ABSTRACT

Once the basic concepts have been mastered, understanding the internals is the next logical step to enhancing your knowledge of the SAS® Macro Facility.

This tutorial addresses the internals by identifying the components of the macro processor, illustrating their relationships with each other and their interactions with the SAS supervisor, and examining how a macro program is processed. Major components include the input stack, wordscanner, word queue, macro compiler, macro executer, symbolic substituter and open code handler. Special topics in macro processing include the performing of symbolic substitution, resolution, evaluation, quoting and unquoting, and autocall processing.

INTRODUCTION

The macro facility, which has been available since the 82.3 version of the SAS® Software System, gives you the capability of enhancing your SAS programming. The facility has added many new features and much functionality to the SAS system, in addition to allowing you the opportunity to implement your own features. It is important to remember however, that the macro facility is a part of the SAS system and the macro processor works with the other components of the SAS supervisor.

At times the concept of the macro processor being a part of the SAS supervisor is forgotten. Often, users attempt to change the basic structure of SAS processing by writing macros they think will accomplish this task. The macro facility is not magic. It does not change the way that the SAS compiler and SAS executer handle your code. The macro processor does allow you to control how your code is to be generated, allowing you greater flexibility and more advanced features.

The concept behind the macro facility is that it is a tool. The facility is not a 'packaged' product, it is a programming language within the SAS language. The macro facility has added many features and capabilities to the SAS language. It allows you to build complex applications, design menuing systems, and provide your own extensions to present SAS software. One of its simplest uses, generating text strings, is often used to avoid repetition of the same SAS statements and to allow you to conditionally execute SAS statements (and portions of SAS statements.) The facility is also useful for processing information from your system. The term system, in this instance, refers to both your operating system as well as your application system. There are automatic macro variables which allow you to obtain information from the operating system, &SYSJOBID for example. Special symbolic variables are also available for letting you obtain information from the SAS Display Manager System as well as design additional features for display manager.

In addition, functions are available for allowing you to obtain information from the DATA step and pass information back to it. These are only a few of the tasks that the macro facility gives you the capability to implement.

As the macro facility is only a part of the SAS system, the macro processor must interact with the other components of the SAS supervisor. The SAS code generated by your macros must be syntactically correct, that is it must follow the same rules and constraints that source code you would enter directly must follow. As you will see in 'Macro Processing', the SAS compiler cannot distinguish between code generated by the macro processor and the source lines you explicitly key into your program. Statement boundary rules, as well as other constructs of SAS processing, remain unchanged. The SAS processor performs sequential execution, that is the code for a step is compiled until the step is complete, and then the step is executed. Text generated from an executing macro, must be in the correct step sequence. If a PROC step is generated by a macro invoked in the middle of a compiling DATA step, the PROC step will end the compilation of the DATA step at that point. The DATA step code following the macro invocation is then flagged with syntax errors.

Before you can fully understand macro processing it is essential that you understand the basics of SAS processing, as the macro processor is only an extension of the SAS processor. The major components and their roles, as well as their interactions within the system, is reviewed in order to give you a clear understanding of how processing is done.

SAS PROCESSING

The input stack is the first component in processing your program. Your lines of source are brought into the system for processing through the input stack, on a line per line basis. Because your code is processed in 'tokens', when the request for a token is
received, the wordscanner tokenizes, breaks source lines apart word by word, and passes these words or tokens, one at a time, to the word queue. The word queue is a six-word channel through which tokens are made available to other components of the system for further processing.

Tokens move through the word queue, one by one, as the token in the first position is requested by the system, another token enters from the bottom, pushing tokens already in the queue up by one position. If the MACRO system option is in effect, additional activity occurs within the word queue. The word queue manager, another wordscanner component, checks each token as it enters position five (from position six) for special triggers indicating macro activity. These triggers are the ampersand (&) and the percent sign (%). An ampersand (&) token indicates a possible symbolic reference and that the symbolic substitution routine should be called. Similarly, a percent (%) token indicates that the word may be a macro invocation or macro keyword and that the open code handler should be called. When the above routines are called, the token is acted upon, (this action is described in 'Macro Processing'), and the result, emitted by the macro facility is placed onto the top of the input stack. (Remember that the token could have indicated an action to be taken by the macro facility, where no text would be generated.) Occupying the top element of the input stack, the result is passed through the wordscanner for tokenization and the first token is placed into the bottom position of the original word queue. As the next token enters the queue, the next to last word is checked for macro triggers. If one is found, the macro processor is called (symbolic substitution or the open code handler.) If no triggers are found, the words continue to move up through the queue, one position at a time, until each reaches the top position. As the tokens advance from the top position, each is passed to the SAS compiler.

