An Introduction to the SAS® Macro Language

Gary L. Katsanis, Strong Memorial Hospital

INTRODUCTION

In the language of the SAS Guide to Macro Processing, Version 6, Second Edition, the macro facility is "a tool for extending and customizing the SAS System and for reducing the amount of text you must enter to do common tasks." This is perfectly true, but it is so general that those who are not familiar with macro processing may not understand how macros can help them. This paper will provide an introduction to macro processing based on uses and real-world applications with the goal of demonstrating how macros can help you to produce better, more efficient, and more effective applications.

The macro facility can be thought of as a programming system. You are all familiar with the DATA Step: a tool for manipulating data files. In an analogous way, the macro facility is a tool for manipulating something: SAS source statements. The macro facility provides you with a flexible and powerful programming language for manipulating the text of your SAS programs - your SAS source statements. The macro facility allows you to work with two new objects: macro variables (identified by the ampersand &) and macros (identified by the percent sign %).

MACRO VARIABLES

A macro variable is a very simple thing. It is a string of text identified by a name. You create a macro variable by using a command and you use it by typing its name where you want the string of text to appear. The same macro variable can be used over and over again once it is defined.

For example, lets say you have a master file called MASTER containing information on clients that is updated periodically and that you keep a copy of the data prior to the last update in a file called OLDMSTR. You may want to run a report program against either the new file or the old file. You might approach this by creating a macro variable called dset and coding the following statement:

%let dset = MASTER ;

Now, you can refer to the macro variable in your code, instead of using the actual data set name; that is, you can code:

PROC PRINT DATA = &dset ;

The macro processor will substitute MASTER for &dset before it runs the source statement, so your program will behave as if you coded:

PROC PRINT DATA = MASTER ;

At this point, we don't have any significant savings. As a matter of fact, it's taken MORE statements to perform a simple PROC PRINT. The efficiency of using macros in this sense is that you can use the macro variable &dset in many places, and you can change your source code in ALL of those places by changing the value of &dset in one place; see Figure 1. This is a very common use of macro variables by themselves: writing an extensive program and then customizing it for use on various data files by use of macro variables. Macro variables can be very long and are limited only by memory considerations. For a general discussion of these techniques, see Septoff, 88.

FIGURE 1:
Macros
A macro is a more complicated object than a macro variable. Like a macro variable, the end result of a macro is text that is included as a part of your SAS program, but unlike a macro variable, macros include executable statements that can manipulate information and perform logical control. Macros and executable statements that are a part of the macro processing language are all identified by the percent character %. The %let statement mentioned above is actually an executable macro statement that can appear anywhere in a SAS program, not just in a macro.

Defining and Calling Macros
You define a macro by coding %macro as the first statement and %mend as the last statement. You give the macro a name on the first statement and call it later by using that name. For example, you could define a macro called %dset that does the same thing as the &dset variable mentioned above. One way you could do this is as follows:

```sas
%macro dset ;
  MASTER
%mend;
```

A statement using this macro to identify the file for a PROC PRINT could be:

```sas
PROC PRINT DATA = %dset ;
```

and the result would be the same as the previous example. You can add flexibility to this by defining macro parameters. Macro parameters are simply macro variables that are available within the macro and nowhere else. For instance, you could define %dset as follows:

```sas
%macro dset(file) ;
  &file
%mend;
```

In this example, "file" on the command line is a macro parameter: a variable that exists inside the macro only and is deleted when the macro ends. To run this macro, you would code:

```sas
PROC PRINT DATA = %dset(MASTER) ;
```

The macro processor would call up the %dset macro, define the &file parameter within it, and substitute the value MASTER. In this case, the &dset macro variable and the %dset macro give identical values when you use them. Moreover, the second example is much longer to type in than just typing "MASTER". Given this, you may ask why you should use macros rather than just macro variables.

