The SAS Macro Facility is an extremely powerful tool, which is often not utilized because it is not understood. This Hands-On PC Workshop is designed to build on the knowledge base established in previous introductory Macro Workshops and Courses. Presenting in the form of an interactive tutorial, this workshop will delve deeper into the labyrinthine of the Macro Processor to help the user understand what is really going on. However, the potential attendee can rest assured that the most innermost Macro secrets will not be revealed.

Some prior experience with SAS Data step programming and Display Manager are assumed. At least 6 months experience with the Macro Facility is highly desirable, and several unresolved Macro problems are absolutely mandatory.

The SAS Macro Facility has now been available for almost 10 years. Unfortunately, the majority of SAS Users have yet to experience the benefits which accrue almost immediately from the utilization of this tool because they do not use it at all. Even more unfortunately, many who have attempted to use SAS Macros have become hopelessly confused by some of its more advanced constructs. In a paper presented at a previous SUGI conference, Kretzman [1] suggests that this is due to "a bewildering set of abstruse quoting functions, multiple ampersand constructions, and a syntax which is just similar enough to regular SAS programming to get you into trouble". Although I could not disagree with the preceding statement, I would suggest that much of the confusion relating to non-trivial aspects of the SAS Macro Facility is also directly related to the complexity and lack of clarity of the documentation presented and with the convoluted examples which are utilized to demonstrate advanced use of the SAS Macro Facility.

The benefits to be derived from the use of the SAS Macro Facility have been well documented in many previous SUGI tutorials and workshops, and need not be dwelt upon further. A basic familiarity with the SAS Macro Facility syntax and capabilities will be assumed. (A suggested reference for those just starting out with the Macro Facility is Tindall and O'Connor [2]). Those of you who are seeking a more complete understanding of the intricacies of input stack processing, the word scanner, the tokenizer, and the open code handler might wish to review Chapters 8 and 9 of the SAS Guide to Macro Processing [3] and O'Connor [4], because none of this information will be presented to confuse you at this workshop. It is the intention of this presentation to provide further insight into a few of the more confusing aspects of the Macro Facility, hopefully clarified by simple examples and interactive exercises.

Some of the SAS OPTIONS which are directly related to the use of the Macro Facility are described below.

1. MACRO
   Quite simply put, if OPTIONS MACRO is not set, the SAS Macro Facility may not be utilized. This option is normally set by the SAS Installation Representative, but can also be set by the user at SAS invocation (by specifying OPTIONS=(MACRO)). This option may not be set once you have started your SAS Session.

2. MAUTOSOURCE and SASAUTOS
   If it is desired to utilize the Macro Autocall Facility, OPTIONS MAUTOSOURCE must be set. The SASTAUTO Option, which is of the form
   
   OPTIONS SASAUTOS = (lib1, lib2, ..., libn)
   
   allows the specification of the Autocall Library. Each of the lib, parameters may be either a filename or a data set name. Both of these options may be set during your SAS Session.

   The objective of the Autocall Facility is to enable the automatic "inclusion" of the source code for a Macro Program at the first reference to that Macro Program name during the session. The net effect of the Autocall Facility is almost identical to that which would occur if the user
   
   %include'd the corresponding flat file at the first reference to the Macro Program name, but this process is performed automatically by SAS.

   Use of the Autocall Facility involves storing the source code for your SAS Macro Programs in flat files in one or more "aggregate storage locations". An "aggregate storage location" is just a fancy SAS multi-platform name for a computer file structure underneath which files are stored, for example, a partitioned dataset under MVS, a macilb under CMS, a directory or VMS text library under VMS, a library under VSE, and a directory under virtually all directory based systems. If your system supports file extensions, the file extension should be .SAS. If the Autocall Facility is active, the first reference to a SAS Macro Program which has not previously been compiled (that is, which is not in Catalog SASMACR in the WORK Library) causes the (possibly concatenated) Autocall Library to be opened and searched for a member file of the same name as the SAS Macro Program. If such a member is found, the file is read in; otherwise an error message is output on the LOG.
Unfortunately, the Autocall Facility allows you the opportunity of copying the following two poor programming practices demonstrated by SAS in their distributed Autocall Libraries:

