Paper TT05

An Animated Guide©: How %Str and %NRStr Work

Russ Lavery, Numeric Resources Contractor – Ardmore, PA

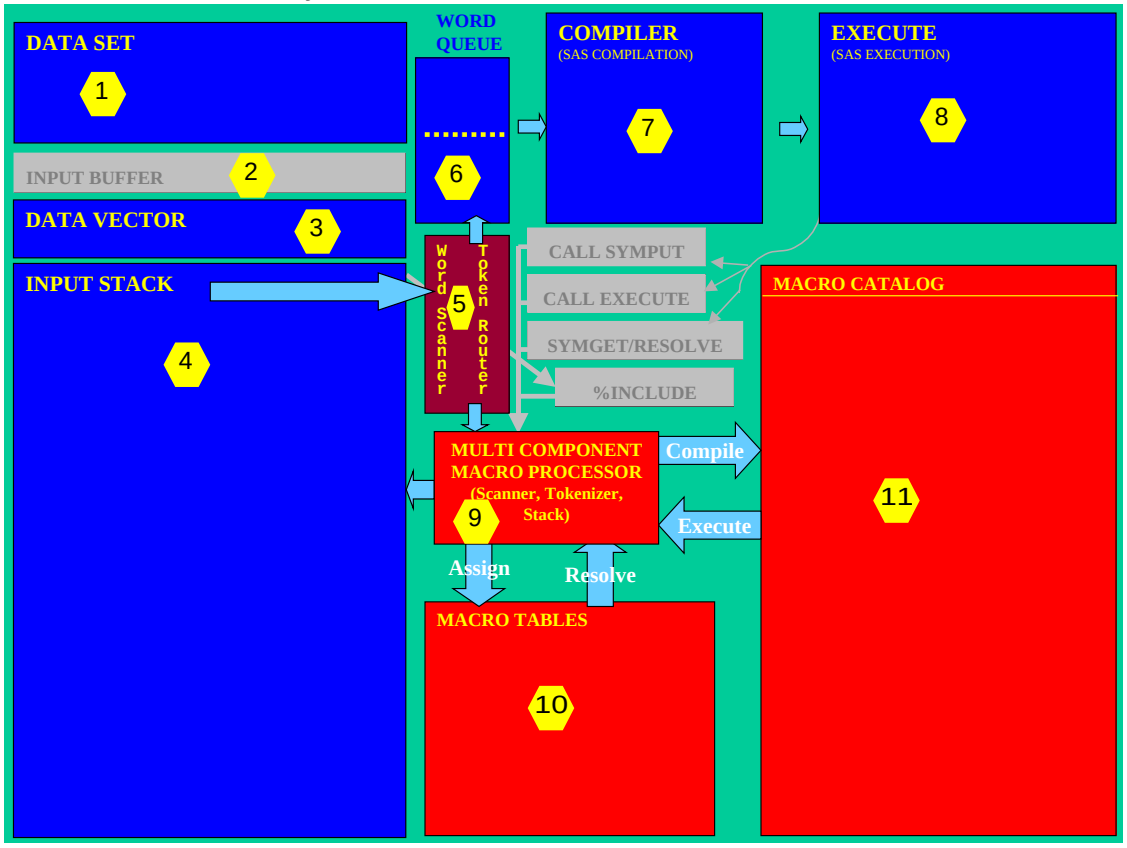

Figure 1

ABSTRACT

This paper builds on a NESUG 2002 paper that described the general functioning of the SAS® Macro Processor. This paper explains details of two macro masking functions: %Str and %NRStr. Basic concepts for this paper are: 1) the map of the SAS Supervisor 2) Tokens flow/pASS through the parts of the system 3) parts of the map monitor tokens as they pass through 4) the idea of SAS tokens as rule triggers for actions to be taken by parts of the map and 5) macro masking prevents the triggering of rules by preventing their recognition. Figure 1 is a map of the SAS system – the SAS Supervisor. It is suggested that this map is a very useful paradigm for understanding the SAS system. Boxes on the above map are either subroutines or storage areas and tokens flow through the components of the supervisor as SAS processes programs.

INTRODUCTION

This paper builds on material from a NESUG presentation titled "An Animated Guide: The Map of the SAS Macro Facility". Basic concepts for this paper are: 1) the map of the SAS supervisor 2) Tokens flow through the parts of the system 3) parts of the map monitor tokens as they pass through 4) the idea of SAS tokens as rule triggers for actions to be taken by parts of the map and 5) macro masking prevents the triggering of rules.

Tokens flow through all the parts of the system. Logically, the macro system “sits on top of” the regular SAS system and exchanges tokens with the SAS system. Parts of the system are always watching for certain types of tokens ,as the tokens pass through, and will initiate subroutines when these types of tokens are recognized. In the language of this paper, “tokens trigger rules”. Macro Masking is the process of disguising tokens so that they do not trigger rules as they pass through “watchpoints” of the system.

Using this map requires a knowledge of the SAS system, the macro system and rules that are applied at each component in both systems. A review is essential so that readers, who understand the system in their own way, can have a common vocabulary concerning the SAS system.

As an abbreviation, (Fig1-1) stands for the object labeled with a (1) in Figure 1- the data set.

The paper has the following structure:

1) Overview of the Map of the SAS system (overview of the Map of the SAS Supervisor) and a review of tokenization
    1A) Non-Macro components of the map and processing non-macro SAS code
    1B) A short review of tokens
    1C) Review of the Macro Components of the Map and processing SAS code
2) Tokens as rule triggers and parts of the system that monitor tokens and apply rules
3) The need to examine rules for 2 storage areas (Symbol Table and Macro Catalog) and two functions (%Str and %NRStr)
4) A main confusion in many macro explanations
5) How tokens are masked
6) Rules for the Macro Symbol Table
    6A) The effect of %Str on tokens going into, and tokens coming out of, the Macro Symbol Table
    6B) The effect of %NRStr on tokens going into, and tokens coming out of, the Macro Symbol Table
7) Rules for the Macro Catalog
    7A) The effect of %Str on tokens going into, and tokens coming out of, the Macro Catalog
    7B) The effect of %NRStr on tokens going into, and tokens coming out of, the Macro Catalog
8) Manual Unmasking
9) Conclusions, references and acknowledgments and contact information.

The Map of the SAS system can be used as a mental framework for SAS macro processing. The map presented here is an integration of several maps presented by different authors. The author wishes to acknowledge an intellectual debt to the authors of the materials cited at the end of the paper.

1) OVERVIEW OF THE MAP OF THE SAS SYSTEM (MAP OF THE SAS SUPERVISOR)

Figure 1 (the main graphic) shows a map of the SAS system, not just the Macro Facility. The control program for the SAS system is sometimes called the SAS Supervisor and the description of SAS processing in this paper is a description of the functioning of the SAS Supervisor. It is impossible to discuss the SAS Macro Facility without placing it in context of the SAS Supervisor's management of the base SAS system (non-macro processing).

