SAS® Macro Language Features for Application Development

Earl R. Westerlund, Eastman Kodak Company

Abstract
The SAS® macro language has many uses. One use is to facilitate development of applications and systems. This talk will discuss some features of the SAS macro language that are useful to application developers. Topics to be discussed include:

- Application development features such as windows, statement style macros, and the autocall facility
- The hows, whys and whens of macro quoting
- Options and statements that aid in macro debugging.

This talk assumes a basic knowledge of SAS macros and macro variables. The opinions expressed in this paper are those of the author, and not representative of Eastman Kodak Company.

Introduction
The SAS macro language is a very useful tool. Like all tools, however, they can be misused. Before we begin to discuss features of the macro language, it might be good to review what the macro language is good for, and other tools that are available to application developers.

When to use macros
The SAS Guide to Macro Processing, Version 6 Edition defines the macro facility as a "tool for extending and customizing the SAS System and for reducing the amount of text you must enter to do common tasks." The macro language is best used if it is used:

- For programs that perform repetitive tasks, such as processing a number of similar SAS data sets in the same way
- For conditional execution of SAS code; for example, executing particular procedures based on the type of data set or desires of the user
- For "packaging" blocks of reusable SAS code for easy retrieval at a later time.

When NOT to use macros
The macro language is so powerful and flexible that when people first learn it, they are sometimes so impressed to the point that they overuse it. Do not use the macro language:

- "Because it's there." I have seen SAS programs which were one big macro definition, followed by the macro invocation, with no conditional or repetitive execution of code. Remember that the macro facility must compile the macro, then execute it before the SAS System even sees it. In other words, it adds overhead to your program.
- If the only time the macro would be used would be in a batch program and, again, there is no conditional or repetitive execution of code. The %INCLUDE statement uses less overhead because it reads in and immediately executes the code. It does not have to compile a macro first.
- Simply for interaction with the end user. There are times when it is appropriate, but there are a number of other ways to interact with the user.

Other development tools
There are a number of other application development tools that are available with the SAS System. They can be used with the macro language when appropriate:

For program control:

- SAS/AF® is a product for development of windowed application systems. Its Screen Control Language was developed for user interaction, and works much better than the macro facility for that.
- SAS/ASSIST® is a SAS program product, written in SAS/AF, that allows users who don't know the SAS language to "point and click" their way through the system. The EIS (Executive Information System) option allows you to add your applications to the ASSIST menus. If your users are going to use...
ASSIST, consider adding your applications to its menus.

* PROC PMENU, in the Base SAS product, allows you to build pull-down menus that can be added to any windows you build.

For data entry and manipulation:

* The SAS programming language (the DATA step) contains many tools for application development. For simple user interaction, the PUT/INPUT and WINDOW/DISPLAY statement combinations are available. In Release 6.03 and beyond, DATA steps can be stored in compiled form, saving some overhead. As of release 6.06, compiled macros still cannot be stored.

* PROC FSEDIT and PROC FSVIEW, in SAS/FSP®, are procedures which allow data entry and editing. Screen Control Language (SCL) is available to customize screens.

Refer to Attachment A for an overview of how the products and features work together.

Application development features

Enough of discussing why you shouldn't use the macro language. I promised to discuss what's available for you to use in the macro language. Following, in no particular order, are some features that are useful to know when developing applications in the macro language.

For interaction with users

* The %PUT and %INPUT statements provide a quick, simple way to interact with the user in line mode. Here's an example of how they can be used:

    %macro which;
    %put Which data set do you want to print?;
    %input dsname;
    PROC PRINT DATA=&dsname;
    RUN;
    %mend which;

    The big drawback is that the %PUT statement will only output to the SAS log. Your log must be directed to the terminal, which means that either your users will see lots of information they don't need, or you lose potentially valuable information about the execution of the job.

* The %WINDOW and %DISPLAY statements build and then display windows for interaction. Here's the same example as above, except using a window:

    %macro which;
    %window whichds
    #3 @5 'Which data set do you want to print?';
    #5 @10 dsname 17 required=yes;
    %display whichds;
    PROC PRINT DATA=&dsname;
    RUN;
    %mend which;

    Upon execution, a window opens with the displayed text, and waits for the required input. This is a simple process. The primary disadvantage is that there are few built-in functions for error checking. Such functions exist in SCL.

Statement-Style Macros

By using the STMT option in a macro definition, you can set up macros so that they look like SAS statements. Consider the following macro, which performs a standard analysis on a specified data set.