1. Inclusion of non-Macro code in the flat file, and
2. Inclusion of more than one Macro Program in a single flat file.

Since all of the code in the flat file which is "included" by the Autocall Facility is processed, the techniques above do work. However, the non-Macro code is only executed the very first time that the Macro Program is invoked, and the storage of several Macro Programs in one flat file contributes significantly to maintenance problems (who can remember where Macro M_WHATEVER is stored?). I would strongly urge you not to put any non-Macro code in any of the Autocall Library Files, and to store each Macro Program in a separate file in this library.

The Autocall Facility does provide the site with the capability of storing Macro Programs in several logical libraries, which might be categorized as SAS Institute Macro Programs, Site Macro Programs, Department Macro Programs, and Personal Macro Programs, and may therefore avoid many arguments relating to file access and security. It has been my experience that the advantages gained by such a structure far outweigh the execution time overhead of opening several libraries to resolve Macro Program references, particularly if statement style Macro Programs (which should be avoided like the plague) are not used.

3. MSTORED and SASMSTORE

OPTIONS MSTORED must be set in order to utilize the Compiled Stored Macro Facility. The SASMSTORE Option, which is of the form

\[ \text{OPTIONS SASMSTORE = libref} \]

allows the specification of the Compiled Stored Macro Library. At the present time, and for the foreseeable future, all compiled stored macro programs are stored as entries in the SASMACR catalog in only one library, that is, concatenation of libraries is not allowed. Both of these options may be set during your SAS Session.

The objective of the Compiled Stored Macro Facility is the reduction of execution time by allowing previously compiled SAS Macro Programs to be accessed during the Session, thereby avoiding the overhead of compiling these Macro Programs. This technique is primarily applicable in a production environment, where it can yield impressive reductions in CPU time (particularly when large Macro Programs are involved).

The Compiled Stored Macro Facility allows you to specify (as an option in the %MACRO statement) that the compiled Macro Program is to be stored in the Compiled Stored Macro Library. An example of such a %MACRO statement appears below:

\[ \text{%MACRO M_MYMAC / STORE DES = 'Sample Macro Program';} \]

Obviously, you will have to compile each SAS Macro Program once to cause it to be stored.

**CAUTION:** If you decide to utilize the Compiled Stored Macro Facility, you are strongly urged to keep a copy (in a very very safe place) of the source code for each Macro Program stored in the Compiled Stored Macro Library. It is not possible to recreate the source code for the Macro Program from the compiled code. The compiled code is not platform independent, can not be ported from platform to platform, and is likely not to be version independent, that is, it is unlikely to work under SAS Release 7.xx.

Assuming that all of the appropriate Macro Options have been set, SAS searches for Macro Programs in the following order:

1. Session compiled Macro Programs,
2. Compiled stored Macro Programs, and
3. Autocall Macro Programs.

4. MPRINT, SYMBOLGEN and %PUT, and MLOGIC

Debugging SAS Macro Programs is a complex and difficult task. The primary problem facing the user is determining whether the bug is in the macro code or the SAS code emitted by the Macro Program. Although none of the techniques are foolproof, the game plan below may prove helpful.