The compiler checks to see if the token represents a step boundary. (Step boundaries consist of PROC statements, the DATA statement, the RUN statement, the CARDS statement and the PARMCARDS statement.) If the token is a step boundary, then compilation of the current step is ended. The step is then passed to the SAS executor. The next step then begins to compile as soon as execution of the previous step is completed.

When the compiled step is passed to the SAS executor, the appropriate DATA step or PROC routine is called.

MACRO PROCESSING

The processing done by the macro facility begins when the word queue manager recognizes a macro trigger (percent sign or ampersand) in the token currently occupying the fifth position of the word queue. The type of action taken depends on the type of trigger that was found.

Handling of Ampersands

If the trigger is an ampersand, the symbolic substitution routine is called. This routine immediately builds his own word queue, in which to copy the bottom three words of the original word queue. The symbolic substitution manager is then called to locate the environment containing the name in the last position of the word queue.

The current referencing environment, the symbol table established by the currently executing macro is checked first. If the variable is not there, the next-outer environment, or symbol table, is checked. This environment would be the global referencing environment, established during SAS invocation, if the executing macro was not invoked inside another macro. If there was a nested invocation, the next-outer environment would be that of the next-outer macro. The outermost environment will always be the global environment.

If the name is located in one of the symbol tables, symbolic substitution places its value into his work area and then advances his word queue in search of another symbolic reference. This continues until there are no more adjacent symbolic references. When this point is reached, symbolic substitution pushes his results (the contents of his work area) onto the input stack and removes the words that constituted the initial trigger from the original word queue.

If the name is not found in any of the symbol tables, symbolic substitution produces no results and the original word queue is undisturbed. If the SEKSOR system option is in effect, symbolic substitution also issues the following message:

WARNING 1501: APPARENT SYMBOLIC REFERENCE UNRESOLVED

Handling of percents

If the trigger found in the next to last position of the word queue is a percent sign, the open code handler is called. This routine builds his own word queue into which he copies the bottom three words from the original word queue. The open code handler checks the word in the last position to see if it is a reserved word. If it is, the routine which performs the specific action is called. (Two of these routines, resolution and evaluation are discussed in 'Special Topics in Macro Processing').

If the last word in the queue is MACRO, the open code handler recognizes that the beginning of a macro definition is occurring. He then calls the macro compiler. The macro compiler builds his own word queue from the last tokens in the word
queue of the open code handler. The macro compiler continues to advance his word queue, compiling the tokens, until he encounters the percent sign followed by MEND. During the time he is compiling, macro program statements are checked for syntax and are compiled, SAS source statements are stored as text. The compiled macro is stored on disk, and the table of compiled macros is updated to include the name of the newly compiled macro.

If the open code handler finds a reserved word, other than %MACRO, it calls the necessary routine to perform the action. If the token is not a reserved word, the open code handler checks to see if it is the name of a macro that the user has defined. He then calls the macro executor who processes macro program statements and places generated SAS code on the input stack.

If the name was not found, the open code handler then checks to see if the autocall facility is in effect. (Processing done by autocall is discussed in 'Autocall Processing'). If the autocall facility is not in place, the open code handler leaves the words in the queue of the open code handler undisturbed. If the SAS system option, MERROR, is in effect, he then issues the following message.

WARNING 1353: APPARENT MACRO INVOCATION NOT FOUND

SPECIAL TOPICS IN MACRO PROCESSING

Certain areas of macro processing tend to be more complex or at least less obvious than others. Five of these areas are symbolic substitution, especially when symbolic values return symbolic values, resolution, evaluation, quoting and unquoting, and autocall processing.

