

%MACRO Bag of Tricks

Rodger Madison, RAND Corporation, Santa Monica, CA

ABSTRACT

Using an example macro this paper covers a variety of generally useful techniques including arrays of macro variables and the SAS® autocall library facility.

INTRODUCTION

This paper covers several coding techniques in the SAS macro language. These techniques are presented in the context of an example SAS macro that performs basic data manipulations using code generated by the macro. The paper assumes the reader has a basic understanding of the SAS language and SAS macro syntax.

%BAGOFTRICKS

It is often desirable to have a SAS program perform the same task on subsets of the input SAS data set, for example using BY or WHERE statements to control processing. The SAS macro language allows you to perform more complicated tasks, involving several steps, on subsets of data. One way of accomplishing this is through the use of arrays of macro variables. The following macro runs a simple analysis on subsets of the input data set defined by values of one of the input variables. The input data set is a hypothetical inventory file with two variables, PART, name of the part, and PRICE, retail price in dollars. There are multiple observations for each PART representing different brands of the part. When the macro executes it creates a list of all parts and from that list generates an array of macro variables, PART1, PART2, ..., PARTn, with the value of each variable being the name of one part in the input data set. The macro next runs a PROC MEANS to generate simple statistics on PRICE for each part and creates a separate Excel file, containing the statistics, for each part.

```
%MACRO BAGOTRICKS(INDATA);

/*
  Generate a list of parts in the input data set.
*/
PROC FREQ DATA=&INDATA;
  TABLES PART / OUT=PARTSLIST NOPRINT;
RUN;

/*
  This DATA step makes a macro array using the parts list
  as input. macro variables PART1, PART2, ... are created
  with the value of each variable being the name of a
  different part.
*/
DATA _NULL_;
  SET PARTSLIST END=LASTONE;

  VARNAME = "PART" || LEFT(PUT(_N_, 3.));

  CALL SYMPUT(VARNAME, PART);

  IF LASTONE THEN CALL SYMPUT("NUMPARTS", PUT(_N_, 3.));

RUN;

/*
  Once we have the array defined we can use it for a
  number of purposes. In this example we generate a series
  of Excel files containing summary statistics for each
  part.
*/
%DO CTR = 1 %TO &NUMPARTS;
```

```

PROC MEANS DATA=&INDATA NWAY NOPRINT;
  WHERE PART = "&&PART&CTR";
  CLASS PART;
  VAR PRICE;
  OUTPUT OUT=STATS;
RUN;

```

```

/*
  Note that I need three periods to
  create a proper Excel file name.
*/
ods phtml file="&&PART&CTR...xls";
PROC PRINT DATA=STATS;
  ID PART;
  VAR _STAT_ PRICE;
  TITLE "&&PART&CTR Statistics";
RUN;
ods phtml close;

%END;

```

```
%MEND BAGOTRICKS;
```

SYMPUT is used to create the array of macro variables. Note that the variable VARNAME contains the name of each variable, created by concatenating "PART" with the observation number. When the array is used in the subsequent %DO loop a double ampersand is needed to have the macro compiler resolve the name correctly. Also note that there are three periods in the Excel file name, since one period, serving as the delimiter of the macro variable, is removed with each pass of the code through the compiler.

The execution of %BAGOTRICKS produces an Excel file for each separate value of PART. However, in cases where the length of the part name is shorter than the longest value, blanks will appear in the Excel file name and also in the printed output of the macro:

```

Power Saws.xls
Nuts          .xls

```

while the TITLE of the printed output is:

```
Nuts    Statistics
```

%NEWTRICKS

The NEWTRICKS macro is a simple variation of BAGOTRICKS. The only different is the use of the %TRIM function to remove trailing blanks from the resolved values of the macro array variables so the Excel file names and TITLES are more aesthetically appealing. The %TRIM function is one of a number of macro functions that are supplied by SAS and referenced (in standard installations) by the default autocall definition. In most cases the SAS supplied autocall library is available without making a specific reference in the SAS code (the library is defined in the SAS configuration file

```

/*
  This works OK but I don't like the spaces in the Excel file names.
  I can drop these by using the %TRIM macro function.
*/
%MACRO NEWTRICKS(INDATA);

```

```

PROC FREQ DATA=&INDATA;
  TABLES PART / OUT=PARTSLIST NOPRINT;
RUN;

```

```

DATA _NULL_;
  SET PARTSLIST END=LASTONE;

  VARNAME = "PART" || LEFT(PUT(_N_, 3.));

  CALL SYMPUT(VARNAME, PART);

  IF LASTONE THEN CALL SYMPUT("NUMPARTS", PUT(_N_, 3.));

RUN;

```

```

%DO CTR = 1 %TO &NUMPARTS;

  PROC MEANS DATA=&INDATA NWAY NOPRINT;
    WHERE PART = "&&PART&CTR";
    CLASS PART;
    VAR PRICE;
    OUTPUT OUT=STATS;
  RUN;

```

```

/*
  Use the %TRIM function to drop trailing blanks
  from values in the macro array. Note that I
  now only need a single period in the Excel file
  name since the right paren of the macro call
  terminates the value.
*/