1. Look at the error message(s) output on the LOG, and simply look at the source code for the Macro Program. It is truly amazing how many errors can be detected and resolved by this manual inspection technique.
2. If the error message(s) on the LOG appear to be standard SAS messages (as opposed to macro error messages), setting OPTIONS MPRINT may be useful. MPRINT causes all of the SAS code emitted by the Macro Program to be listed on the LOG. Unfortunately, the error message numbers which appeared under SAS Version 5 (and which clearly separated macro errors from other SAS errors) have been eliminated under Version 6, so it is not always evident where the error message originated.
3. If it appears that the problems are the result of incorrect branching in the Macro Program, it is likely that one or more symbolic macro variables contain a value other than that which was expected. Setting OPTIONS SYMBOLGEN will cause the resolved values of all symbolic macro variables to be output on the LOG. In addition, the judicious use of the %PUT statement to output text strings and/or symbolic macro variable values on the LOG often proves effective.
4. If all else fails, and the execution of the Macro Program is not as expected, OPTIONS MLOGIC may be appropriate. This option will cause the flow of Macro Program execution, evaluation of parameters, the value of macro conditions being evaluated (%IF), the value of %DO loop counters, and the beginning and end of each Macro Program being executed to be noted on the LOG.

Further details relating to the SAS Options associated with the use of the Macro Facility may be obtained by perusing the SAS Language: Reference [3], the SAS Guide to Macro Processing [3], SAS Technical Report P-222 [6], and O'Connor [7].
ENVIRONMENTS

What is an environment, other than a forest in which the spotted owl lives? With respect to the SAS Macro Facility, an environment is an area of the program in which specific symbolic macro variables are defined or known. The easiest way to think of referencing environments is as a series of boxes, each of which contains one or more smaller boxes. The largest box, which portrays the global referencing environment, represents the entire SAS session, and contains symbolic macro variables which are created in open code (that is, outside of any Macro Program), symbolic macro variables created in %GLOBAL statements, all automatic macro variables, and possibly symbolic macro variables created by the DATA Step SYMPUT routine.

Every Macro Program creates its own local referencing environment (which is portrayed by an enclosed box). This environment is empty until the Macro Program creates a symbolic macro variable (other than with a %GLOBAL statement). Furthermore, each Macro Program which is invoked from another Macro Program creates a local environment which is contained in the environment of the invoking Macro Program. It is extremely important to understand that the local referencing environment created by any Macro Program (and all of the symbolic macro variables in it) exists only while the Macro Program is executing; when the Macro Program stops executing, the corresponding environment ceases to exist. The environment in which macro activity is currently occurring is the current referencing environment.

These concepts may be illustrated by some simple macro code and a related environment diagram [Figure 1].

```
%LET M_OPEN = open code;
%MACRO M_A;
  %LOCAL M_AVAR;
  %LET M_AVAR = local environment for m_a;
%B
%MEND M_A;

%MACRO M_B;
  %LOCAL M_BVAR;
  %LET M_BVAR = local environment for m_b;
  %LET M_CVAR = (local environment for m_c)
%MEND M_B;

%MEND M_C(M_CVAR);
%M_A
```

Now that the concept of macro referencing environments has been introduced, we can begin to consider the relationship between these environments and the manner in which the macro processor creates symbolic macro variables and resolves references to these variables. Symbolic macro variables can be created by any of the following statements:

1. Iterative %DO statement,
2. %GOTO statement,
3. %GLOBAL statement,
4. %INPUT statement,
5. %LET statement,
6. %LOCAL statement,
7. %MACRO statement,
8. %WINDOW statement, or
9. SYMPUT routine (special case).

These concepts may be illustrated by some simple macro code and a related environment diagram [Figure 1].

![Figure 1: Macro Referencing Environments](image)

The %GLOBAL statement always creates a variable in the global referencing environment. If a variable with the same name already exists in the global environment, the %GLOBAL statement has no effect.

The %LOCAL statement always creates a variable in the local referencing environment of the corresponding Macro Program. If a variable with the same name already exists in the local environment, the %LOCAL statement has no effect.

Parameters specified on the %MACRO statement always create variables in the local referencing environment of the corresponding Macro Program.