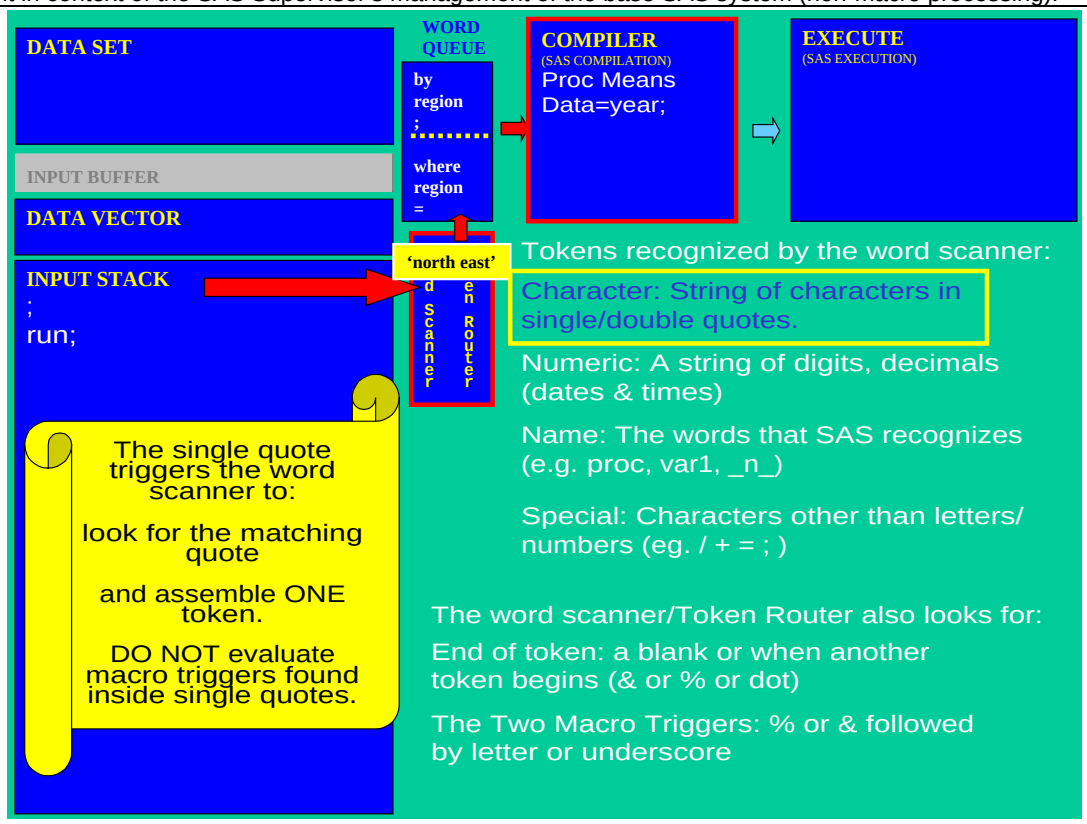

Figure 2

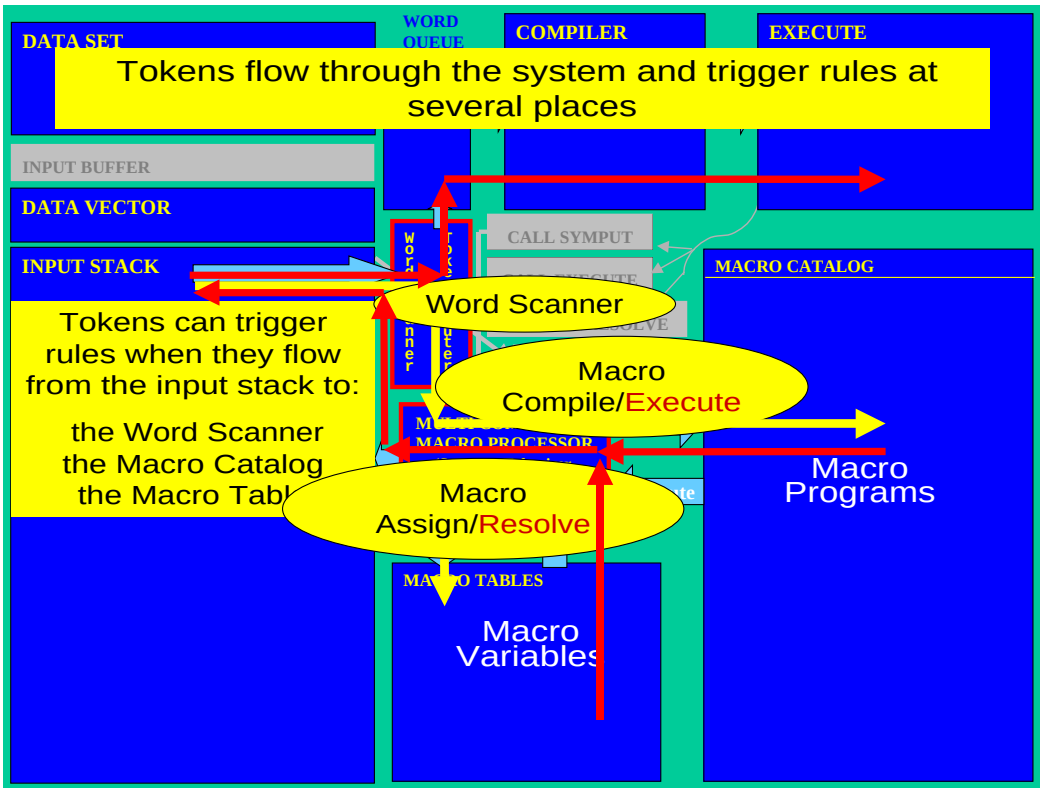

FIGURE 3

1A) NON-MACRO COMPONENTS OF THE MAP & PROCESSING NON-MACRO SAS CODE

Each box in Figure 1 is a subroutine or storage area. A box is a component of the SAS Supervisor or the Macro Facility, and serves a unique purpose. The blue boxes, and the dark red box, are required for processing regular SAS code. The light red boxes are used specifically for macro processing. The phrase “the macro system sits on top of the base SAS system” means that the macro system gets its tokens from the base SAS system and passes its results (more tokens) back to base SAS (blue boxes) for additional processing.

In Figure 1 the data set (Fig1-1) can be text or a SAS data set. If the data set is a text file, data will flow from the text file into the Input Buffer (Fig1-2) and then to the Program Data Vector (PDV) (Fig1-3). The input buffer holds one line of unparsed data from a text file in preparation for passing it to the PDV. The Input Buffer is not used in this presentation, but is included on the map for completeness. If the data set is a SAS data set, observations are read directly from the data set into the PDV(Fig1-3).

Understanding the PDV is critical for SAS programmers. It exists in RAM and can be thought of as a one-row Excel® spreadsheet. Observations are read into the PDV and all calculations, coded in a data step, are performed in the PDV. When a data step has finished processing an observation, values in the PDV are written to the output file.

It is suggested that the reader look at both Figures 2 through 6 as the next components of the map are discussed. The tokens flow through the system as is shown in Figure 2. All the tokens shown in following figures were once on the input stack. The Input Stack (Fig1-4) is a little known part of the SAS system. Submitted code does not go directly to the compiler, as is often thought, but goes to a holding area called the Input Stack.

The compiler can’t use code without pre-processing and the Input Stack holds code as it waits to be processed. Individual characters from the top of the Input stack flow into the word scanner/token router (WS/TR) (Fig1-5). The WS/TR has two functions: 1) It takes individual characters from the Input Stack and assembles them into tokens (groups of characters that the compiler can process) 2) It also decides if the token should go to the Word Queue (Fig1-6) or to the "Multi-Component-Macro Processor (Figure 2 shows all tokens going to the Word Queue and Figure 5 shows tokens going to the macro processor).

1B) A SHORT REVIEW OF TOKENS

Tokens have been mentioned quite often and a review of tokens might be appropriate at this time. The SAS system understands tokens as instructions. Tokens flow through the map of SAS and are the way a programmer to passes instructions to the SAS system. The conversion of text to SAS tokens is an important, and basic, part of the SAS system. There are many kinds of tokens but we will only discuss the four most common. They are:

Character Tokens: Strings of characters in single/double quotes.
(note that SAS handles single quoted character strings differently from double quoted character strings)
Numeric Tokens: A string of digits, decimals (dates &times)
Name Tokens: The words that SAS recognizes (e.g. proc, var1, _n_)
Special Tokens: Characters other than letters/numbers (eg. / + = ;)

As the WS/TR it takes characters off the Input Stack and tries to assemble them into tokens, it checks for a couple of things. First it checks every character it pulls off the Input Stack to see if the token currently being assembled has ended. The end of a token is indicated by either a blank space, a period or the start of another token (e.g. + or ;). The WS/TR also checks for two characters called "Macro Triggers". These characters (the & followed by a letter or underscore or the % followed by a letter or underscore) are signals to the WS/TR that following text should be routed to the Macro Processor and not to the Word Queue.

The special tokens are troublesome in macro processing. They are very often “rule triggers” or tokens that, as they pass through certain boxes on the Map, trigger SAS to take some action. It is easier to understand macro processing if the rules, and “rule applying points” are made more explicit. Below please find some of the rules that special tokens, and others kinds of tokens, trigger. The rule applying points, and more rules, will be discussed later.

Blank	Macro processor will trim leading/trailing blanks - in WS/TR
;	End SAS Statement
,	Separate Parameters Inside a Function
+ - / *	Arithmetic Operators (trigger an arithmetic operation)
**	
= EQ	Comparison Operators (trigger an evaluation)
LT GT < >	
LE GE	
& | NOT	Logical Operators–differ from Comparison operators (trigger a logical evaluation)
OR AND	
(	Start to Collect a Parameter List for a Function
)	End of collecting a Parameter List for a Function
‘ “	Look for the Matching Component of the pair of quotes
% &	Macro Triggers
&name	Macro Resolution
%name	Macro Invocation

The Word Queue (Fig1-6) holds six tokens and allows the supervisor to access "previously assembled" tokens and build *context* for the token currently being assembled in the word scanner. The SAS Compiler (Fig1-7) drives the system. It requests tokens from the word scanner until it receives a token that indicates a step boundary (e.g. run, quit, proc). In Figure 2, the compiler is waiting to receive the run that is on the Input Stack.

When the SAS Compiler receives a “step boundary token” (run, quit and others), it takes total control of the system and attempts to compile code. No tokens are assembled in the word scanner, or moved, while the SAS compiler is in control. If the code is correct (matching quotes, semicolons etc.), it will be compiled and passed to the SAS Execute module. The SAS Execute (Fig1-8) module takes total control of the process. Other parts of the map, become inactive. If the job has no run errors (data mismatches, etc.) and the code runs. Generally, tokens do not move while the SAS Execute module is in control.

1C) REVIEW OF THE MACRO COMPONENTS OF THE MAP & PROCESSING MACRO SAS CODE

The Macro Processor (Fig1-9) is shown as a single box, but actually has many components. The Macro Processor has its own scanner, tokenizer and stack. It also has major components called the “Open Code handler” and the %Eval. Due to space limitations, this sub-system will be shown as a box, and components described as needed.
For all its complexity, the macro processor is simply a token “management system”. It is a token storage and retrieval system, much like an automated version of the Microsoft clipboard. It (generally) takes text/tokens from the Input Stack, stores them in one of two areas (Fig1-10 & 11), and puts those tokens to the Input Stack at a later time.

The macro system has two token storage areas: the Macro Symbol Table (Fig1-10) and the Macro Catalog (Fig1-11). There are two different storage areas because there are two different storage processes. The Macro Symbol Table is like a memory location on a calculator. It stores and recalls tokens. The Macro Catalog is a storage area that, combined with the Macro Processor, allows conditional processing of tokens. It allows programming steps to be applied to the process of recalling tokens. Using statements like %if you can control *whether* a certain section of code (a group of tokens) gets moved to the Input Stack. Using statements like %do allows you to control *how many times* a section of code is recalled and moved to the Input Stack.

The Token Router part of the WS/TR decides if tokens are sent to the Macro Processor or to the Word Queue. The Macro Processor 1) either stores the tokens on one of the two types of memory storage, or 2) uses the token to trigger recall of other tokens, from one of the two memory storage areas. Token flow details will be developed later.

2) TOKENS AS RULE TRIGGERS AND PARTS OF THE SYSTEM THAT MONITOR TOKENS AND APPLY RULES

As can be seen in Figure 3, parts of the Map of the SAS system are constantly alert, monitoring tokens as they pass. Arrows indicate paths of token flow and certain tokens, at certain spots in the system, will trigger the application of a rule (start a subroutine). Not all boxes on the Map are "rule trigger points" and the rules that are triggered (the process kicked off) by a token differ from box to box. The WS/TR is a major "rule sensing spot", as is the Macro Processor itself. While mentioning other spots, this paper will concentrate on these two.

3) THE NEED TO EXAMINE RULES FOR 2 STORAGE AREAS (TABLE AND CATALOG) & 2 FUNCTIONS (%STR AND %NRSTR)

It is logical to conclude that, since the two storage areas have different capabilities, the processes for getting tokens into and out of the two types of storage areas are different. It is also logical to conclude, since the tokens will be processed differently depending on their destinations, that tokens will trigger different rules- depending on their destination. Simply said, a token going into the Symbol Table might trigger a rule, but the same token, going to the Macro Catalog, might not trigger a rule- or might trigger a different rule. This difference in token processing, the fact that processing depends on the destination of the tokens, forces a structure on the study of Macro Masking and on this article. It is key to identify rule trigger points and list the possible rules that can be triggered at particular points.

4) A MAIN CONFUSION IN MANY MACRO EXPLANATIONS

A source of confusion in most macro documentation is the inappropriate re-use of the words "compile" and "execute" (example %Str and %NRstr are compile functions). SAS performs at least three "compiles" and three "executes" in a complicated macro program. These three compiles and executes have very different rules, do very different things, and fail in very different ways. Calling these three different processes by the same names makes the macro process harder to understand. As shown in Figure 1 shows, this paper will give the different compiles and executes different names. The names are: SAS Compile, SAS Execute, Macro Compile, Macro Execute, Assign and Resolve.

We all have experience with SAS Compile and SAS Execute. They are the modules that process regular SAS code. The other compiles and executes are associated with loading code into, and removing code from, the two macro storage areas. Putting tokens into the Macro Symbol Table will be called an "Assign". Removing tokens from the Symbol Table will be called a "Resolve". Putting tokens into the Macro Catalog will be "Macro Compilation". Removing tokens from the Macro Catalog will be called "Macro Execution".

5) HOW TOKENS ARE MASKED

Characters are really stored as numbers and translated to human readable symbols for display. When SAS masks a character, it adds a constant to that number (character), and "remembers" what it did. Computer terminals, and programs, only need 127 characters and the ASCII character set has space for 256. SAS masks by adding a constant to the number (character). The constant is large enough so that the sum of the original number plus the constant is greater than 127 (it is "shifted" out of the range of printable characters). In V8 SAS, all shifted characters print as a small box. In V9, while the symbols are without meaning to the programmer, they are not all the same.

Parts of the SAS system have rules that will be triggered by both the shifted and original character, but most parts/rules will not respond to the shifted character. Generally, masked, or shifted tokens, flow pass the "rule trigger points" in the system without being recognized and without triggering rules.