    %macro analyze(dsname) / stmt;
    PROC MEANS DATA=&dsname;
    RUN;
    PROC UNIVARIATE DATA=&dsname;
    RUN;
    %mend analyze;

    To run this macro, you simply invoke it as if it were a SAS statement:

    ANALYZE MYLIB.MYDS;

    With statement-style macros, you can integrate your macros into the SAS System. They look like other SAS statements; there is no apparently (to the user) artificial distinction between your macros and SAS commands. There are two disadvantages. First, the macro call must form a complete SAS statement; you cannot imbed a statement-style macro within another SAS statement. Second, the SAS system must do an extra check of the beginning of each statement to determine whether it is a statement-style macro. This adds to the overhead of running a program.
The Autocall Facility

The autocall facility is a way of making a library of commonly used SAS macros available without having to explicitly include them in each program in which they are used. When a macro is invoked and not yet defined, the autocall library is searched for the macro. If the macro is in the library, the SAS system reads the macro, compiles it and then runs it.

The form of the macro library depends on the operating system. On CMS, macros are stored as members of a CMS MACLIB. On MVS, they are members of a partitioned data set. The name of the member must be the same as the name of the macro.

Macro libraries that are to be available to the general public can be defined in the control program (EXEC, CLIST, etc.) that invokes SAS. They are defined by the SASAUTOS fileref.

You can add a macro library to the autocall list by using the SASAUTOS option. For example, to add the macro library MYMACS to the existing autocall library on CMS, use the statement:

```
OPTIONS SASAUTOS=(SASAUTOS 'MYMACS MACLIB *');
```

Data driven macros

The SAS System provides tools to facilitate data exchange between the macro facility and SAS data sets. It can be very useful to be able to run certain procedures based on the value of a variable in a SAS data set. You can use the CALL SYMPUT statement to accomplish this. For example, suppose you want to do different analyses depending on the value of the DATA TYPE variable in a data set. You could accomplish it this way:

```
%macro analyze(dname);
  DATA _NULL_;
  SET &dname(OBS=1);
  CALL SYMPUT("DTYPE",DATATYPE);
  %if &dtype = TYPEONE %then %analyze1;
  %else %if &dtyprp = TYPETWO %then %analyze2;
  %else %startanl;
%mend analyze;
```

When invoked, this macro will cause the macro variable &DTYPE to be assigned the value of the DATA_TYPE variable in the first observation of the data set. After the DATA step is done, the value has been assigned and is available for use. Based on the value of &DTYPE, one of three analysis macros is run. You can also use CALL SYMPUT to customize titles, select data set variables to be analyzed or, choose the type of analysis to be done based on a data value.

Transferring data from macro variables to SAS data set variables is easier. If the name of the macro variable is constant, you can simply issue an assignment statement in a SAS DATA step:

```
CHARVAR = "&macvar";
NUMVAR = &macvari;
```

If the name of the macro can change based on data value, you can use the SYMGET function. The following example assumes there are three macro variables &X1-&X3, the values of which are to be assigned into SAS variables based on the value of the variable TYPE in SAS data set INFO, which has values 1 to 3:

```
DATA NEW;
  SET INFO;
  XX = SYMGET( 'X' || LEFT(TYPE));
RUN;
```

You can use these features to transfer information from one step to another. The following example generates a plot containing the mean as a reference line. The mean and standard deviation are listed in a footnote.

```
* Step 1: get the mean;
PROC MEANS NOPRINT DATA=MYDATA;
  VAR Y;
  OUTPUT OUT=STATS
    MEAN=YBAR
    STD=STD;
RUN;
* Step 2: create macro variables;
DATA _NULL_;
  SET STATS;
  CALL SYMPUT ('MEAN', PUT(YBAR,6.2));
  CALL SYMPUT ('STDEV', PUT(STD,6.3));
RUN;
```
* Step 3: generate the plot with mean as ref line;
PROC GPLOT DATA=MYDATA;
  PLOT Y*TIME / VREF=&mean;
  TITLE 'Plot of Y Across Time';
FOOTNOTE
  "MEAN=&mean; ST.DEV.=&stdev";
RUN;

Macro quoting

Macro quoting is one of the most difficult concepts to grasp - even for an experienced macro user. Understanding why it is necessary goes a long way toward using it effectively.

When it is necessary

The macro language is strictly a character-based language. Its sole purpose is to generate fragments of SAS code. Even "numeric" macro variables are actually character variables. The problem occurs when you need to use a character that has special meaning (such as & or %) in a character string in such a way that the macro language might interpret it as part of a macro statement. For example, the following macro prints data about a company from a database. The company is identified by its stock ticker.