Functions and Other Macro Statements
One thing that you can do in a macro that you cannot do with a macro variable is to perform character manipulation and calculation. For instance, titles are often centered on a page; while a TITLE statement will center text, a PUT statement will not. You can get around this by creating a macro. For example:

```sas
%macro center(text) ;
  %let space = %eval(132-%length(text));
  %let indent = %eval(&space/2) ;
  @&indent "&text"
%mend;
```

This example does several things. First of all, the %macro statement creates a temporary macro variable called &text; next it calculates values for &space using the %eval function (which treats its arguments as numbers - unless you use %eval, the macro processor will treat all information as character data). Finally, it calculates &indent and then uses &indent and &text to place the parameter in the center of a page using PUT statement syntax. If you called this by coding:

```sas
PUT %center(This is a Title) ;
```

the result would be:

```sas
PUT @58 "This is a Title" ;
```

which would be centered on a 132-column page. You could generalize this by using a second parameter for the page width as well.

Macro Processing
Macro processing is fairly involved; good references exist in the SAS Guide, and in such places as Hendren (89). Some aspects should receive special notice, though.

* Macro processing is triggered by the characters & and %.
* When a & is encountered, the macro processor will search its macro variable tables for the value of the variable in the string im-
Beginning Tutorials

mediately following. That value is substituted for the reference to the macro variable.

* When a % is encountered, the macro processor will interpret the following characters as a macro you defined or as a macro function like %let or %eval. Don’t name your macros with the same name as a macro function! When a reference to a macro is found, the macro will be run and the result will be inserted in place of the reference. When a macro function is called, the function will be executed, and the result (if any) substituted for the original reference.

* After the macro processor has interpreted the value of a macro variable, a macro, or a macro function, the result will be scanned again until no more references with & or % exist. The final result is treated as if it were a part of your original SAS program.

Some macros and macro functions may leave code to be executed by the SAS compiler, others like %let leave nothing to be executed but rather perform some utility function. The only line in the %center macro that leaves anything for the SAS compiler is the fourth line, which uses %indent and %text to place the title you specified in the center of a page. For a detailed discussion, please refer to the references mentioned above.

Other Concepts

Two other major concepts in macro processing are use of the %if statement and the %do statement. These are often used together. The %if statement uses a condition and an action. If the condition is true, the action will be processed as macro code, otherwise the action will be skipped. An %else statement exists as well. For example, you might create the following macro:

```sas
%macro varlist(oldnew);
  %if %upcase(&oldnew) = MASTER
    %then VAR1 - VAR3;
  %else OLDVAR1 - OLDVAR3;
%mend;
```

Now, %varlist(MASTER) would resolve to "VAR1 - VAR3" and anything else would resolve to "OLDVAR1 - OLDVAR3".

The %do statement is coupled with an %end statement, and when you use them in a macro, all lines between the two statements are treated as a unit. For example, you might code:

```sas
%macro printn(oldnew);
  %if &oldnew = MASTER
    PROC PRINT DATA = &oldnew;
    VAR %varlist(&oldnew);
  %end;
%mend;
```

Now, if you coded "%printn(MASTER)" , the macro would resolve to:

```sas
PROC PRINT DATA=MASTER;
VAR VAR1 - VAR3;
```

Note the reference to another macro, %varlist. When the macro processor reaches references like %varlist, it will put the first macro on hold, process the new macro in its entirety, and use the result as part of the original macro.

SUMMARY AND EXAMPLES

At this point, we’ve seen some information about macro variables, macros, and macro functions, as well as their uses and utility in some sections. These techniques can be used as a significant time saver during program development, can minimize the effort to support an application, and can provide a level of flexibility and control that is not available elsewhere in the SAS System.

Creating a List of All Files In a SAS Library

An illustration can be found in the following application. In this application, data sets are created interactively and saved in a permanent SAS Library called INLIB. On occasion, the data sets have to be concatenated through a SET statement, requiring a list of the files. For example, if there were 3 files called DATA1, DS2, and RECORD, we would need the SET statement: "SET DATA1 DS2 RECORD;".