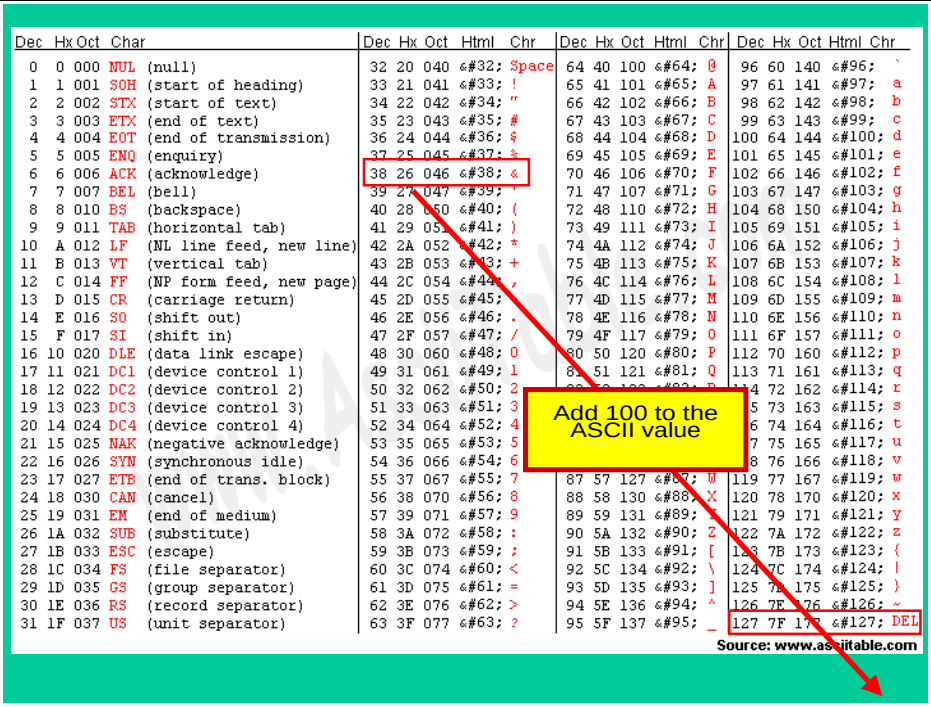

FIGURE 4

6) RULES FOR THE MACRO SYMBOL TABLE (A.K.A. THE SYMBOL TABLE)

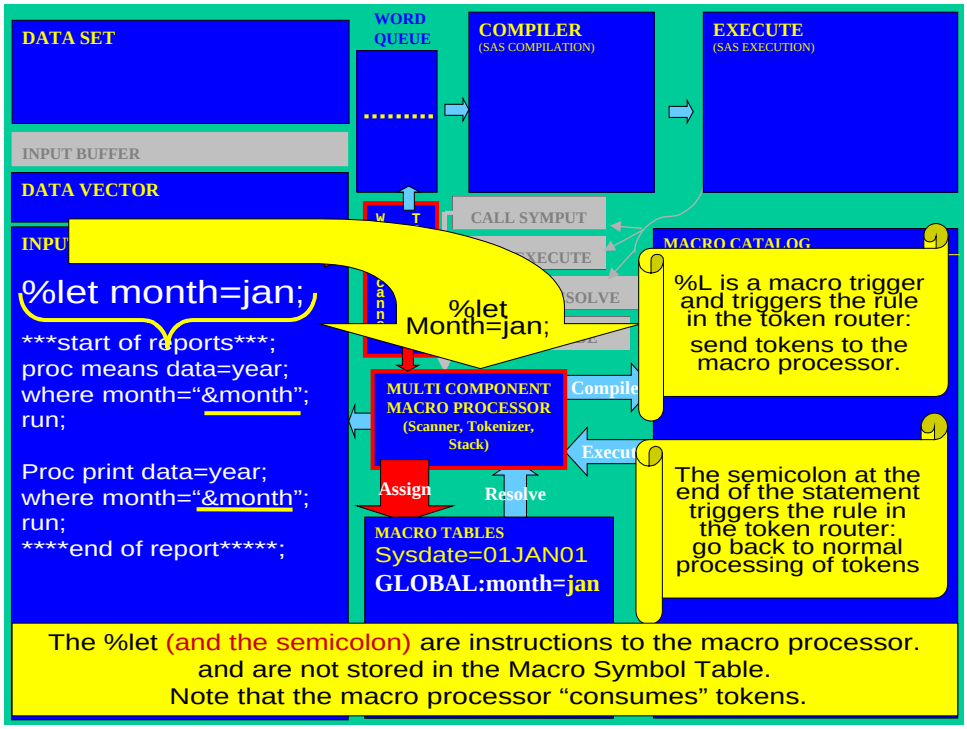

Figure 5

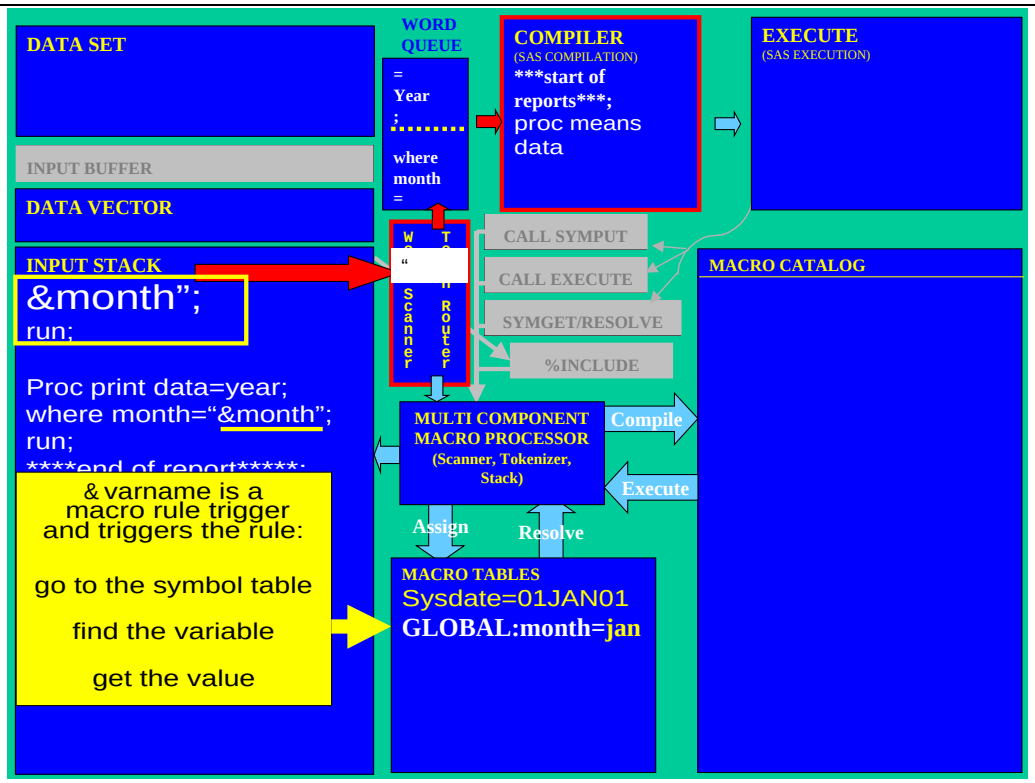

FIGURE 6

The Macro Table (or Macro Symbol Table or Symbol table) is used to store strings that get recalled to the input stack at a later time. The Macro Symbol Table can be thought of as something like the clipboard in Windows® products. The Symbol Table, generally, is not used to conditionally process code. This paper calls putting values into the Symbol Table "**Assigning**" a macro value and the process is illustrated in Figure 5. Text strings are often Assigned (put into the Macro Table) with commands like:

```
%Let comp= The Acme Storage Company;    or   %Let year= 1988;
```

Commands, like those immediately above, are typed into the body of SAS code and when they reach the top of the Input Stack the WS/TR examines them. The "macro trigger tokens" %L (remember a % or & followed by a letter or an underscore is a macro trigger) triggers a rule in the Token Router part of the WS/TR. The rule is: pass this token, and following tokens up to the first semicolon, to the Macro Processor for further processing. The semicolon at the end of the %Let statement, as it passes through the WS/TR on its way to the macro processor, triggers a WS/TR rule: stop sending tokens to the macro processor.

In Figure 5, the information on the right of the equality in the %Let is simple text. The macro processor creates a macro variable called month and assigns it the value jan. The %Let statement puts the text between the equal sign and the first semicolon into a named storage area (here, the name is month) in the Macro Symbol Table.

In Figure 6, we see the start of Macro Resolution. The token &month has reached the top of the input stack and is being passed to the WS/TR. The two characters &m are a macro trigger and trigger the rule in the WS/TR: "send this one token to the Macro Processor". Figure 7 shows the token being passed through the Word Scanner to the Macro Processor. Figures 7 and 8 show how the Macro Processor (MP) responds to that token. The MP recalls tokens from the Macro Symbol Table to the top of the input stack.

As the tokens pass through the macro processor, on the way to the Symbol Table, (see Figure 7) they can trigger rules. If the right hand side of the %Let statement is not simple text, but contains macro triggers, the triggers are evaluated as part of the process of being stored in the Macro Symbol Table. Figures 9 and 10 develop some of the issues we will explore with %Str and %NRStr masking and the Macro Symbol Table. %Str and %NRStr act as tokens flow into the two macro storage areas.

