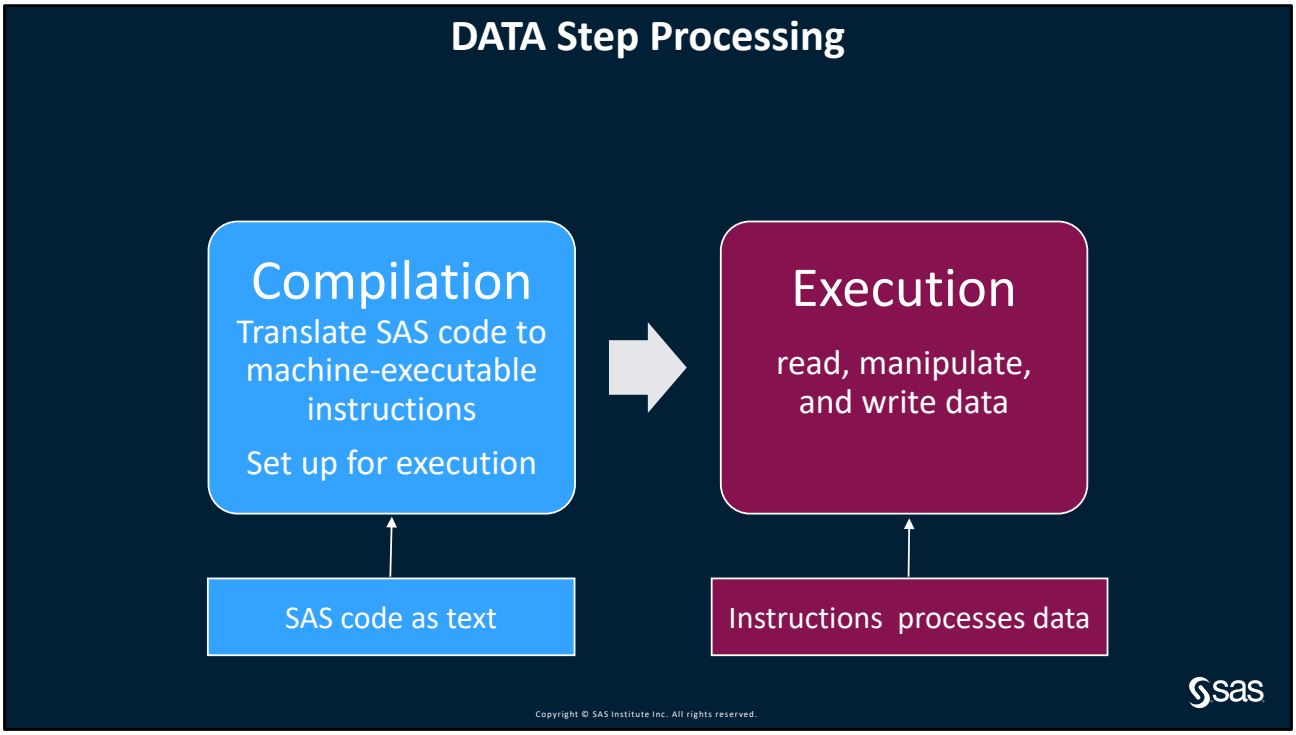
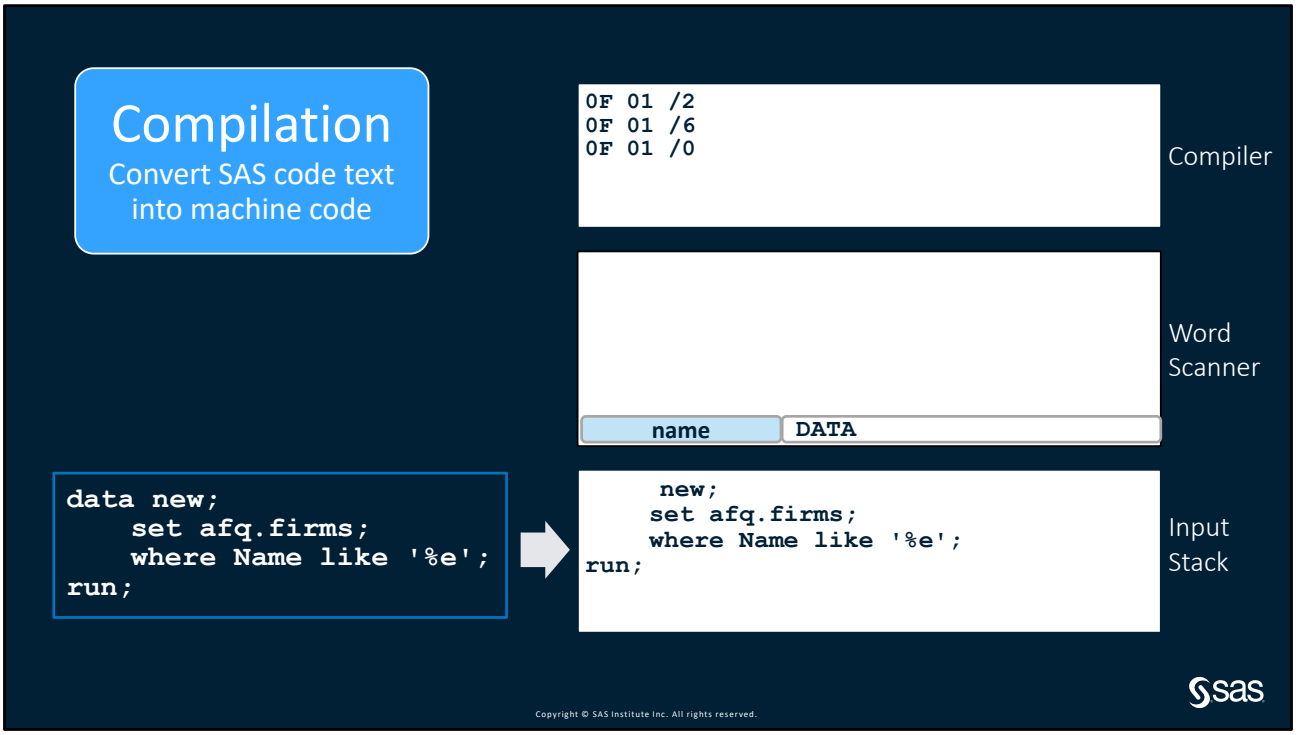
A Fine Quotation: Quoting in SAS® Macro Programming