Symbolic Substitution

The process of symbolic substitution is basically described in 'Handling of Amperands'. It is reemphasized here in order to show that even though the symbolic reference may grow in complexity, the actual simplicity of the routine remains the same. In many applications it is often useful for one symbolic reference to yield another symbolic reference. This is often done by adding additional ampersands to the front of the macro variable reference (or references in cases of concatenation.) It is important to remember two basic concepts here. First of all, an ampersand immediately followed by another ampersand (with no space in between) will always resolve to a single ampersand when processed by the macro facility. Secondly when tokens are concatenated, they are processed from left to right. Consider the following application.

Suppose you are asked to produce a report listing sales observations and summary statistics by department for weekly sales information. The user requesting the report has asked that he see the observations and summary statistics together, for each department. You decide to use the PRINT and MEANS procedures for this application, but realize you will need to invoke these procedures multiple times in order to group the output as requested. If you subset the data set based on the values of DEPT, the variable representing department number, and invoke the procedures on each subset the application is simple. The only problem is that you will have to modify the program each time a department is added or removed. Using the macro facility, the process can be automated, regardless of departments that may be added in the future. The following example illustrates one method of coding this application.

/* Set up macro variables to represent */ /* the values of Department */
PROC SORT DATA=LIB.DPTSALES;
   BY DEPT;
DATA _NULL;
   SET LIB.DPTSALES END=EOF;
   BY DEPT;
   IF FIRST.DEPT THEN
      DO;
      N+1;
      CALL SYMPUT('DEP'|[LEFT(N),DEPT]);
      END;
   IF EOF THEN
      CALL SYMPUT('TOT',LEFT(N));
RUN;

/* Design the macro Gensys, */ /* to perform the application */
%Macro GENSYS;
   %DO I=1 %TO &TOT;
   DATA TEMP;
      SET LIB.DPTSALES;
      IF DEPT=&&DEP&I;
   RUN;
   PROC PRINT;
      TITLE 'SALES OBSERVATIONS FOR DEPARTMENT &DEP&I';
   RUN;
   PROC MEANS;
      TITLE 'SUMMARY OF SALES FOR DEPARTMENT &DEP&I';
   RUN;
   %END;
%Mend GENSYS;

The first step in the application is to read the input data set, and set up macro variables to represent each value of department. This is
done by using the FIRST. method to enter a DO loop each time the BY value changes. The DO loop first increments a counter, N, by one each time the FIRST. condition is met. This counter is then left justified, converted to a character string, and concatenated onto the literal 'DEP'.

The current value of DEPT is then assigned to the macro variable TOT. Since the number of department values will often be unknown, TOT can be used whenever this information is needed.

The second step defines the macro GENSYS to perform the application. The %DO loop, incrementing I by one each time through the loop, will generate the steps to perform the application, the number of times specified by the macro variable TOT. In the first execution of the loop, the DATA step subsets the input data set, if DEPT is equal to the first BY value represented by &&DEP&I. PROC PRINT AND PROC MEANS are then invoked on the data set, TEMP. This process is repeated each time the loop is incremented, with the value of &I reflecting which BY value is being used. Notice that a different title will also be generated for each execution of the loop. The process for evaluating &&DEP&I may appear complex, but you will see that it is simply the same process of symbolic substitution.

How is &&DEP&I evaluated? Remember that the tokenizer breaks the string into tokens as it is taken from the input stack. These tokens are then passed into the word queue one by one. Consider the first reference to &&DEP&I, on the DATA step IF statement. At the time the first ampersand reaches the next to last position of the word queue, the preceding equal sign is occupying the position immediately above. The ampersand triggers symbolic substitution who builds his own word queue and copies the bottom three words from the original word queue. He then finds an ampersand in the last position. Symbolic substitution evaluates && as &, and places & in his work area. He then advances his word queue.

The next word, DEP, enters the queue. As there is no action to perform on the token as it enters the next position, he copies DEP into the work area (beside the &.) The next token to move up is the ampersand. As the ampersand reaches the next to last position, and I enters the last position, &I is evaluated as 1, (the first time through the %DO loop) and the 1 is placed into the work area. Advancing the queue again, symbolic substitution finds a blank. He then pushes the results in his work area onto the input stack and removes the initial trigger from the original word queue (the equal sign is not disturbed.)

The top element on the input stack is now the string &&DEP&I. The word scanner passes the words & and DEP to the word queue. When the & moves from the last to the next to last position, symbolic substitution is called. He again builds his word queue, copying the bottom three words. He places the value of &&DEP&I onto the input stack, and copies the & and DEP from the original word queue. The tokenizer then passes this result (the first value of department) to the word queue.