The macro processor will go to almost any lengths to avoid creating any new symbolic macro variables. Consequently, all other macro program statements (other than the SYMPUT routine) which can create symbolic macro variables are processed identically. The macro processor first searches the current (local) referencing environment for the existence of a symbolic macro variable of the appropriate name. If the macro processor finds such a variable, it assigns the new value to that variable. If the macro processor fails to find a symbolic macro variable of the appropriate name in the current (local) environment, it searches outward (one environment at a time) for the desired variable. If it finds such a variable, it assigns the new value to that variable. If it fails to find an existing symbolic macro variable of the appropriate name in any environment, it creates a new symbolic macro variable in the current (local) referencing environment, and assigns it the appropriate value.

Although it would be nice if the DATA Step SYMPUT Routine worked the same way, that would be too much to expect. The SYMPUT Routine searches all referencing environments for an existing symbolic macro variable of the appropriate name, and assigns the new value to any variable found just as indicated above. However, if it fails to find an existing symbolic macro
variable of the appropriate name in any environment, the SYMPUT Routine does not automatically create such a variable in the current (local) referencing environment. It searches outward, starting from the current environment, looking for a referencing environment which is not empty, and creates the new symbolic macro variable in the first such non-empty environment it encounters.

Resolution of symbolic macro variables proceeds in a manner which is analogous to that utilized for the creation of these variables. The macro processor first searches the current (local) referencing environment for the existence of a symbolic macro variable of the appropriate name. If the macro processor finds such a variable, it utilizes the contents of that variable to resolve the symbolic macro variable reference. If the macro processor fails to find a symbolic macro variable of the appropriate name in the current environment, it searches outward (one environment at a time) for the desired variable. If it finds such a variable, it utilizes the contents of that variable. If it fails to find a symbolic macro variable of the appropriate name after it has searched the global referencing environment, it outputs a message indicating that an unresolved symbolic macro variable has been detected (if OPTIONS SERROR is set).

MACRO QUOTING

The Macro Quoting Functions are the single area of the macro processor which cause the most user confusion and frustration. This is somewhat surprising, because the purpose of the quoting functions is quite similar to that of the single quote (') in standard SAS code.

The primary objective of the macro quoting functions is to hide something (normally containing one or more special characters or relational operators) from the macro processor.

There are basically only three situations which require the use of a macro quoting function.

1. It is desired to include one of the special characters quote ("), double quote ("), left parenthesis (\), right parenthesis (\), percent sign (%), blank ( ), or semi-colon (;) in a symbolic macro value which appears on the right hand side of a %LET statement.

2. It is desired to prevent one of the special characters, such as plus (+), minus (-), asterisk (*), from being treated as a numeric operator, or to prevent one of the special character strings, such as GT, LT, AND, NOT, <, from being treated as a relational condition by the nefarious %EVAL function.

   NOTE: Even though you may have not explicitly specified the use of the %EVAL function in your macro code, this function will be invoked automatically by the macro processor for the evaluation of expression in the constructs noted below.

   (a) %DO macro-variable = expression %TO expression %BY expression;
   (b) %DO %UNTIL(expression);
   (c) %DO %WHILE(expression);
   (d) %IF expression ... ;
   (e) %QSCAN(argument,expression,delimiter)
   (f) %QSUBSTR(argument,expression,expression)
   (g) %SCAN(argument,expression,delimiter)
   (h) %SUBSTR(argument,expression,delimiter)

3. In addition to the conditions described in item 2 above, it is also desired to prevent the special characters & and % from causing symbolic macro variable resolution or Macro Program execution, respectively.

   Warning: Even though you may have not explicitly specified the use of the %EVAL function in your macro code, this function will be invoked automatically by the macro processor for the evaluation of expression in the constructs noted below.