Mark Jordan

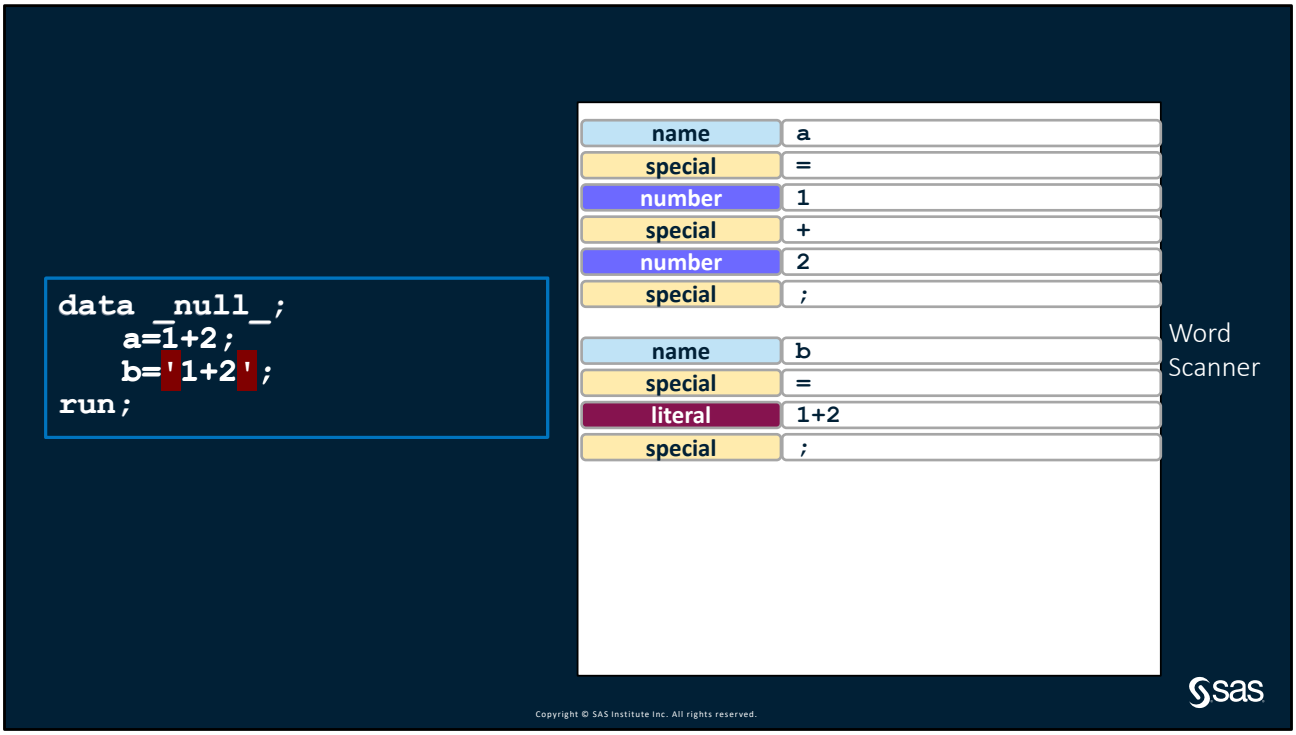
"Misquotations are the only quotations that are never misquoted."