The second reference to &&DEP&I is enclosed in double quotes as part of a text string on the TITLE statement. Remember that a text string enclosed in single or double quotes is processed as one token. When this token enters the word queue, it is recognized as a double quoted literal. The word queue manager calls a special routine to handle double quoted literals, the resolve routine. (This routine is also used to execute the RESOLVE data step function.)

Resolution builds a word queue in which to process the literal. He simply takes in a string, and generates a string. &&DEP&I is recognized by the word queue manager and processed by symbolic substitution before entering the word queue of resolve. The words inside the text string continue to move through this word queue until the string is complete. Remember that tokens entering the resolve word queue, require no additional action.

Evaluation

The evaluation routine works very much like the resolve routine. It also only processed tokens that require no further action. When eval is called, he builds a word queue and receives tokens from the word queue manager. Remember that this routine only sees the actual elements he will compare. Any other type of action is performed before tokens enter his queue. In the expression:

```
%LET x = 5;
%LET y = 3;
%LET z = 7;
%LET a = 2;
%LET b = 1;
%LET c = 0;
%LET d = 1;
%LET e = 1;
```

...the tokens that enter the word queue are the values of &x, &y, &z, &a, &b, &c, &d, and &e. Eval then performs the addition and pushes the result back onto the input stack.

An important concept to remember is that the same evaluation routine handles both implicit and explicit evaluation. Explicit evaluation occurs when you use the %EVAL function. This function should be used when you are trying to assign a numeric value requiring integer arithmetic or the result of a comparison (0 or 1) to a macro variable. It should not be used on either side of the operator on a %IF statement, in the numeric positions on a %DO statement, or in the numeric positions of macro functions. In all of these places evaluation is implicitly done for you. If you do use %EVAL in these places, the evaluation routine is called twice.

Quoting and Unquoting

Quoting functions, the most mystical part of the macro language are as essential to macro programming as the quotation marks in the SAS language. The need for quoting is obvious from the simple example below.

```
DATA;
```
with a delta character. This allows the quoting, or removing the significance of special characters occuring as a result of symbolic substitution. A means for quoting specifically for the macro facility was needed.

For lack of a good special character to substitute (due to restricted meanings and different keyboards), the decision was made to begin literal with %STR( and to close it with ). %STR was the initial literal quoting function written for the macro facility.

%STR is used to remove the significance from all special characters except ampersands and percent signs. Another function, %NRSTR, was written to also quote these characters. %STR and %NRSTR are not executed by the macro processor, instead these functions are components of tokenization. They are recognized by the tokenizer just as single or double quotes are. Because %STR and %NRSTR are both a function of tokenization, they only remove the significance from special characters in their actual argument. As you know, tokenization occurs before macro execution or symbolic substitution. A means of quoting special characters occuring as a result of resolution or execution was needed.

The first quoting function written for this purpose was %QUOTE. It removes significance from the same special characters as %STR. The difference is that %QUOTE is executed by the macro processor during macro execution and symbolic substitution. It is used to quote characters in values that are unknown until macro execution has occurred. %NRQUOTE was added to work in the same way as %QUOTE, but to also work on ampersands and percent signs. Adding the capability of quoting unanticipated special characters was the purpose behind %QUOTE and %NRQUOTE.

Adding the capability of quoting all these characters during macro execution without attempting to resolve any macro variable references or macro invocations was the reason for adding %SUPRQ. This function is actually an autocall macro, shipped on the installation tape. %SUPRQ uses an as a different routine, metasynt for retrieving symbolic values.

Quoting, or removing the significance of special characters, is done by replacing the character with a delta character. This allows the characters to be processed without the usual meanings or action being taken. In order for the meaning of a quoted character to be restored, the delta character must be replaced with the original character.

Unquoting, or removing the delta character is performed is performed two different ways. Unquoting is performed explicitly by the %UNQUOTE function. This function should be used when you have quoted a special character and want to restore its significance for further processing by the macro facility. Unquoting of delta characters is also done automatically as the token passes from the fourth to the third position in the word queue.