   (a) %DO macro-variable = expression %TO expression %BY expression;
   (b) %DO %UNTIL(expression);
   (c) %DO %WHILE(expression);
   (d) %IF expression ... ;
   (e) %QSCAN(argument,expression,delimiter)
   (f) %QSUBSTR(argument,expression,expression)
   (g) %SCAN(argument,expression,delimiter)
   (h) %SUBSTR(argument,expression,delimiter)

Two of the macro quoting functions (%STR and %NRSTR) are designed to take effect at Macro Program compilation time; all of the other quoting functions are effective at Macro Program execution time. A good rule of thumb for which of these two classes of quoting functions to use at any given time is if the value to be quoted does not contain any symbolic macro variable names (%varname) or Macro Program names (%macname), use %STR; otherwise use %BOQUOTE, %NRQUOTE, or %SUPERQ.

The %QUOTE and %NRQUOTE macro quoting functions are now functionally obsolete. Consequently, use of these functions is not recommended.

The %LOWCASE, %QSCAN, %QSUBSTR, and %UPCASE macro quoting functions merely quote the result of the %STR, %NRSTR, %BOQUOTE, %NRQUOTE, or %SUPERQ function. How does the macro processor actually quote the specified strings? Very simply. Under SAS Release 6.xx, it merely appends a single hexadecimal character to the start of the quoted string and another single hexadecimal character to the end of the quoted string. The starting hexadecimal character not only marks the beginning of a quoted string, but also denotes the type of quoting to be applied to the string. The ending hexadecimal character is currently always the same (irrespective of the quoting function used). Furthermore, the exact hexadecimal characters used to mark the start and end of the quoted string differ from platform to platform, and are not documented by the Institute. Consequently, as chronicled in several Usage Notes, it is possible for the user to inadvertently include these marker characters in the string to be quoted (particularly if the SYMPUT routine is used to create the symbolic macro variable). Finally, these marker characters are conceptually stripped off by all of the Macro Functions, so they do not affect comparisons (%EVAL) or string lengths (%LENGTH, %SUBSTR, etc.).
All strings which have been quoted by a macro quoting function remain quoted as long as the string is being used by the macro processor, and therefore need not be quoted again.

When the macro processor generates text from a quoted string and passes this text to the SAS System, the marker characters are supposed to be removed automatically. However, due to bugs in some maintenance levels of several current releases of the SAS System on some platforms, the marker characters are not always correctly stripped. If this problem is encountered, the %UNQUOTE function can always be utilized in open code to correctly remove the quoting marker characters. (The %UNQUOTE function can also be used within a Macro Program, but this feature is seldom required).

COMMAND LINE MACRO PROGRAMS

A poorly documented feature of the SAS Macro Facility is the ability to develop a Macro Program which can be invoked on the Command Line of any screen. The Program Editor is the most likely candidate for this feature. Construction of a Command Line Macro Program is identical to the creation of any other Macro Program, as long as it is remembered that the code emitted from such a Macro Program must be processed by the Command Line interpreter, rather than the SAS System. In other words, the code emitted must consist of only valid Command Line commands.

The most useful feature of a Command Line Macro Program is the ability to emit to the Command Line interpreter more characters than could be typed on the Command Line itself (normally a maximum of 78 characters).

A Command Line Macro Program can be invoked in the normal manner by typing %macro macname on the desired Command Line. Alternatively, OPTIONS CMDMAC can be set, and the Macro Program can then be invoked by typing macname on the appropriate Command Line. However, if the latter approach is taken, the SAS session will incur (possibly non-trivial) additional CPU and I/O overhead to determine whether or not each word typed on any Command Line is in fact a valid Command Line Macro Program.

CONCLUSION

The SAS Macro Facility is indeed a powerful and robust tool. Not only does it allow the novice to utilize many of its capabilities, but it also contains many advanced features which can be used by the more sophisticated programmer. Hopefully, the information presented in this workshop will help explain some additional aspects of the macro processor, thereby enabling the attendees to make a more productive use of the capabilities of this tool.

REFERENCES


Author Contact:

Mark Bercov
Bercov Computer Consultants Ltd.
3641 - 13 St. S. W.
Calgary, Alberta, Canada
T2T 3R2
(403) 243 - 0686

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. © indicates USA registration.