Hesketh Pearson



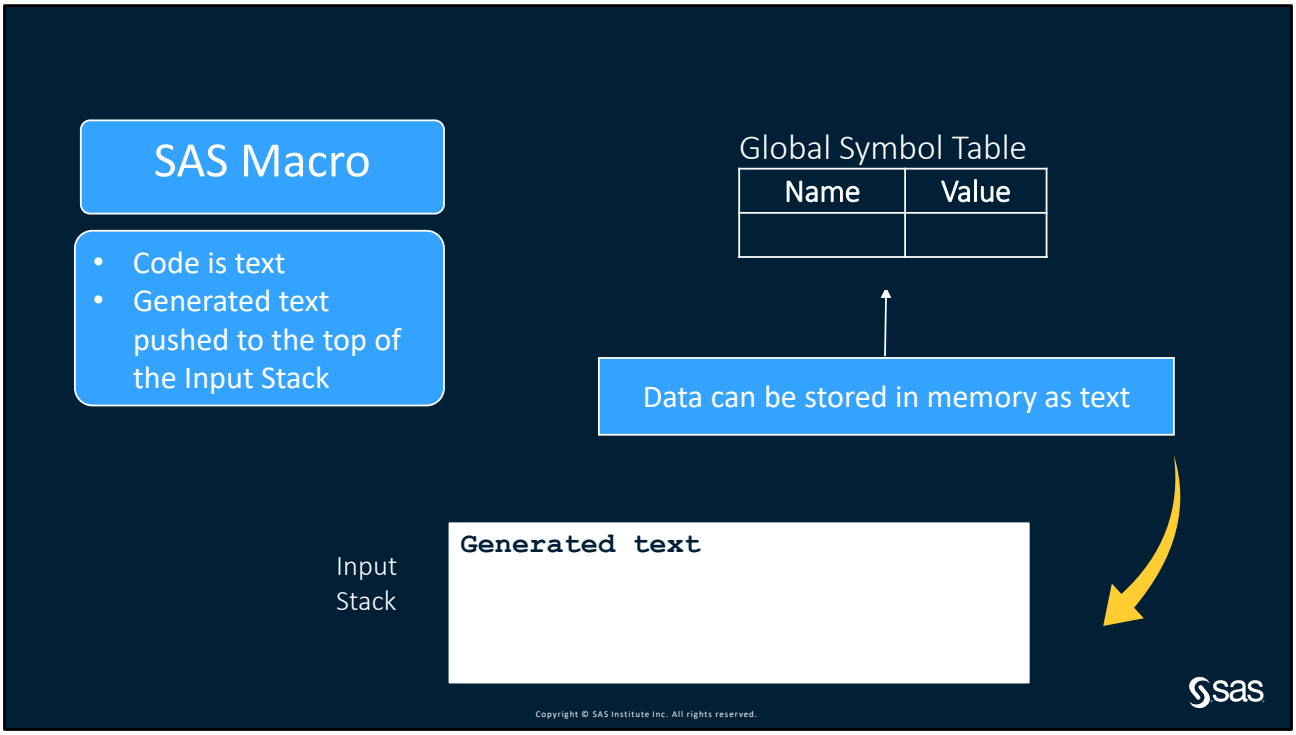
When we run a DATA step, the code we write goes through two phases: compilation and execution. In the compilation phase, our SAS code text is translated into machine-executable instructions, the system resources required for execution are reserved, and the memory locations for holding the data to be processed established. We call that memory area the Program Data Vector, or PDV for short.Finally, in the execution phase, the machine-executable instructions read, manipulate, and write data.