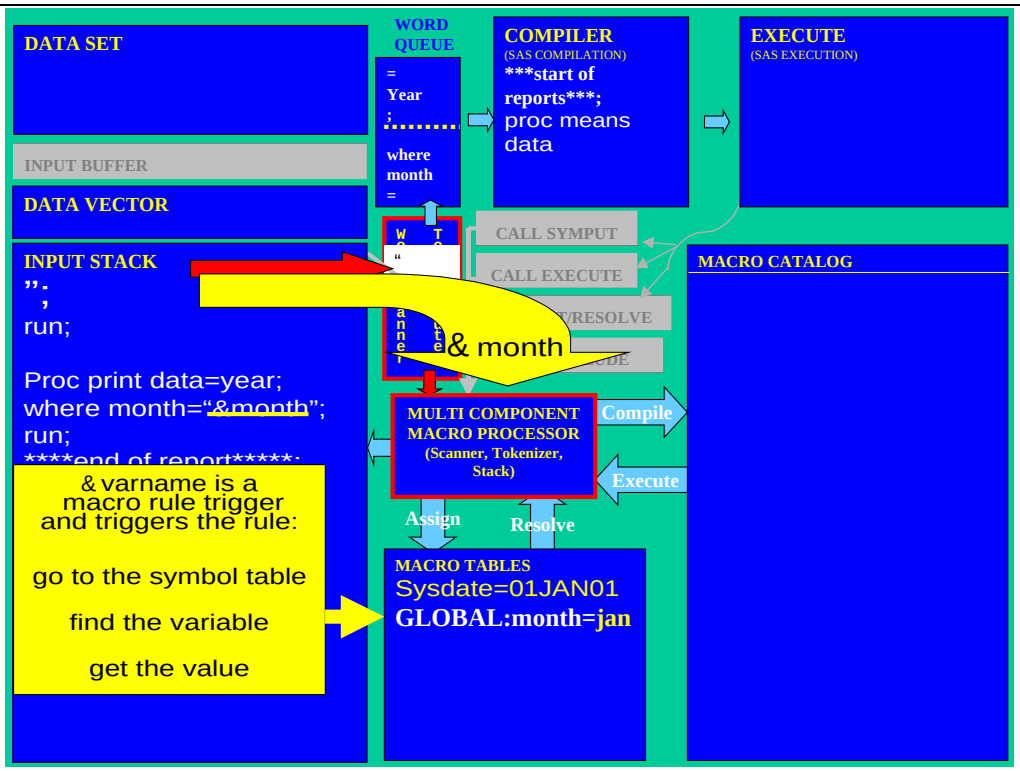

FIGURE 7

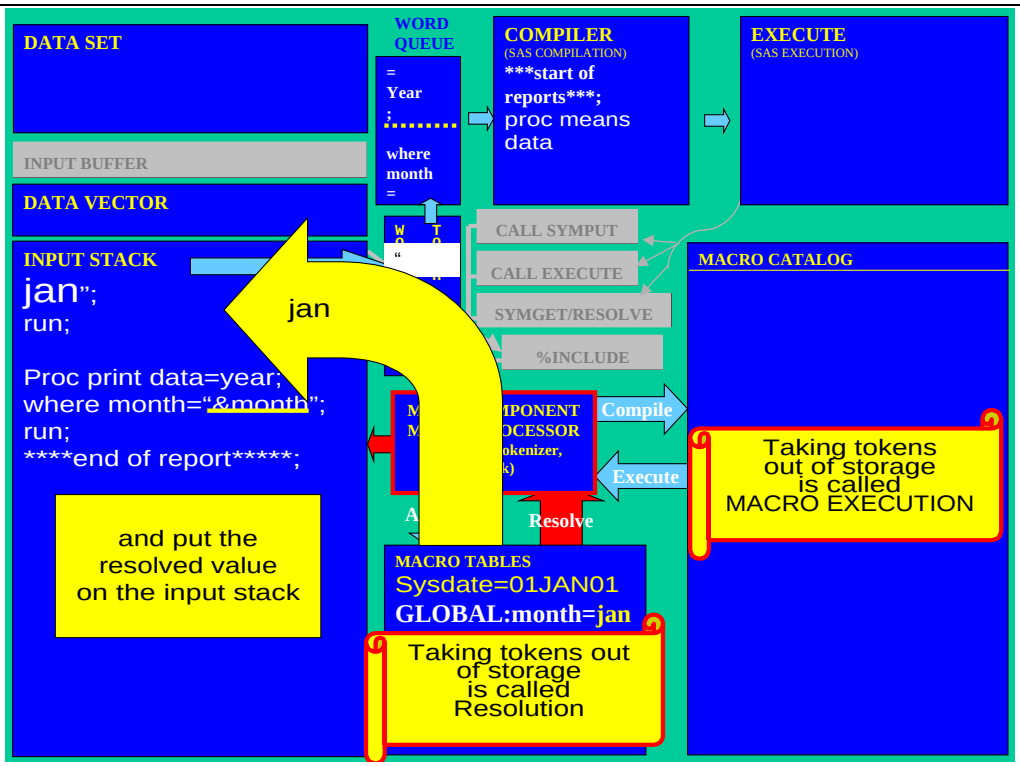

FIGURE 8

Some of the rules applied (and that can be blocked by %Str and %NRStr) as tokens flow *into* the Macro Symbol Table are:

1) The syntax of a %Let is: % let macro_name = value to be stored; (Fig-5)
2) The first semicolon ends a %Let statement (Fig-9)
3) The %Let trims leading and trailing blanks that are on the right of the =.
4) Macro functions (%upcase, %substr etc) are evaluated before tokens go into the Macro Symbol Table (Fig-10)
5) Macro Triggers (& and %) are resolved/executed before tokens go into Macro Symbol Table. (Fig-10)
6) The Macro Processor looks for matching quotes on the right side of the equal sign in the %Let.

In Figure 9, the programmer wants to create “a cheat”. He wants to store in a macro variable called oops, the text:

```
Proc print; run;
```

Mistakenly, he codes and runs the following

```
%Let oops= Proc print; run;
```

His intention is to use &oops to recall the `Proc Print; run;` paragraph to the input stack, and run it. He intends to use this “cheat” to save some typing when printing the most recently created data set. This “cheat” will not function properly because of how it is stored in the Macro Symbol Table. The Macro Processor would store the string “Proc Print” in the Macro Symbol Table and leave

`Run;` on the input stack.

Some rules for this process are shown in the graphics below.

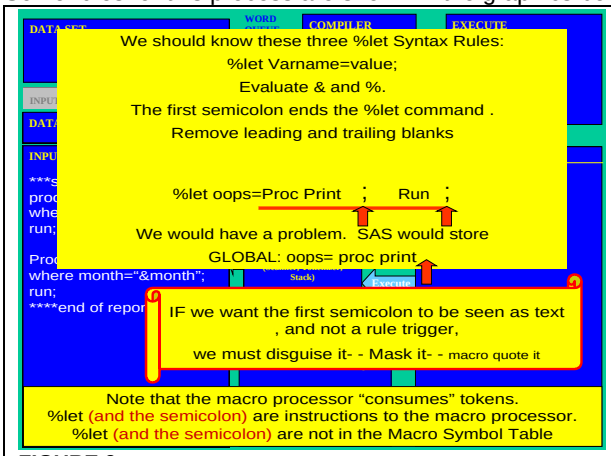

FIGURE 9

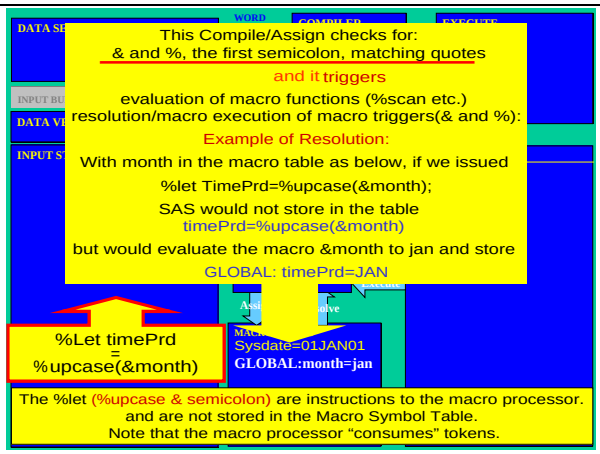

FIGURE 10

6A) THE EFFECT OF %STR ON TOKENS GOING INTO, AND TOKENS COMING OUT OF, THE MACRO SYMBOL TABLE

The rules, given above, allow us to see a problem with Figure 9. In Figure 9, the first semicolon, after the %Let, triggers a rule in the WS/TR. The rule is: “stop sending tokens to the macro processor”. If we want to have this semicolon not trigger the rule in the WS/TR, we must mask it so that the WS/TR does not recognize it.

In the figure below, the semicolons have been masked. This is indicated by the semi-transparent triangles covering the semicolons. This is a convention that will be carried through the paper.