To deal with this, we developed a macro that ran PROC CONTENTS and read the output data set, saving the result in a series of macro variables, which were then inserted into our SAS program. The macro is presented in Figure 2.
FIGURE 2: Figure 2 - Macro to Read All Members in a Library

```sas
OPTIONS NOSOURCE;
PROC CONTENTS DATA=INLIB._ALL_ OUT=MEMLIST; creates a file with the names of all members
RUN;
in the library "INLIB"
DATA _NULL_; pulls in the member list
LENGTH MVNAME $8;
SET MEMLIST(KEEP=MEMNAME) END=LASTONE; MVNAME will be "ds1" on the first
MVNAME = "ds" || LEFT(PUT(_N_),2.)); observation, "ds2" on the second, etc.
CALL SYMPUT(MVNAME,MEMNAME); saves the value of MEMNAME in macro variable
end named by value of MVNAME
IF LASTONE THEN CALL SYMPUT("numds",_N_); saves number of members into macro variable &numds
RUN;
%macro memlist; executes loop once for each member
%do i = 1 to &numds;
&&ds&i
%end;
%mend;
```

Note that this macro uses some techniques that we have not seen yet. In the DATA Step, you see a CALL SYMPUT statement. This statement creates a macro variable or sets the value of a macro variable based on the value of a variable in the SAS data set. You can find more about SYMPUT and SYMGET (which loads the value of a macro variable into a SAS data set variable) in Chapter 6 of the SAS Guide.

You also see an example of an iterative %do loop. In this case, we want to repeat a statement several times, as determined by the value of the macro variable %numds. As in the similar DATA Step statement, the counter (in this case %i) is initialized to 1, then incremented automatically; and all statements between the %do and the %end statement are executed until the counter reaches the value %numds.

You also see an example of building the name of a macro variable by use of the values of another macro variable. Consider the string "&&ds&i". A rule of macro processing states that two ampersands && will always resolve to a single ampersand, &. When &&ds&i is seen in the first pass through the %do loop, &i will be 1. The two leading ampersands will resolve to a single ampersand; ds (which is not a macro variable) will be untouched, and %i will resolve to 1. The result is &ds1 which itself is a macro variable. The macro processor will recognize this and resolve &ds1 again, substituting its value, DATA1.

Tailoring a Program to Run on a Specified Date Range

You may have found yourself in the situation where you want to run the same program over and over again, with the only difference being in a calendar date range used to select a subset of your data. The macro below outlines an approach to dealing with this. A character string is passed to it, consisting first of "M" for month, "Q" for quarter, and "Y" for year, followed by a number for the quarter or month, and the last two digits of the year. For example, to refer to March 1988, you would specify "M 3 88". You can also specify a character string to be passed verbatim as a comment. This macro goes through several phases. First, it separates the input and checks validity. Next, it calculates values to identify the beginning and the end of the time period in SAS date format. It then uses a DATA_NULL_ and the WORDDATE format to translate the SAS date value into a text string. Finally, it presents the values of the macro variables as created. Refer to Figure 3 at the end for more details.
SOME OTHER ISSUES

Semicolons present a problem in assigning macro variables: since a semicolon is treated as the end of a macro function line, you cannot just imbed a semicolon in the middle of a string you would like to define as a macro variable.

A way to deal with this situation is to use the quoting functions. These are documented in Chapter 10 of the SAS Guide; all quoting functions share a common attribute: they treat special characters like & , % and ; as if they had no special meaning, or they restore the special meaning to those characters. For example, you might want to assign a macro variable to several lines of SAS code. The statement:

```sas
%let printn = PROC PRINT DATA=&dset;
   VAR NAME VAR1 - VAR3;;
```

would create an error, since the ";" after &dset would be considered as the end of the %let statement. The right way to do this is to code:

```sas
%let printn = %str(PROC PRINT DATA=&dset;
   VAR NAME VAR1 - VAR3);
```

The %str function removes the special significance of the semicolons and the entire line, including semicolons, is stored in the variable called &printn.

There are a series of automatic macro variables that are created for your use. These are documented at the end of Chapter 2 in the SAS Guide. The automatic variables provide information about things like the date, time, environment (interactive or not), status after the most recent command, and so forth. One variable, &SYSPARM, can have its value set outside of SAS processing in an OPTIONS statement; useful for tailoring a noninteractive or batch program.