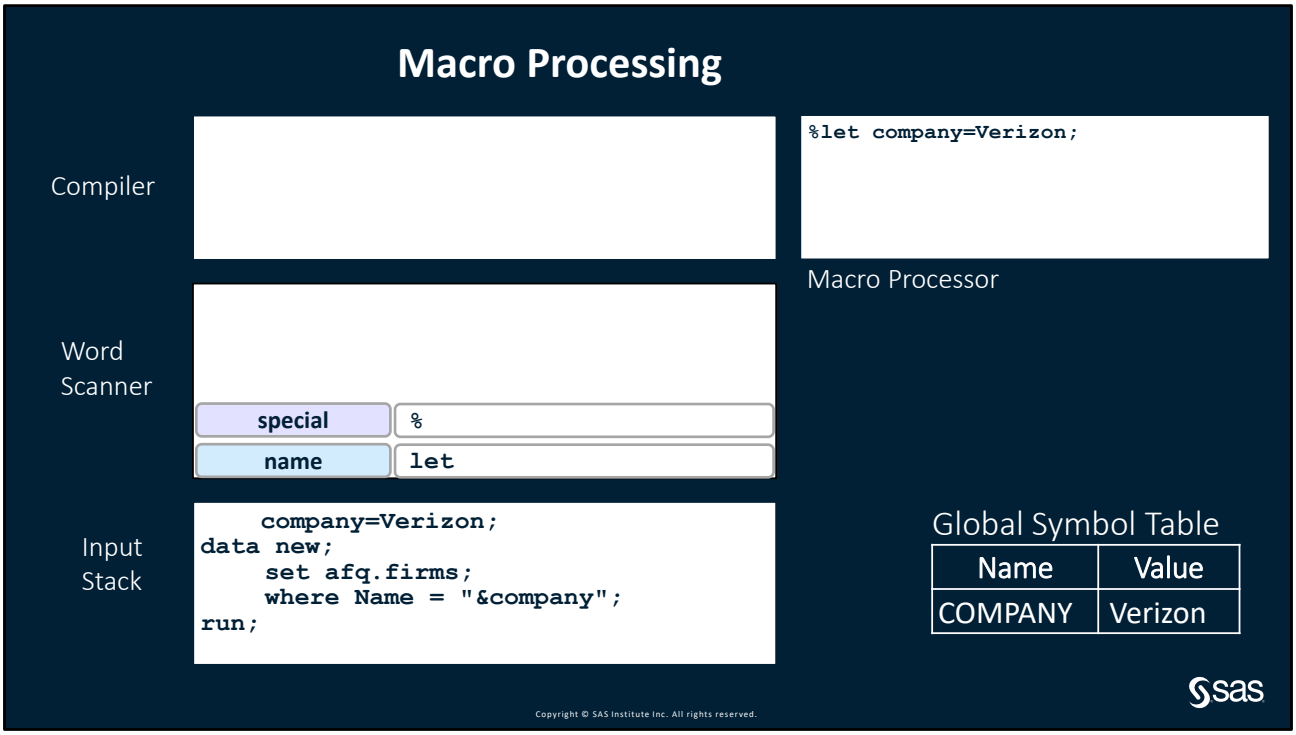
During compilation, the text of our SAS code must be translated into machine-executable instructions. When we run our code, the SAS code text is sent to an area in memory called the input stack. The input stack serves as input to the Word Scanner, which breaks up the text received from the Input Stack into digestible bytes called tokens. Tokens are a combination of text with a bit of metadata added to help the Compiler determine how best to translate the token into machine-executable instructions.