Autocall Processing

The autocall facility allows you to invoke macros that will be compiled on a demand basis. That is, unless you actually invoke a macro, it does not get compiled. This feature has two obvious advantages. It is more efficient, in that a system that can execute one to any number of macros depending on user response, now will only compile those macros that are actually invoked. It also makes the fact that you are using macros less obvious to the users, who may not even recognize they are using them.

Autocall works by having you place the source code for your macro definitions in members having the same name as the macros they contain, in partitioned data sets under OS/VS and maccibs under VM/CMS. The library (or concatenated libraries) is then referenced by using the LIBNAME specified by the SASAUTOS= system option. In order for the macro facility to recognize that autocall macros are available, the MAUTOSOURCE system option must also be in effect.

Guidelines to follow in using the autocall facility are generally to use good judgment in planning the macro libraries. It is a good idea to place the source code for macros into the autocall library only after they have been tested and debugged. It is also a good idea to only put autocall macros in this library. Be careful when choosing names for the macros, especially if they are defined as statement style macros. You do not want them confused with other functions or keywords.

The installation tape your site receives provides you with an autocall library supplied by SAS Institute Inc. The SAS clist, exec, and catalogued procedure are already set up to reference this library. Under TSO, you can use the AUTOS parameter of the SAS clist to specify your library name. Invoking the clist as

SAS AUTO(by.MACLIB)

automatically allocates your macro library, concatenating it in front of the system level library. You can also allocate other libraries with different LIBNAMEs, and change the SASAUTOS= system option to reflect which library you are working with.
Under OS, you can supply a SASAUTOS DD statement to concatenate your library with the system library. If you do not concatenate the library, your allocation will override that of the system library.

Under CMS, you can modify your version of the SAS exec on your A disk, to concatenate your library in front of the Institute supplied library. Remember to add your maclib name to the GLOBAL statement, as well. You can also FILEDEF your maclib, global it, and set the SASAUTOS= system option appropriately. This can be done before or after invoking the SAS exec.

Understanding the processing done by the macro facility when autocall is used will help you to use the facility more easily and efficiently. When a string of the form %X is encountered by the open code handler, remember that it is first checked against the list of reserved words. If it is not found, the string is then checked against the compiled macro list. If it is not there, the open code handler checks to see if autocall is active, the MAUTOSOURCE option is in effect. If it is, the open code handler then checks to see if the SASAUTOS= library has been defined. If not, the following error is generated.

ERROR 1354: SOURCE LEVEL AUTOCALL DDNAME NOT FOUND OR CANNOT BE OPENED.
AUTOCALL HAS BEEN SUSPENDED.

If you are running interactively, you can then allocate the library and reset the MAUTOSOURCE option. You will need to invoke the macro again.

If the library has been defined, autocall will bring the source code from the library member of the same name into the system. All the source code in the member, including the macro definition, is compiled. As soon as the code in the member has been compiled, the invoked macro will be executed. Usually, the only source code in the member is the macro definition, but occasionally you may want to include other macro definitions or even additional SAS code.

In using autocall there are certain concepts you will want to remember. First, autocalling macros is more efficient than using the %INCLUDE statement to bring in the source code for your macros. Autocall was written for this purpose alone, and performs open and close routines specifically for autocall. The macro facility will never autocall a macro that has already been compiled. This can pose a problem when the macro you have autocalled contains a syntax error. One way to correct an error is to use the INC command on the display manager command line to bring the member into the program edit window. Edit the macro, correcting your error. Use the SAVE command, with the REP option to place the corrected member in the library. Then, submit the source code from the program editor window and reinvoke the macro. The system will recompile the macro from your source. Remember, it is suggested that you debug your macros before placing them in the autocall library.

A network of autocall macro libraries can provide your programmers and users with many additional capabilities. Multiple libraries can be concatenated, usually with the user level first, then the system level at the site, and then the library supplied by the Institute.

CONCLUSION

If you have attempted to use the macro facility and found it difficult remember that a good understanding of the SAS system is important before many of the macro concepts are clear. A knowledge of macro processing will make your programs easier to design, implement, and debug. Topics in macro processing will become clear with more use of the facility. Remember that the macro facility is provided as a tool with which you can enhance your own applications, it is a programming language within the SAS language.

Further information on these topics can be found in The SAS Guide to Macro Processing®, 1987 Edition.

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