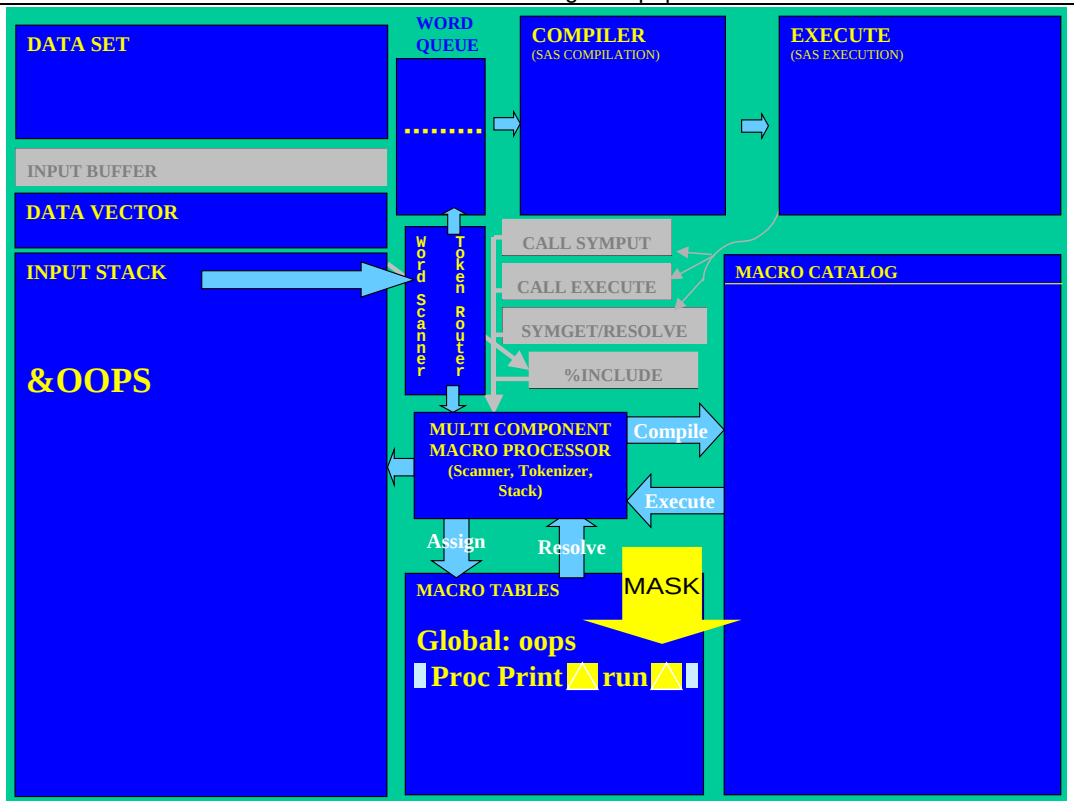

FIGURE 11

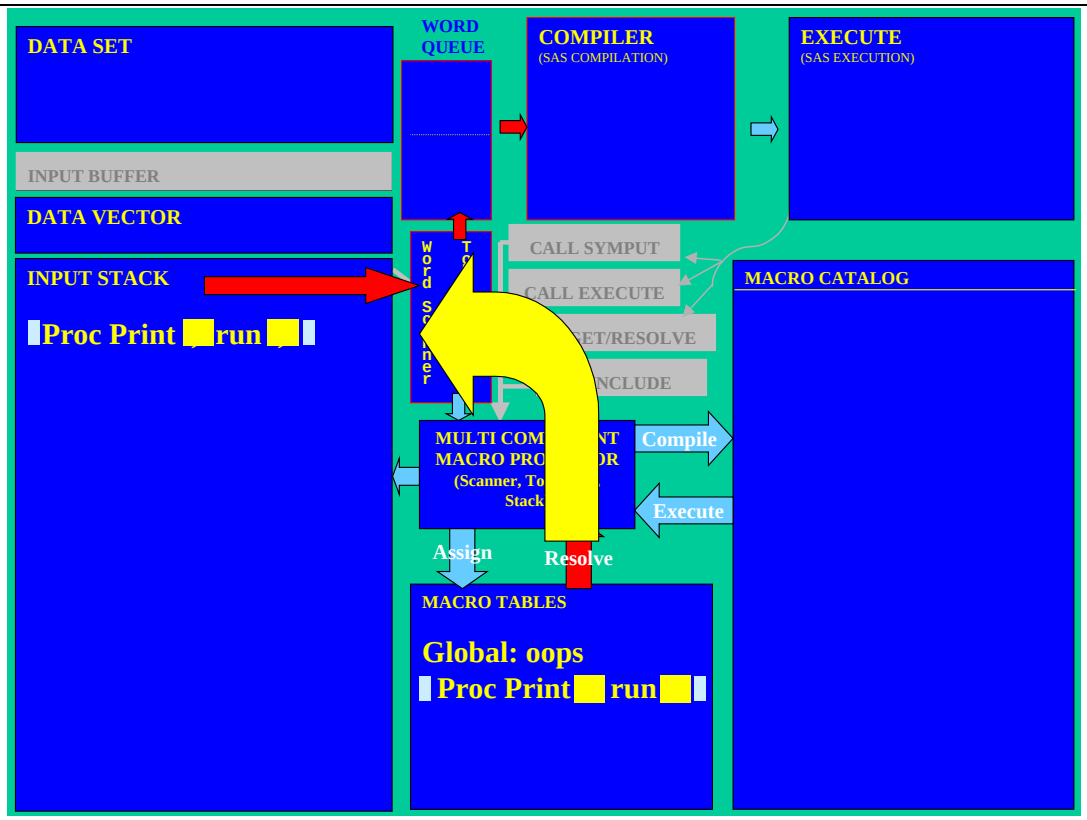

FIGURE 12

To store the full print paragraph in the Macro Symbol Table the programmer must code:

```
%Let oops= %Str(proc print; run;) ;
```

The %Str, some think, executes (masks tokens) in the WS part of the WS/TR as the characters flow from the input stack through the WS/TR. It is thought that the first unmasked semicolon is a rule trigger for the WS/TR, and to not have it trigger a rule, it must be masked before/while in the WS/TR.

The masked/shifted semicolons flow through the WS/TR and do not trigger the Token Router rule: “stop sending tokens to the macro processor (send them to the word Queue)”. When the code above executes, all the tokens in parentheses would be sent into the Macro Symbol Table, as is shown in Figure 11. Figure 11 shows the masked semicolons stored in the Macro Symbol Table and some boxes at the start and end of the string of tokens.

These boxes are not masked parenthesis from the %Str() function, but are non-printable characters that hold information SAS uses for internal processes. Many different masking functions can be used to mask a string and so SAS places characters, at the start and end of the masked string, to identify what type of masking was applied. Different masking functions add different non-printable characters, though they all look like boxes when they appear in a V8 log.

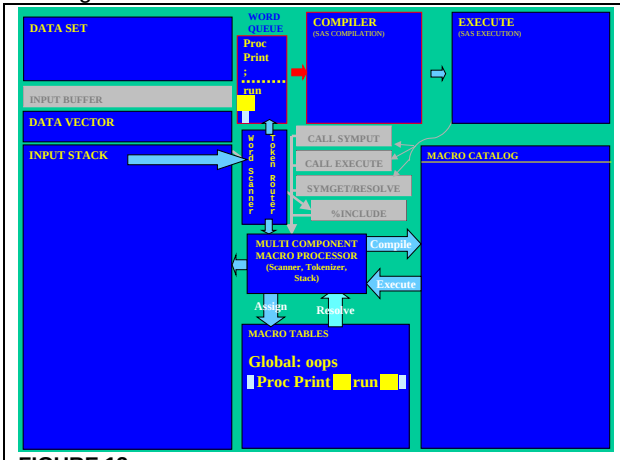

FIGURE 13

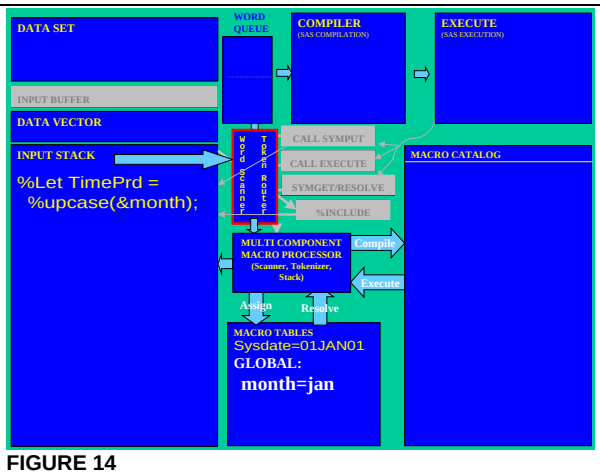

FIGURE 14

When the &oops reaches the top of the input stack (Figure 11), the macro variable is Resolved and the tokens are recalled to the input stack (Figure 12). At this time the semicolons are still masked.

The compiler requests tokens, and tokens flow along the path: Input Stack – WS/TR – Word Queue - Compiler. The masked semicolons do not trigger rules in the WS/TR (if any rules were appropriate) and simply flow into the Word Queue. Between the third and fourth position in the Word Queue is a subroutine called the automatic unmasking barrier. Masked tokens are automatically unmasked at this point. In Figure 13, the semicolon after the print has been unmasked, while the semicolon after the run has not yet been unmasked. The SAS compiler would be confused by masked tokens and the automatic unmasking barrier insures that the SAS compiler never encounters masked tokens.

References describe %Str() and %NRStr() as compile functions. This means they have their effect on tokens as the tokens are flowing into the storage areas (on Assignment and Macro Compile – but not SAS Compile). The character strings %Str or %NRStr are never stored in either type of storage area. What is stored is ***the result of their actions on the token***s inside the trailing parentheses - those tokens after masking. %Str will mask many tokens. The list is : + -*/<>= ^; , blank AND OR NOT NE LE LT GE GT. Very importantly, %Str does NOT mask the two characters that start macro triggers: the & and the %. Since %Str does not mask % and &, it will not prevent the execution of macro functions in Assignment. This characteristic of %Str, is shown below.

```
%Let month=jan;
   %Let TimePrd =%upcase(&month);
%put _user_;
```

& month resolves and Function executes. TimePrd is All Caps.

```
--LOG--
GLOBAL MONTH jan
GLOBAL TIMEPRD JAN
```

6B) THE EFFECT OF %NRSTR ON TOKENS GOING INTO, AND TOKENS COMING OUT OF, THE MACRO SYMBOL TABLE

Figure 10 showed that macro triggers complicate storing characters in the Symbol Table. % and & are evaluated as tokens flow into the Macro Symbol Table. Issues mentioned in Figure 10 are developed in Figures 14 to 18, specifically issues relating to the functioning of %NRstr.

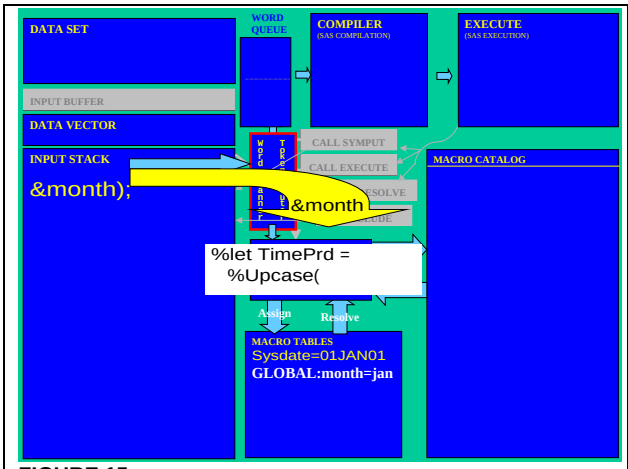

FIGURE 15

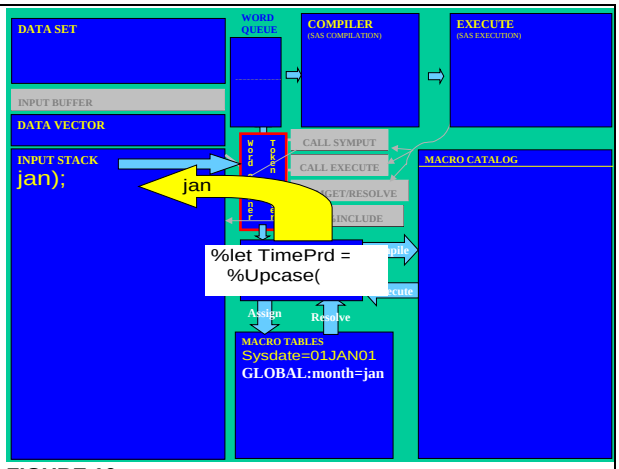

FIGURE 16

Figure 14 is the starting point for discussing %NRStr. There is a macro variable called month in the Symbol Table with a value of jan (note that jan is in lower case). A suggested process for loading a macro called TimePrd into the Symbol Table is shown in Figures 14 to 17. In Figure 14, the WS/TR recognizes the Macro trigger %L as it takes characters from the input stack. Tokens flow to an internal stack in the macro processor until the WS/TR encounters the macro trigger &m. It passes &month to the macro processor and jan is returned to the input stack. The WS/TR resumes passing tokens to the Macro Processor stack. When the Macro Processor receives the semicolon, it creates the macro variable TimePrd and the %upcase executes and JAN is stored in the symbol table. -see Figure 17. Macro invocations (& and %) and functions are evaluated as tokens are loaded into the Symbol Table.

%NRStr masks all the tokens that are masked by %Str and, in addition, the two tokens that start macro triggers. I suggest that %NRSTR performs its actions in the WS/TR and the results of the Masking are passed on to the Macro Processor. Workings of %NRSTR are illustrated below.

```
%Let month=jan;
%Let Cmplx=%NRSTR(%upcase(&month));

%put _user_;
%put &cmplx;
```

%NRStr blocks Resolution of &month and the execution of %UPcase!! A Masked & is stored in the Symbol .table!!

```
28   %put _user_;
GLOBAL MONTH jan
GLOBAL CMPLX +•upcase(▣month) –
GLOBAL TIMEPRD JAN
%put &Cmplx;
   %upcase(&month)
```

V9.12 Log does not use boxes to show masking

The process for Cmplx is not illustrated in detail, but an interim step is shown in Figure 18. At this point, the WS/TR has masked the characters that are rule triggers and the %Let command is in the "internal stack" of the Macro Processor-about to be executed. Masking is indicated by semi-transparent triangles covering the characters and by the boxes at the start and end of the masked string. Note that the % and & are both masked.

As seen above, the command %put _user_ does not attempt to Resolve/unquote any macro invocations in the symbol table. The command %put &cmplx will unmask tokens and cause resolved tokens to be put to the log. Note that + • are two characters. The first is the internal SAS symbol that indicates that this string has been masked by %NRStr and the second character is the (v 9.12) masked representation of the %.