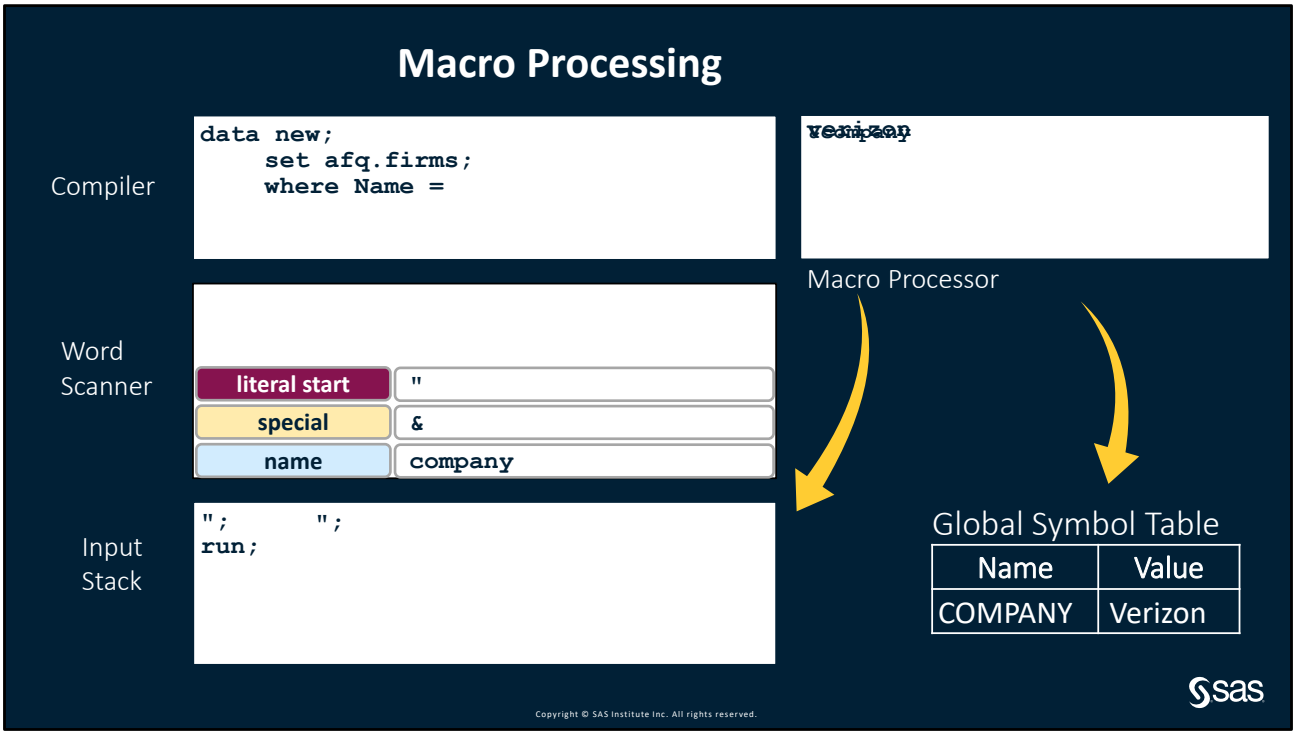
When we look at the tokenization of these two SAS statements, it's easy to see how the SAS Compiler knows how to proceed. The plus sign special token in the first expression will be converted to machine-executable instructions which add the two numeric values, while the plus character in the literal token will be handled as data - just another character in the literal string. It's all in the metadata – and those quotation marks in the second statement were crucial to making the correct interpretation.