```

```

*/
ods phtml file="%TRIM(&&PART&CTR).xls";
PROC PRINT DATA=STATS;
  ID PART;
  VAR _STAT_ PRICE;
  TITLE "%TRIM(&&PART&CTR) Statistics";
RUN;
ods phtml close;

```

```

%END;

```

```

%MEND NEWTRICKS;

```

%MORETRICKS

The NEWTRICKS macro solves a problem in the original code by accessing a library containing SAS supplied macros. The SAS autocall library facility allows you to place additional libraries, containing your own macros, in the autocall search path. The MORETRICK macro calls two user written macros stored in such a library. The macros in this example are simple utilities that count observations and check the results of a MERGE. More important than their actual function is the mechanism by which they are called, which involves concatenating a user library with one of the SAS autocall libraries:

```

OPTIONS SASAUTOS=("!sasroot\core\sasmacro", "D:\rodger\RAND\sas\automacs");

```

```

%MACRO MORETRICKS(INDATA);

```

```

PROC FREQ DATA=&INDATA;
  TABLES PART / OUT=PARTSLIST NOPRINT;
RUN;

```

```

DATA _NULL_;
  SET PARTSLIST END=LASTONE;

```

```

VARNAME = "PART" || LEFT(PUT(_N_, 3.));

CALL SYMPUT(VARNAME, PART);

IF LASTONE THEN CALL SYMPUT("NUMPARTS", PUT(_N_, 3.));

RUN;

%DO CTR = 1 %TO &NUMPARTS;

PROC MEANS DATA=&INDATA NWAY NOPRINT;
  WHERE PART = "&&PART&CTR";
  CLASS PART;
  VAR PRICE;
  OUTPUT OUT=STATS;
RUN;

/*
  Here I merge a file that tells me how
  many different brands of each type of
  part I carry. I also check to make sure
  that each part is in the brands data set.
  The %MERGFLAG and %NUMOBS macros are defined
  in the autocall library.
*/
DATA STATS;
  MERGE STATS(IN=INS)
    BRANDS(IN=INB);
  BY PART;

/*
  The MERGFLAG macro creates a variable
  named FLAG indicating the result of the
  MERGE. A value of 0 indicates the
  record was found in both input files.
*/
%MERGFLAG(INS, INB)

  IF FLAG = 0 THEN OUTPUT;

RUN;

/*
  NUMBOBS returns the number of observations
  in the input data set and places that value
  into macro variable NSTAT.
*/
%NUMOBS(STATS, NSTAT)

/*
  If NSTAT is 0, the BRANDS file does not have a record
  for this part.
*/
%IF &NSTAT NE 0 %THEN %DO;

  ods phtml file="%TRIM(&&PART&CTR).xls";

```

```

PROC PRINT DATA=STATS;
  ID PART;
  VAR _STAT_ PRICE;
  TITLE "%TRIM(&&PART&CTR) Statistics";
RUN;
ods phtml close;

%END;
%ELSE %PUT "No BRAND file entry for %TRIM(&&PART&CTR)";

%END;

%MEND MORETRICKS;

```

The OPTIONS statement places the user library D:\rodger\RAND\sas\automacs after (one of) the libraries defined in the SAS configuration file (there are several such libraries in the standard configuration). It would also be possible to reverse the order:

```
SASAUTOS=("D:\rodger\RAND\sas\automacs", "!sasroot\core\sasmacro");
```

In which case, your macros would be searched first and would be chosen if there were identically named macros in both libraries. You may also include several of your own libraries:

```
SASAUTOS=("D:\rodger\RAND\sas\automacs", "D:\rodger\RAND\sas\more.automacs", "!sasroot\core\sasmacro");
```

or replace the SAS supplied library entirely:

```
SASAUTOS=("D:\rodger\RAND\sas\automacs", "D:\rodger\RAND\sas\more.automacs");
```

ANOTHER TRICK

When the SAS macro facility encounters a %MACRO statement in a program it compiles the code between that statement and a following %MEND statement and executes the compiled code only when the macro is reference further on in the program. This behavior allows you to create "super comments" by the simple device of enclosing code you wish to ignore between a %MACRO and corresponding %MEND, but never calling the macro. For example:

```
%MACRO CMT;
```

```
* Unwanted code.;
```

```
%MEND CMT;
```

This trick will allow you to ignore code that may have SAS syntax errors, but will macro syntax errors and unbalanced quotes will still create problems.

CONCLUSION

The SAS macro facility offers a variety of built-in tools and allows, using the autocall library facility, a programmer to extend and customize the language for their specific application. Tools in the autocall library can be as simple or complex as the application demands.

The code in this paper was run using SAS 9.1.3 under Microsoft Windows® 2000 and also using SAS 9.1.3 under Red Hat® Linux release 7.3.

CONTACT INFORMATION

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

Rodger Madison
 RAND Corporation
 1776 Main Street
 Santa Monica, CA 9040-3208
 Phone: 310.393.0411 X7616
 Email: rodger@rand.org

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.