR) or controlling the exact interpretation of macro special symbols (e.g., %STR). Because macro variables are character variables, the %EVAL function is required to perform arithmetic. The example above included a simple instance of conditional execution (whether or not to create a value for the FOOTNOTE statement. Below we will have the program elect to create a backup of the CLAIMS dataset if we are running the report at year's end (i.e., in month 12).

Figure 4 Macro Functions

```sas
%MACRO CLAIMREP(when=);
%IF %SUBSTR(&WHEN,3,2) = 12 %THEN %BACKITUP;
DATA REPORT; SET CLAIMS;
   IF DATE = &WHEN THEN CURRENT=AMOUNT;
   ELSE IF DATE = (&WHEN - 1) THEN PRIOR=AMOUNT;
   ELSE DELETE;
PROC PRINT DATA=REPORT;
   VAR CURRENT PRIOR;
   TITLE1 "MONTHLY CLAIMS REPORT FOR &WHEN";
   TITLE2 "&TITLE2";
   TITLE3 "&TITLE3";
%MEND CLAIMREP;
***************;
```
%MACRO BACKITUP;
%LET YEAR=%SUBSTR(&WHEN,1,2);
%LET BEGYEAR=%EVAL(&YEAR*100);
%LET ENDYEAR=%EVAL((&YEAR+1)*100);
LIBNAME OUT 'WHATEVER.WORKS.ON.YOUR.OPERATING.SYSTEM';
DATA OUT.CLAIMS&YEAR; SET CLAIMS;
   IF &BEGYEAR < DATE < &ENDYEAR;
%LET TITLE2 = AT END-OF-YEAR, 19&YEAR WAS BACKED UP";
%MEND BACKITUP;

%CLAIMREP(WHEN=9112);

The %SUSTR function allows us to capture the year portion of &WHEN, which we then manipulate arithmetically with %EVAL to produce begin and end values for selecting the data in the CLAIMS dataset to write out.

SPECIAL TOPICS

Along with the special symbols (& and %), on occasion we need to consider delimiters, characters than indicate the end of a macro variable. Blanks and equal signs (=) have been used above. Two others are the ampersand (again!) and the period (.). Figure 5 (%CENTER is borrowed from Gary Katsanis) illustrates the use of a program fragment, a section of code which could be written using just regular SAS statements, but it would much messier. We will center text strings for a report.

Figure 5  A Few Delimiters

%LET YEAR = 91;
%LET MONTH = 12;

DATA NULL; SET CLAIMS; FILE PRINT;
   PUT %CENTER(FINAL REPORT FOR &YEAR/&MONTH);
   PUT %CENTER(NOTE: BACKUP FILE CREATED &YEAR&MONTH);
   PUT %CENTER(---&YEAR./&MONTH---);

%MACRO CENTER(TEXT);
%LET SPACE = %EVAL(132-%LENGTH(TEXT));
%LET INDENT = %EVAL(&SPACE/2);
   @&INDENT "TEXT"
***WAS THAT A MISSING SEMI-COLON?***
%MEND CENTER;

DEBUGGING

Identifying and fixing macro language errors can be frustrating and frequently reduces one to brute trial-and-error methods. There are several SAS options (MPRINT, SYMBOLGEN, MACROGEN, MTRACE) which help tremendously. Remember to turn them off once the program is complete,
though. Make extensive use of the %PUT statement. Test incrementally, and comment out macro language references to verify that the regular SAS code is not the source of errors. Finally, moving programs between different releases of SAS can introduce problems that were not there before.

Figure 6 A Not So Simple Macro

DATA TRPK; SET IN.TRPEAK(KEEP=TRTREG CTR CPK);
  RENAME CTR=YO CPK=Y1; RUN;

%GLOBAL T FIRST CONTIN;
%LET CONTIN=YES;
%MACRO TTEST;
  DATA ONE; SET TRPK;
  ZT=YO-Y1*(&T/100);
  RUN;
  PROC MEANS DATA=ONE T PRT;
  VAR ZT;
  OUTPUT OUT=T&T T=T PRT=PVALUE; RUN;
  %IF FIRST=YES %THEN %DO;
  DATA ALLT; SET T&T;
  IF .05 <= PVALUE <= .95;
  ITERAT=&T; RUN;
  %END;
  %ELSE %DO;
  DATA ALLT; SET ALLT T&T(IN=LATEST);
  IF LATEST THEN ITERAT=&T; RUN;
  %END;
%END TTEST;
*************i
%MACRO CONTROL;
***************;
%DO T=1 %TO 99;
  %IF FIRST= %THEN %LET FIRST=YES;
  %ELSE %LET FIRST=NO;
  %TTEST;
%END;
***TEST FOR CONTINUOUS, CAPTURE MIN & MAX;***
DATA _NULL_; SET ALLT END=EOF;
  RETAIN MIN 100 MAX 0 LAST 0;
  IF _N_=1 THEN LAST=ITERAT;
  ELSE DO;
    IF ITERAT NE LAST+1 THEN DO;
      CALL SYMPUT('CONTIN','NO');
      STOP;
    END;
    ELSE LAST=ITERAT;
  END;
BEGINNING TUTORIALS

IF ITERAT<MIN THEN MIN=ITERAT;
IF ITERAT>MAX THEN MAX=ITERAT;
IF EOF THEN DO;
   CALL SYMPUT('MIN',MIN);
   CALL SYMPUT('MAX',MAX);
END; RUN;

%MEND REPORT;
%MEND CONTROL;
*************
%MACRO REPORT;
%IF "&CONTIN"="NO"
   %THEN %LET TITLE=DISCONTINUOUS;
%ELSE %LET
   TITLE=CONTINUOUS (MIN IS &MIN, MAX IS &MAX);
PROC PRINT DATA=ALLT LABEL;
   LABEL T='T STATISTIC'
   PVALUE='PVALUE'
   ITERAT='ITERATION';
VAR ITERAT T PVALUE;
   TITLE "THE DISTRIBUTION IS &TITLE"; RUN;
%MEND REPORT;
*************
%CONTROL;

SUGGESTED READING

The macro language has proven exceedingly popular and any of the SAS user conferences' proceedings will most likely contain several papers on this topic. A very short list might include the following:


Earl R. Westerlund, "SAS Macro Language Features for Application Development," to be presented at the NorthEast SAS Users Group's Fifth Annual Conference, Baltimore, MD, October 4-6, 1992.

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