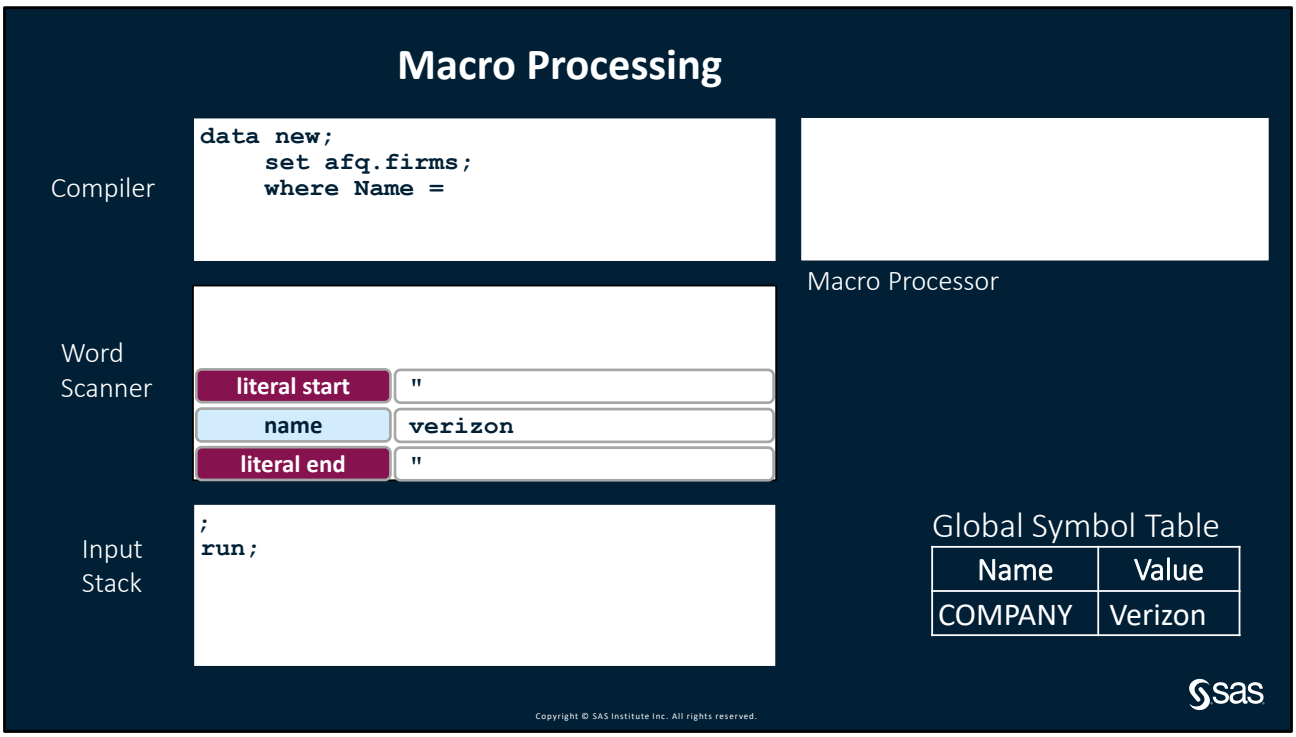
The macro processor is separate from the SAS Compiler. The Macro processor is not aware of, nor does it have access to, the PDV. In the macro world, everything is text. Your code is written as text. Values manipulated by the macro code are also text. And the output of a macro process is – you guessed it – more text – which is pushed onto the top of the input stack.



A special token containing a percent sign or ampersand, followed immediately by a name token, is called a macro trigger. A macro trigger causes the Word Scanner to pass text to the macro processor for processing. This macro statement merely stores some text in memory. Note that the Symbol Table does not make provision for identifying the data type of the value stored. Everything in macro is text.



An ampersand followed immediately by a name token is a macro trigger known as a variable reference. When the macro processor receives a variable reference it looks for the variable in the Symbol Table, and if found, retrieves the text stored there. This text is pushed to the top of the input stack, and

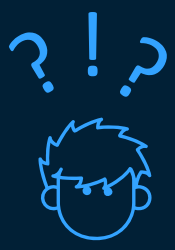


Tokenization continues. Note that the text passed to the macro processor doesn't include any clues about how to treat the text. Quotes are just another character in macro – so there's no easy way to tell a literal value from code.