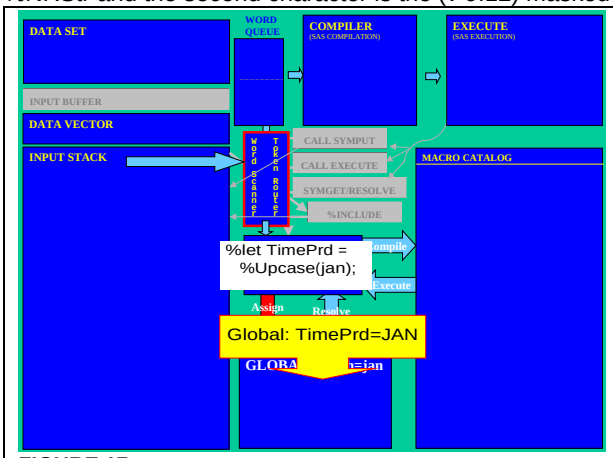

FIGURE 17

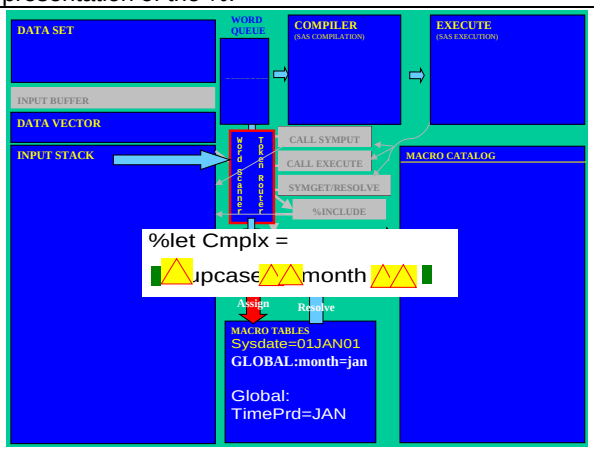

FIGURE 18

7) RULES FOR THE MACRO CATALOG

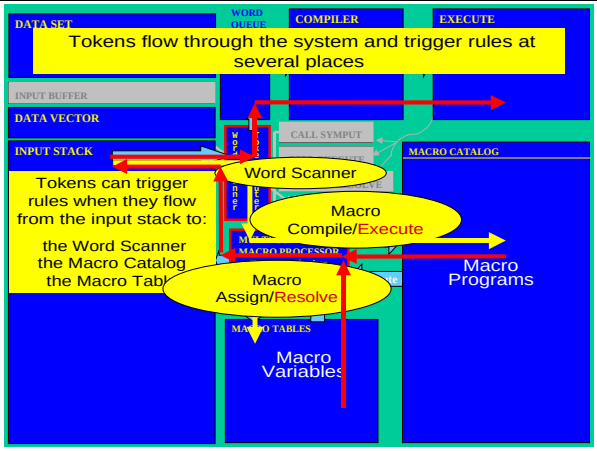

FIGURE 19

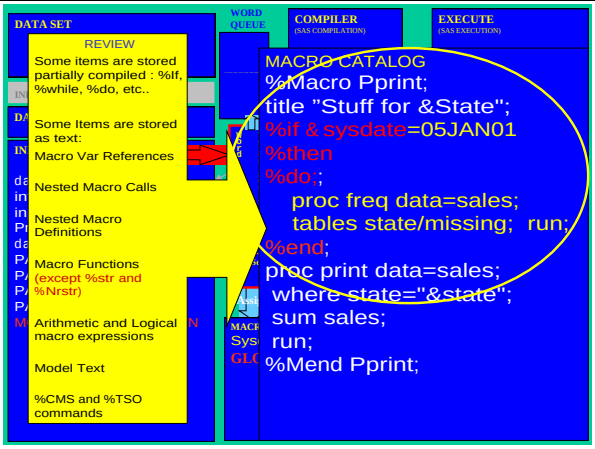

FIGURE 20

This section will discuss the effects of %Str and %NRStr on tokens going into, and flowing out of, the Macro Catalog (Fig1-11). Figure 19 shows some of the paths tokens can take through the SAS supervisor and places where rules are triggered. Since the underlying paradigm for this paper is that macro masking prevents tokens from triggering rules, we need to understand the rules for tokens flowing into the Macro Catalog and rules for tokens flowing out of the Macro Catalog. Rules applied to tokens entering the Macro Catalog are illustrated in Figures 20 to 22.

Unfortunately, the amount of code that must be shown requires that the size of different boxes in the map be changed. Not changing the map has been a goal of this paper but can no longer be avoided. In Figure 21, the Input Stack is enlarged and in Figures 20 and 22, the Macro Catalog is enlarged.

Figure 20 reviews several important rules of Macro Compilation. As tokens flow into the Macro Catalog (are Macro Compiled) the Macro Compiler/Processor checks that for every %if there is a %then. Every %do must have a %end.

The tokens starting with a % (%if %then %do etc) are stored in a partially compiled form. Most other tokens are stored as text. An additional rule is that the first semicolon ends a %then statement and this can be seen in Figure 21. Macro Compilation is not a very complex process.

Unlike macro variable references/invocations (eg. &state) flowing to the Macro Symbol Table, there is no attempt to evaluate the macro references/invocations as they flow into the Macro Catalog. These will be evaluated on Macro Execution, when the tokens flow out of the catalog. Accordingly, tokens like &sysdate and &state will be stored as text, as is shown in Figure 20.

There is no attempt to evaluate the truth of the %if %then statement as tokens flow into the Macro Catalog. In fact, there is no checking to see if any logical statement exists between the %if and %then tokens. The truth of the %if will be evaluated as tokens flow out of the Macro Catalog through the Macro Processor.

Unlike what happens as tokens flow into the Symbol Table, macro functions are not evaluated as the tokens flow into the Catalog- EXCEPT for %Str and %NRStr. These functions are the subject of this paper and we will see the effect of these functions in following slides.

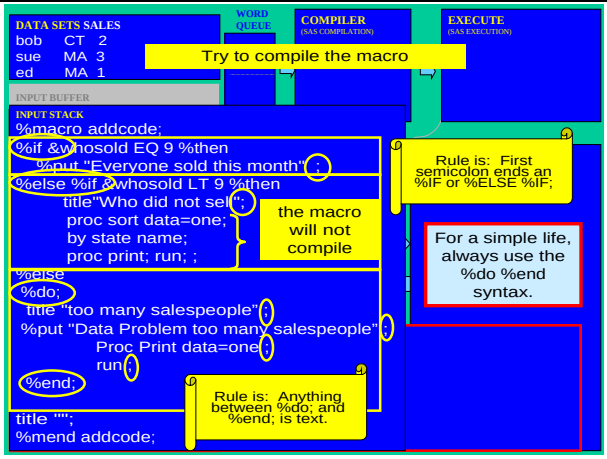

FIGURE 21

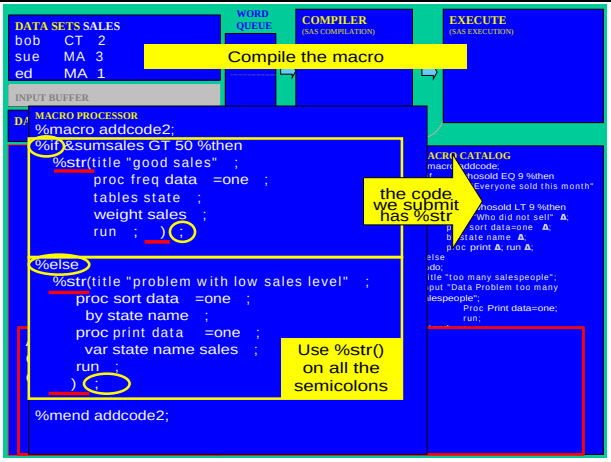

FIGURE 22

Figure 21 shows the rule "The Macro Compiler thinks the first semicolon ends a %if %then statement". The first semicolon after the %if is an instruction to the Macro Processor, as it Macro Compiles the tokens. The macro in Figure 21 will not compile because the %Else %If %Then statement ends with the first semicolon. The bracketed code in Figure 21 (near the box saying "the macro will not compile") will be interpreted as NON-macro code stuck in the middle of a macro and the Macro Compiler can not handle this situation. One simple solution is to use %if %then %do %end, as shown in Figure 21. Alternatively, the semicolons between the %do %end are considered as text, not instructions. The type of problem, shown in the %else %if in Figure 21, can also be fixed by the use of the %Str function as is shown in Figure 22. In Figure 22 the %Str masks the semicolons on Macro Compilation and the first semicolon seen by the macro processor is the one in the yellow circle.

The author suggests that the rules for tokens coming out of the catalog are best illustrated in two complicated slides. It is difficult to explain the actions of the macro processor without knowledge of the system. Accordingly, the slides show the values in the Symbol Table, the command that was run (before compilation) and how the macro processor resolves the command. As support for the interpretation given in this paper, the log is included in the upper right hand corner and the log shows all steps. The creation of the values in the Symbol Table is not shown in this paper.

In Figures 23 and 24 we see the steps of a simple test of the %if logic in Macro Execution. Because of the inability to animate steps in a process in a paper, input to the Macro Catalog and output from the Macro Catalog are both shown in the macro Processor box. The upper line of the code is input. It is shown with an arrow → pointing to the right, indicating that this is the code is sent to the catalog.

%if &Sco1 EQ %Str(G+W) %then	Stands for the simple test code that was sent to the Macro Catalog. Full code is shown to right.	%if &Sco1 EQ %Str(G+W) %then %do; %put It was TRUE ; %end; %else %put FALSE;
%if G W EQ G W %then	Stands for the if statement that was retrieved from the Macro Catalog after Macro Resolution	

What is stored in the Macro Catalog is: %if &Sco1 EQ G W %then - ***the result of the masking***. This code is returned to the macro processor and undergoes a three-step process of evaluation. The Macro Processor processes the statement from "inside out" and passes results up to the next step of evaluation. The lower if statement, in the Macro Processor, shows how the macro was evaluated.