%macro company(ticker);
  %if bquote(&ticker) eq EK /* special for Kodak */
  %then PROC PRINT DATA=STATS.EK;
  %else %do;
    PROC PRINT DATA=STATS.OTHER;
    WHERE COMPANY="&ticker";
    RUN;
  %end;
%mend company;

There are two problems with the above macro. First, if we decide to get information on General Electric, we have a problem. General Electric's ticker symbol is GE, which also happens to be SAS's comparison operator for "greater than or equal to." The %if statement becomes

%if GE eq EK ...

which the macro language does not know how to interpret. Adding the %BQUOTE function takes care of this. It hides the special meaning of "GE" from the macro processor so the comparison can be made.

The second problem occurs in the third line. The semicolon at the end of the line "%then PROC PRINT DATA=STATS.EK;" closes the %if statement, not the PROC PRINT statement. Consequently, there is no semicolon after the PROC PRINT statement, which will cause an error. Using a %STR statement ensures that a semicolon is passed to the SAS language.

The corrected macro becomes:

%macro company(ticker);
  %if bquote(&ticker) eq EK
    %then %str(PROC PRINT DATA=STATS.EK);
  %else %do;
    PROC PRINT DATA=STATS.OTHER;
    WHERE COMPANY="&ticker";
    RUN;
  %end;
%mend company;

Kinds of macro quoting

There are two basic kinds of quoting functions. %STR and %NRSTR quote strings at macro compilation time. They prevent the macro language from treating your string as part of the macro language while the macro is being compiled. The difference between %STR and %NRSTR is that %STR will resolve macros and macro variables, while %NRSTR will treat them as text and not resolve them.

The second kind of quoting functions take effect at macro execution time. %BQUOTE, %NRBQUOTE and %SUPERQ quote the resolution of a macro value, to prevent situations where special characters and mnemonic operators (like "GE") that happen to appear in a string are interpreted as macro code. The difference between %BQUOTE and %NRBQUOTE are similar to %STR and %NRSTR: %BQUOTE resolves macros and macro variables before quoting, and %NRBQUOTE does not. %SUPERQ only takes a macro variable as an argument; otherwise, its function is pretty much the same as %NRBQUOTE.

Refer to the SAS Guide to Macro Processing for more details on macro quoting. It contains some useful tables describing which functions quote which special symbols, and when they take effect.
Options and statements for debugging

Using any application development facility effectively requires effective debugging tools. Version 6 of the SAS System provides some useful options for tracing the execution of your macros. These are SAS system options, and can be invoked at startup in the configuration file or on the command line, or can be changed during execution through the OPTIONS statement or options window. Here are the available options:

- The MPRINT option displays in the SAS log all SAS code that has been generated by the macro facility. Use this to verify that the code that you want to generate is in fact being generated.

- The SYMBOLGEN option displays the resolution of all macro variables. Use this to verify that your variables are resolving correctly.

- The MTRACE option traces the execution of a macro. Messages are sent to the log indicating the beginning of a macro, statements that are executed, whether %IF conditions are true or false, and when a macro finishes execution.

- The %PUT statement is not a system option, but it can be very useful in determining the source of a problem. Use it to print macro variable values to the log, or to print messages indicating that you've reached a certain point in the macro.

System Variables

There are a number of automatic macro variables that you can use to control macro execution or enhance output. Some of them are:

- &SYSDATE contains the date that the SAS job ran.

- &SYSTIME contains the time that the SAS job started.

- &SYSDEVICE contains the name of the graphics device specified by the DEVICE= option.

- &SYSDSN contains the name of the most recently created SAS data set.

- &SYSMSG contains a message that you want to display in the message area of a macro window. Use the %LET statement to set a value for &SYSMSG.

- &SYSERR contains the return code from the last DATA or PROC step.

- &SYSRC contains the return code from the last host system command issued from within a SAS session.

Conclusion

While SAS Institute has provided a number of new features and products to facilitate application development, the SAS macro language remains a very useful and powerful facility. Knowing more about the tools available to you can make your system development task, and your users' lives, much easier.

Acknowledgments

SAS, SAS/AF, SAS/ASSIST and SAS/FSP are registered trademarks of SAS Institute Inc., Cary NC, USA

References


SAS® Application Development Tools: Relationships

- SAS/ASSIST®
  - "EIS" option
  - Interface between application and ASSIST
  - Window interface to application
    - macro language - for code manipulation
    - DATA step - for data manipulation
    - FSEDIT / FSVIEW / FSBROWSE - for data editing, browsing, entry
    - PROCedures - for data analysis and reporting
    - PROC PMENU - to add pull-down menus to windows