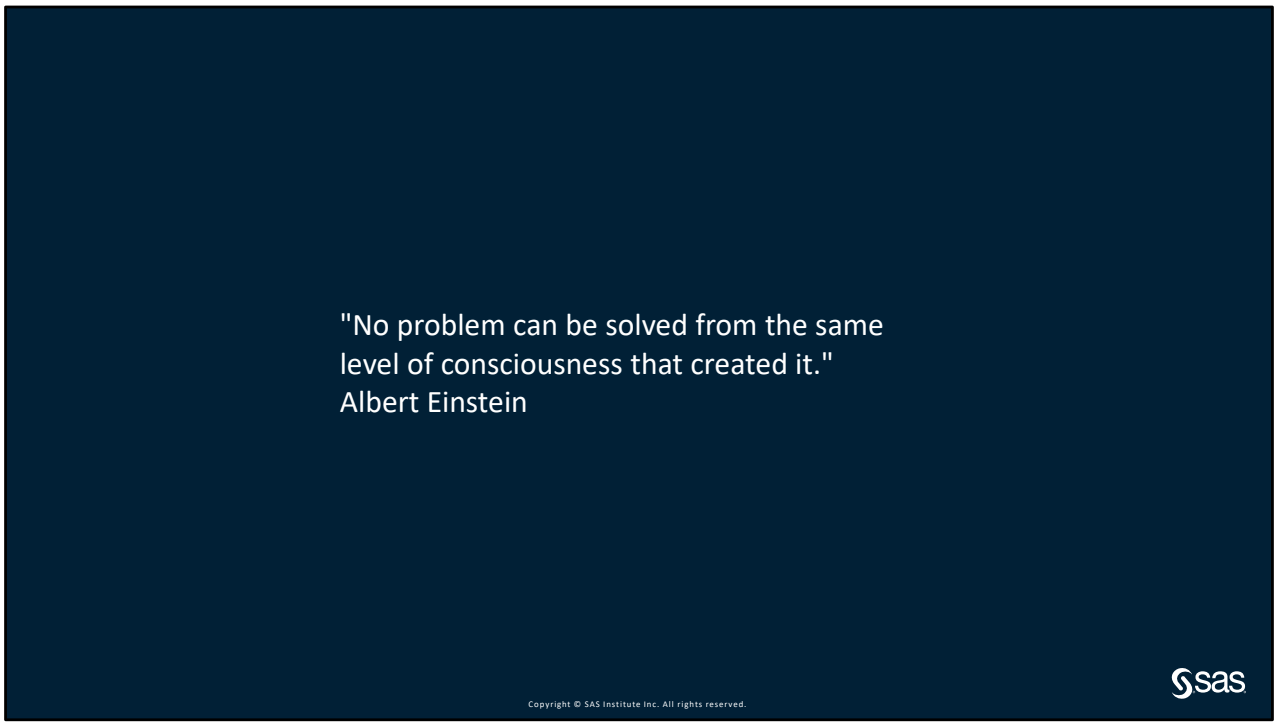
The Macro Dilemma

Global Symbol Table

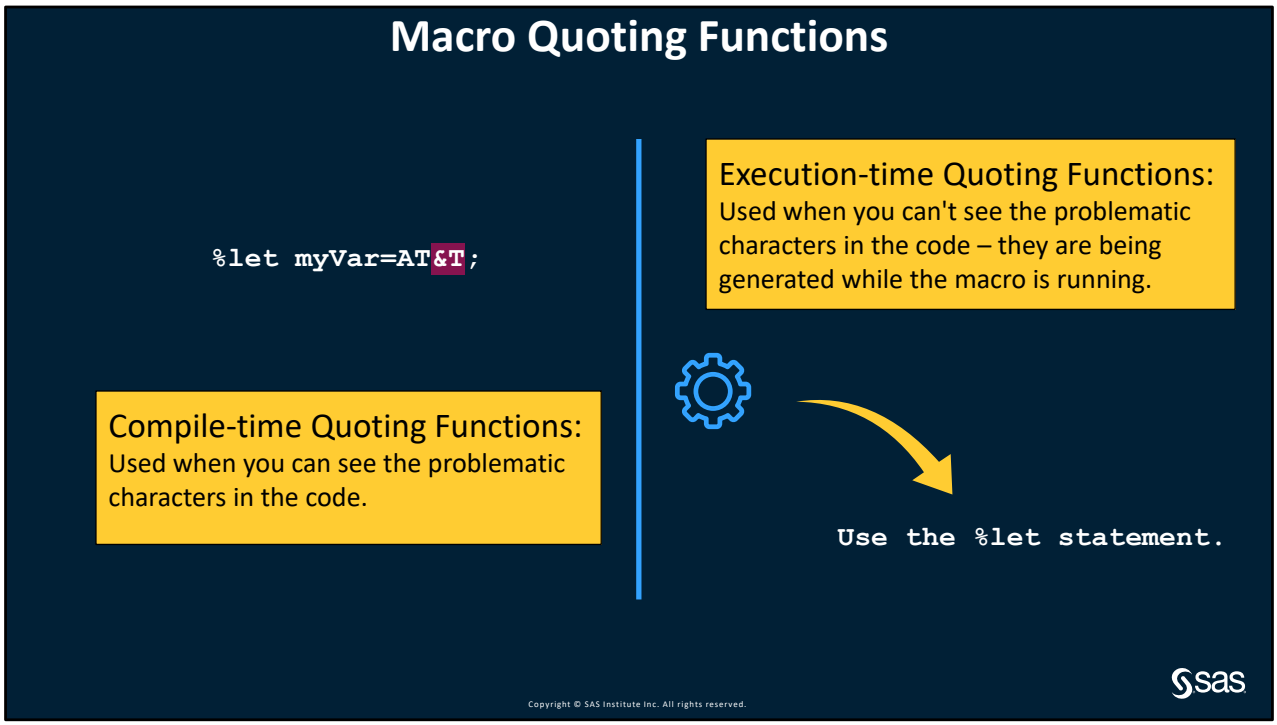
Name	Value
V1	AT&T
V2	Let's eat!