Several SAS options can be used with the macro facility (beyond SYSPARM mentioned above). The MACRO option turns macro processing on and off; make sure you specify "MACRO" before you try to use the macro facility! The options MPRINT (which prints the SAS code ultimately generated by a macro), MTRACE (which prints the macro statements as they execute), and SYMBOLGEN (which prints the resolved values of macro variables) are the Version 6 options for macro diagnostics. These are documented in the System Options section of the SAS Language Reference, Version 6, First Edition and also in Appendix 1 of the SAS Guide to Macro Processing, Version 6, Second Edition.

Be careful about timing in macro applications. For example, a DATA Step is compiled before it executes. Macro variables created by SYMPUT are not available until after the DATA Step is executed, so you cannot use macro variables created in a DATA Step in the same DATA Step. Other timing issues exist as well; see Hendren (89) and Chapters 8 and 9 of the SAS Guide.

Bibliography

Note: this covers only a few of the many articles written about macros and presented in NESUG and SUGI over the years. I have limited this to references written in the last three years and referred to above or found to be particularly useful.

I would like to thank the management of the Office of Clinical Practice Evaluation for their unstinting support, and the staff, especially Daniel Gronell and Keith Skelton, for many valuable suggestions.


SAS is a registered trademark of SAS Institute Inc, Cary NC, USA.
FIGURE 3:  
OPTIONS NOSOURCE;

%macro date(parm);

%global pdstart pdend start end title year;
%local period position month emonth eyear not;

%let pdstart = ;
%let pdend = ;
%let pdtitle = ;
%let period = %upcase(%scan(&parm,1));
%let position = %scan(&parm,2);
%let year = %scan(&parm,3);
%let title = %scan(&parm,4);
%let eyear = &year;

%if %length(&period) ne 1 %then %goto bomb;
%if %index(MQY,&PERIOD) = 0 %then %goto bomb;
%if %eval(&position+O) > 12 %then %goto bomb;
%if %eval(&position+O) < 1 %then %goto bomb;
%if %eval(&year+O) > 98 %then %goto bomb;
%if %eval(&year+O) < 1 %then %goto bomb;

%if &period eq M %then %do;
  %let month = &position;
  %let emonth = %eval(&position+1);
  %if &emonth = 13 %then %do;
    %let emonth = 1;
    %let eyear = %eval(&year+1);
  %end;
%end;

%else %if &period eq Q %then %do;
  %let month = %eval((&position-1)*3+1);
  %let emonth = %eval(&month+3);
  %if &emonth = 13 %then %do;
    %let emonth = 1;
    %let eyear = %eval(&year+1);
  %end;
%end;

%else %do;
  %let month = &position;
  %let emonth = &position;
  %let eyear = %eval(&year+1);
%end;

NFIGURE 3 - Macro for Date Ranges

%macro date(parm);

%global pdstart pdend start end title year;
%local period position month emonth eyear not;

%let pdstart = ;
%let pdend = ;
%let pdtitle = ;
%let period = %upcase(%scan(&parm,1));
%let position = %scan(&parm,2);
%let year = %scan(&parm,3);
%let title = %scan(&parm,4);
%let eyear = &year;

%if %length(&period) ne 1 %then %goto bomb;
%if %index(MQY,&PERIOD) = 0 %then %goto bomb;
%if %eval(&position+O) > 12 %then %goto bomb;
%if %eval(&position+O) < 1 %then %goto bomb;
%if %eval(&year+O) > 98 %then %goto bomb;
%if %eval(&year+O) < 1 %then %goto bomb;

%if &period eq M %then %do;
  %let month = &position;
  %let emonth = %eval(&position+1);
  %if &emonth = 13 %then %do;
    %let emonth = 1;
    %let eyear = %eval(&year+1);
  %end;
%end;

%else %if &period eq Q %then %do;
  %let month = %eval((&position-1)*3+1);
  %let emonth = %eval(&month+3);
  %if &emonth = 13 %then %do;
    %let emonth = 1;
    %let eyear = %eval(&year+1);
  %end;
%end;

%else %do;
  %let month = &position;
  %let emonth = &position;
  %let eyear = %eval(&year+1);
%end;
FIGURE 3 (cont):