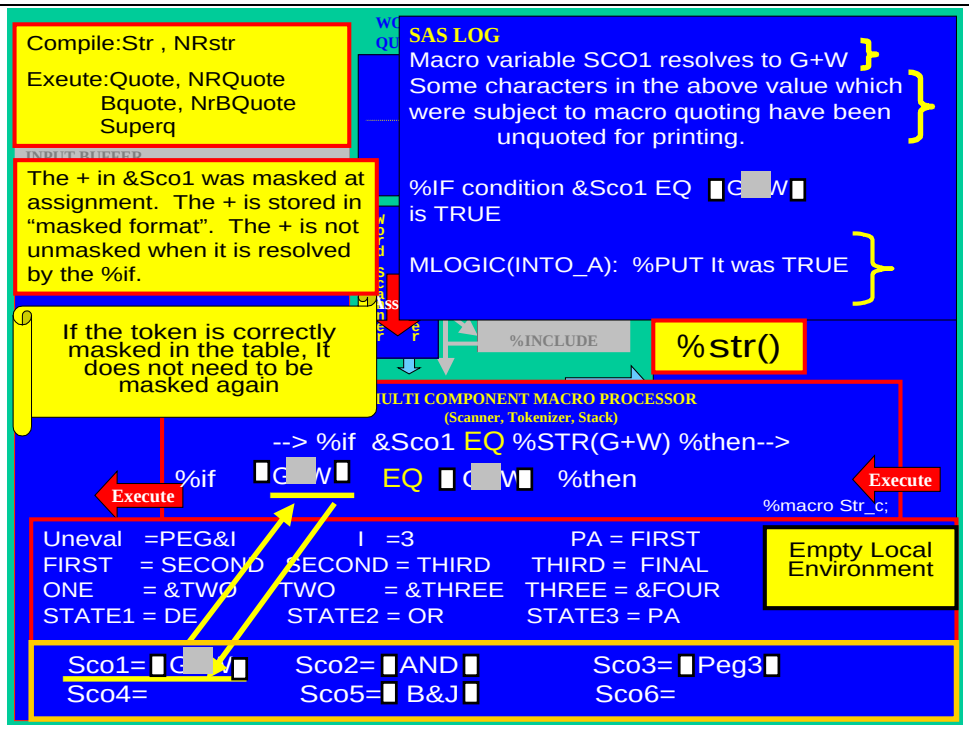

FIGURE 23

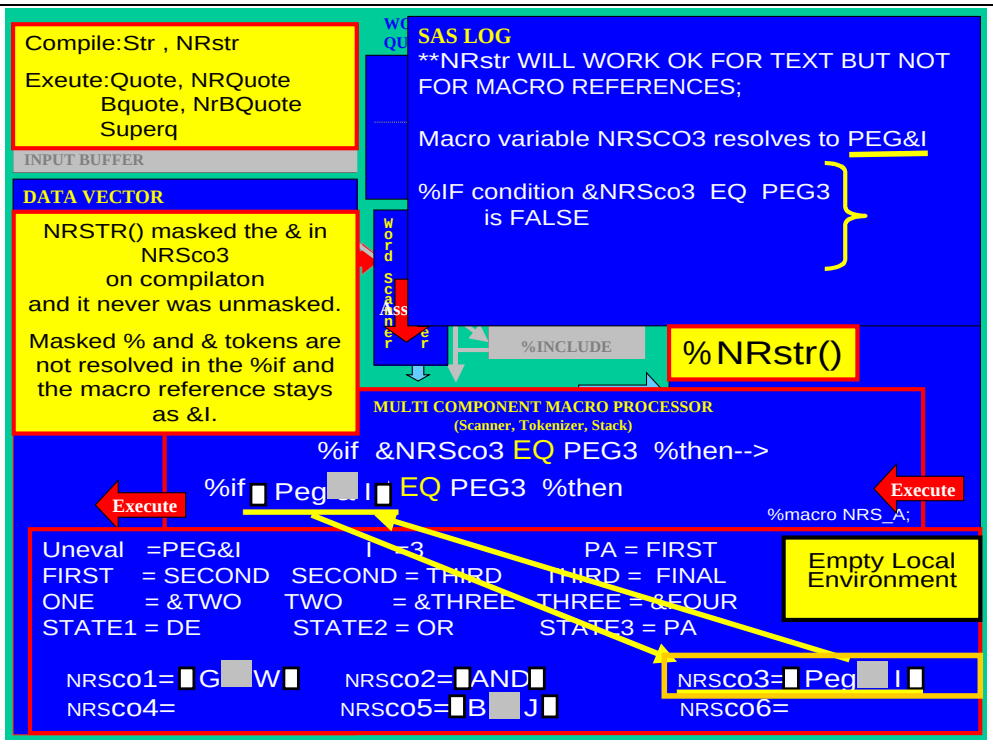

FIGURE 24

The first step is resolution of unmasked macro references. Note how the &sco1 was evaluated in Fig. 23 and &NRSco3 was Resolved in Fig.24. This step can have many sub-steps as &&s delay the resolution of macros. Note, as is shown in Fig. 23/24, the macro being Resolved can contain masked figures after all resolution step are completed.

The second step happens after all macro references are Resolved, and is the processing of macro functions. If the code contained %if %upcase(&Sco1) EQ %Str(G+W) the %upcase would be evaluated after all macro references (the & stuff) had been Resolved. (no example given here.) The third step is the actual evaluation of the truth, or falseness, of the %if statement. The Log shows this result. These rules can be checked by examining the logs in Figures 23 and 24.

7A) THE EFFECT OF %STR ON TOKENS GOING INTO, AND TOKENS COMING OUT OF, THE MACRO CATALOG

Earlier on, Figure 22 showed the use of the %Str function to mask commas following a %if %then. Figure 25 shows an enlarged Macro Catalog after compiling the macro shown in Figures 21 and 22.

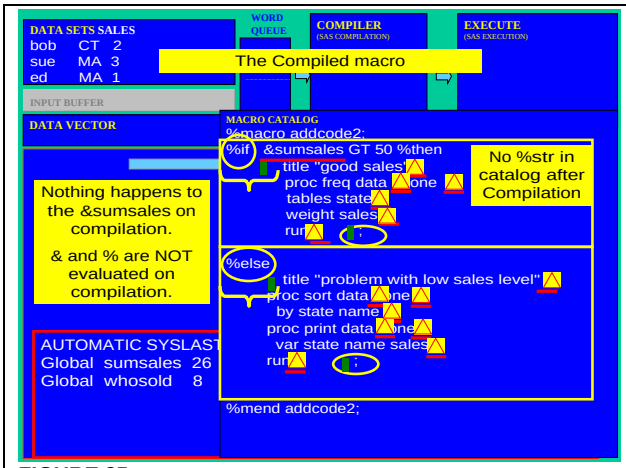

FIGURE 25

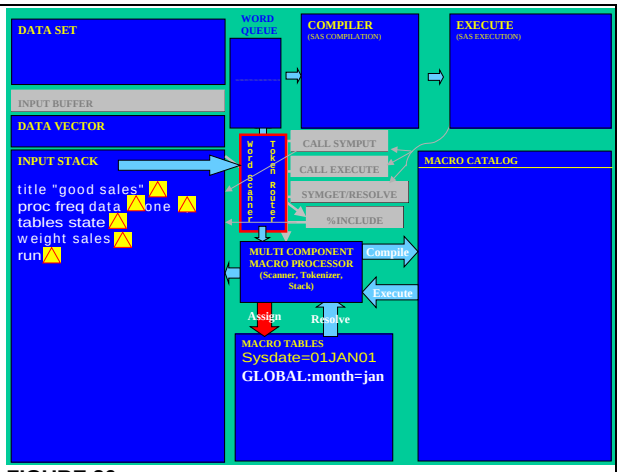

FIGURE 26

On the input stack (Figure21), the %Str can be seen but the %Str token is not stored in the catalog (see Figure 25). %Str and %NRStr have their effect as tokens go into storage areas. What is stored, in the storage areas, is the tokens after %Str and %NRStr have done their work (see Figure 25). Note that the tokens ; and = are all masked. Also note the leading and trailing rectangles that indicate to SAS what type of masking was performed.

It is suggested that if the macro Addcode2 were executed, the tokens would not unmasked as they pass through the Macro Processor on their way to the Input Stack. It would follow that if &sumsales were greater than 50, the text put on the Input Stack would look like that shown in Figure 26 and that the masking would be removed by the automatic unmasking barrier in the Word Queue.