Figure 3 - Macro for Date Ranges -- Continued

```sas
%let pdstart = MDYstr('%()'&month,01,&year%str('%)) ; define start and
date constants
%let pdend = MDYstr('%()'&emonth,01,&eyear%str('%)-1) ; end as SAS
use DATA Step to create
DATA _NULL_; start and end as
data constants
    SDATE = TRIM(PUT(&PDSTART,WORDDATE18.));
    EDATE = TRIM(PUT(&PDEND,WORDDATE18.));
    CALL SYMPUT('START',SDATE);
    CALL SYMPUT('END',EDATE);
    STOP;
    RUN;

%goto quit ;

%bomb:
    %put Error in date input ;
    %let not = NOT ;

%quit:
    %put Date values for &period &month &year &not set. ;
    %put PDSTART is &pdstart ;
    %put PDEND is &pdend ;
    %put START is &start ;
    %put END is &end ;
    %put TITLE is &title ;
    %put YEAR is &year ;

%mend date ;

OPTIONS SOURCE ;
```

NESUG '90 Proceedings
accurately size the CPU requirements needed for the larger ATM network. If we determine that we need an upgrade, we can go forward with the argument that we need a bigger CPU because 'the number of ATM machines is growing.' This argument is much more understandable to the financial executive.

Taking this example one step further, we can now be pro-active and more accurate in determining future CPU requirements. We can answer questions like 'What will happen if my ATM network grows by 50%?'

There are two keys to the process. The first is to find the 'business element' that you can associate with resource utilization. These generally fall into two categories:

* Number of Logged on Users or Logged on Devices
* Number of 'accounts' or 'items' in the database

Start by reviewing the application at a high level, much as a systems analyst would do. For example, if the process in question sequentially scans the database, then the number of accounts in the file will probably be the key. If the process in question has to do with trivial online transactions, then the number of logged on users/logged on devices is a good candidate to study.

Remember that correlations can change over time. They need to be periodically reviewed and verified.

The second key to Application Level Capacity Planning is in summarizing all of the resources used by the business unit. In a dedicated environment, this is very simple. For example, if you have a MVS system (either physical or logical) that supports only one application, then you can simply summarize the RMF type 70 records to get an extremely accurate resource picture. Unfortunately, this is a rare occurrence.

The next level down is when you can associate applications to specific MVS address spaces. For example, in a CICS MRO environment, some installations have isolated applications into individual Application Owning Regions (AOR). If this is the case in your environment, you can differentiate resource usage by using either the RMF type 72 records or SMF type 30.

Although this is fairly accurate, (you will lose some overhead items such as VTAM, DB2 locking, and MVS paging), there are two significant drawbacks with this approach. First, it is a maintenance headache for the analyst. If you fail to get a new address space defined in the IEAICS member of PARMLIB, you will effectively lose the information. More importantly, if the decision is ever made to combine two applications together into the CICS AOR or the IMS MPR, you have no method to adjust or recover.

A third method for capturing application level resource usage is to collect resource statistics from each of the system functions used by the application. These sources are:

- batch - use SMF type 30. Identification of the application should be able to be done by review of the jobname or the job accounting.
- TSO - use SMF type 30. Identification of the application should be able to be done by review of the jobname or the job accounting.
- CICS - use SMF type 110. Identification of the application can be done by associating different transaction types with the different applications.
- DB2 - use SMF type 100 & 101. Identification of the application can be done using either the plan name, the authorization id, or the correlation id. Plan name is best because it will apply for all types of attaches.
- IMS - use the IMS log to extract resource usage by transaction code. Each transaction code can then be rolled up into an application like CICS.

The key is to have good standards. If the application acronym is imbedded in the first three characters of the DB2 plan, jobname, TSO userid, and the IMS transaction codes, then you can write simple SAS code to extract the application id. CICS transaction codes are more difficult to associate with an application because they are only four digits long. Either a set of SAS "IF" statements or a SAS format, however, can accomplish the task.

One word of warning. Because you will be using different sources of instrumentation, you need to be careful not to double count CPU burn. For example, in the case of a CICS attach to DB2, you will have a type 30 record for the CICS address space that represents both CICS and DB2 user resources, a type 110 record that represents CICS, and a type 100/101 record for DB2. If you add all of the CPU resources