7B) THE EFFECT OF %NRSTR ON TOKENS GOING INTO, AND TOKENS COMING OUT OF, THE MACRO CATALOG

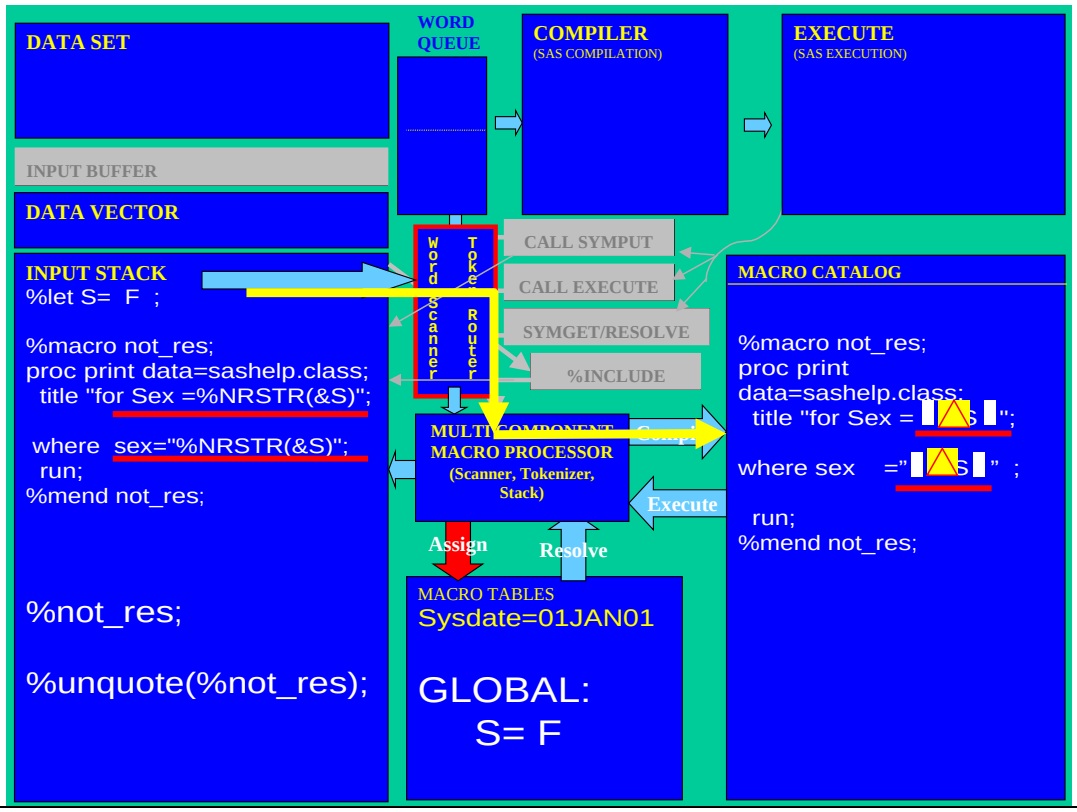

Figure 27

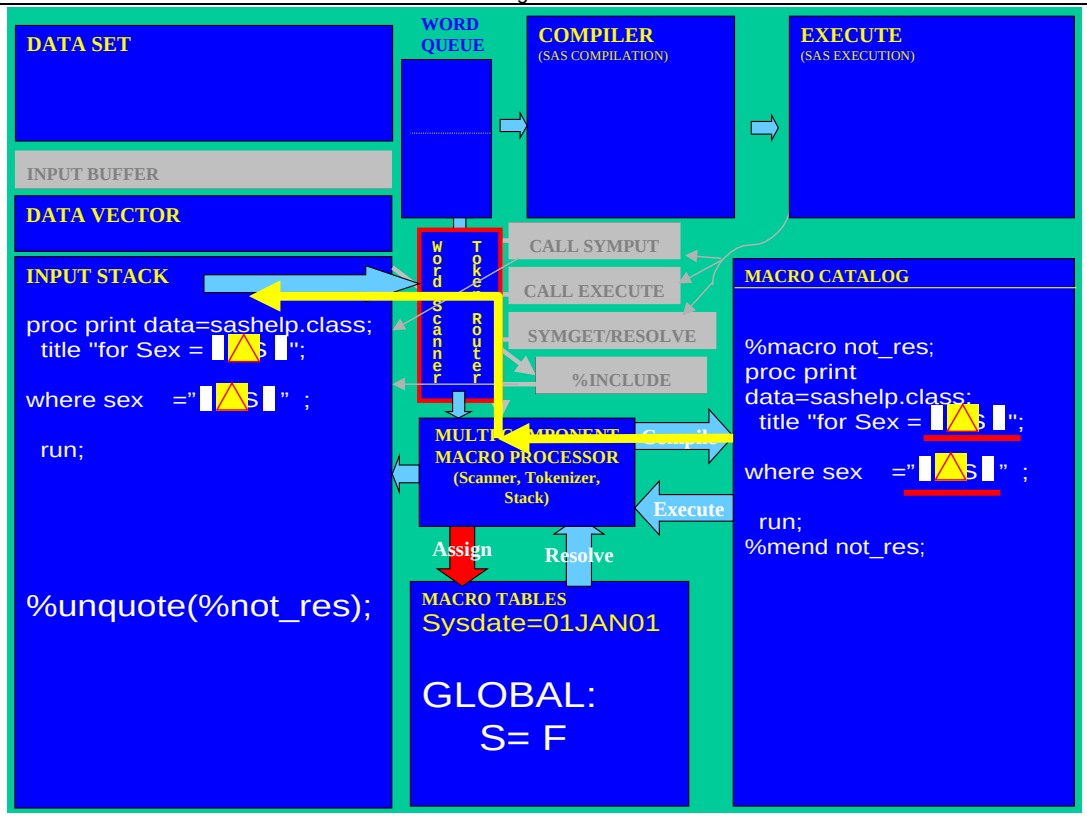

FIGURE 28

The effect of the %NRStr in a % If statement is illustrated in Figure 23. Figure 27 shows the effect of the %NRStr on macro invocations in code. The %NRStr masks the ampersands on compilation. This is a bit of overkill since one does not need to mask an & on Macro Compilation. An & is not Resolved as it goes into the Macro Catalog.

In Figure 28 we see taht when the macro executes, the masked tokens are be put on the input stack and not unmasked until the macro unmasking barrier. This causes a problem. The %unquote, working it's way up the input stack would allow the ampersands to Resolve and is explained below.

For a program to function, the programmer might need tokens to be "in a masked condition" as the tokens come out of the Macro Catalog. Manuals say execution functions mask as tokens flow out of storage, and could therefore be used in this situation. However, the programmer might still decide to use a use %Str or %NRStr and mask the tokens as they flow into the catalog. If the tokens at masked as they are put into storage, they will be masked as they come out. If the tokens are masked as they are put into the storage they will not need a masking on Macro Execution/Resolution. It can be programmer preference to mask as tokens go into or out of storage (Figure 27).

7C) MANUAL UNMASKING

When the macro in Figure 28 executes, the ampersands remain masked as they come out of the Macro Catalog. They are not evaluated on passing through the Macro Processor. Ampersands are put on the input stack with masking intact, as can be seen in Figure 28. When the tokens flow through the WS/TR the masked ampersands do not trigger any rules in the WS/TR. They are put on the Word Queue and the ampersands are unmasked at the Automatic Unmasking Barrier. The SAS Compiler has no facility for resolving macro variables and the ampersands are passed on to SAS Execution. The Where clause passed on to the compiler and execute modes is:

```
where upcase(sex)="&Sex";
```

This is syntactically valid and while it produces no notes, warnings or errors; it also produces no observations as shown below.

```
NOTE: No observations were selected from data set SASHELP.CLASS.
NOTE: There were 0 observations read from the data set SASHELP.CLASS.
      WHERE UPCASE(sex)='&S';
```

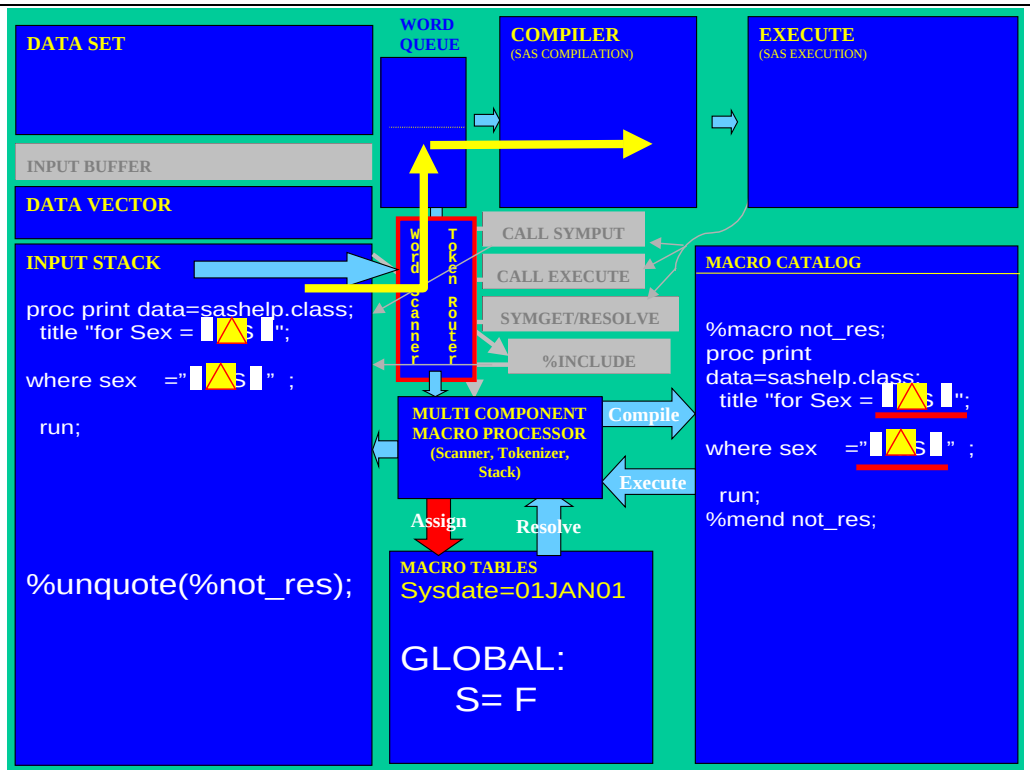

FIGURE 29

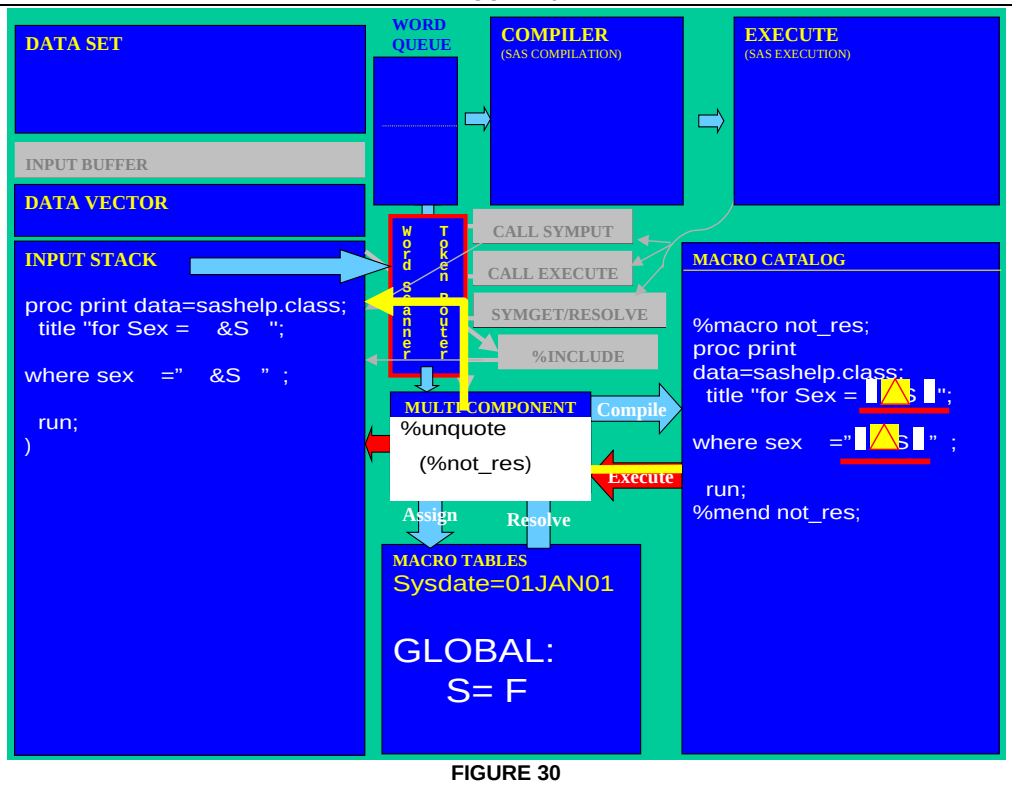

FIGURE 30

If the macro were called with the statement
`%unquote(%not_res);`
(shown working up the input stack in Figures 27 – 29 and evaluating in Figure 30) the results are different.

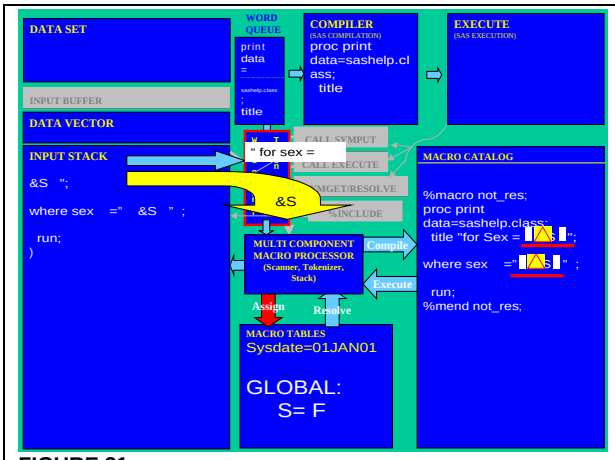

FIGURE 31

The macro evaluates and results are shown. It is suggested that the following process happens.

The %unquote function is read by the WS/TR and passed to the Macro Processor where it is held in a stack pending execution.

It is proposed that un-quoting is a Macro Processor capability that happens inside the Macro Processor as tokens move towards the Input Stack, as is shown in Figure 28. Tokens are returned from the Macro Catalog in between the parenthesis of the %unquote function- which then executes inside the Macro Processor

Unmasked tokens are put on the Input Stack and processed normally. When a macro trigger reaches the top of the Input Stack (Figure 29) it is passed to the Macro Processor for resolution.

CONCLUSION

It is thought that the Map of the SAS system is a useful method for explaining the complex SAS macro process.

REFERENCES

It is suggested that interested readers study the books by Aster, Carpenter, and Berlew.

Lavery, Anbari & Nsereko (2002), "An Animated Guide, The Map of the SAS Macro System" In the Proceedings of the 15th North East SAS Users Group Conference, 220-228

ACKNOWLEDGMENTS

Special thanks must go to Dr. Ian Whitlock for time spent reviewing this and for his help to the SAS community. Additional thanks go to the authors of the many excellent SAS books on macros and to personnel of SAS institute.

CONTACT

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

Russell Lavery Contractor for Numeric Resources
Ardmore, PA 19003,
Email: russ.lavery@verizon.net