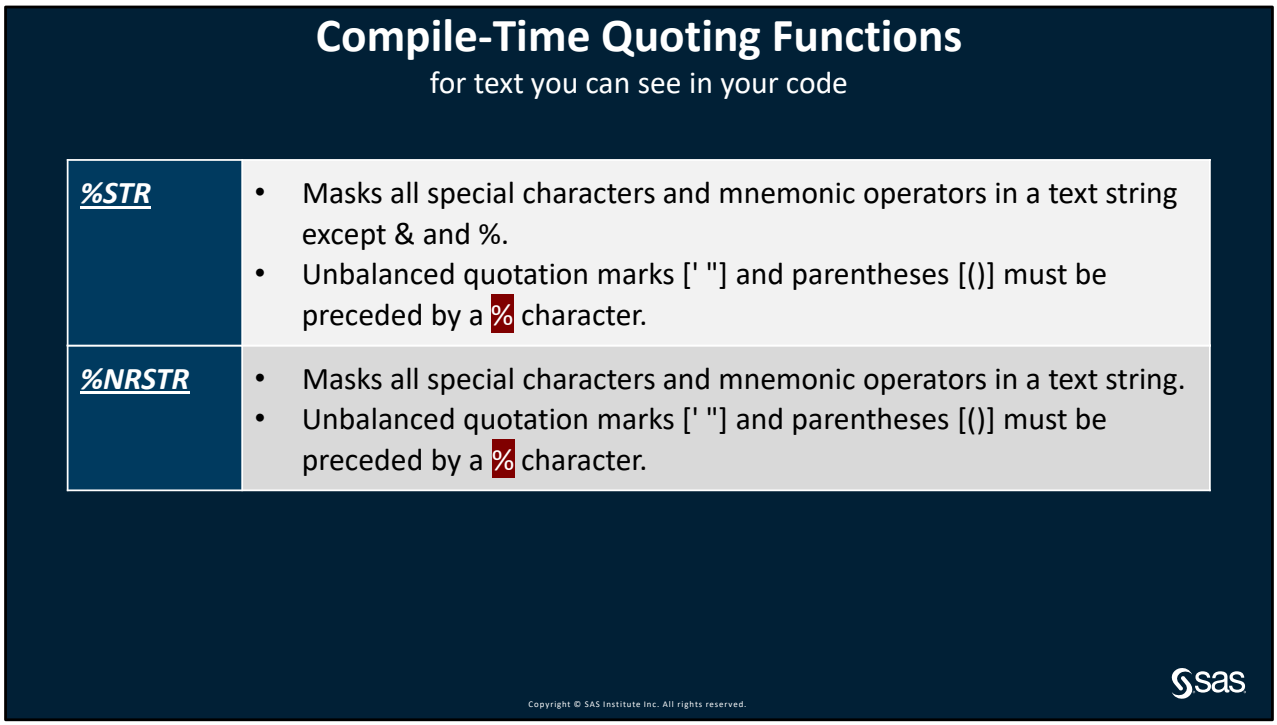
If macro code is text, and macro data values are text, and quotes are just another character, then how can macro distinguish a literal value from code? It's a real dilemma.



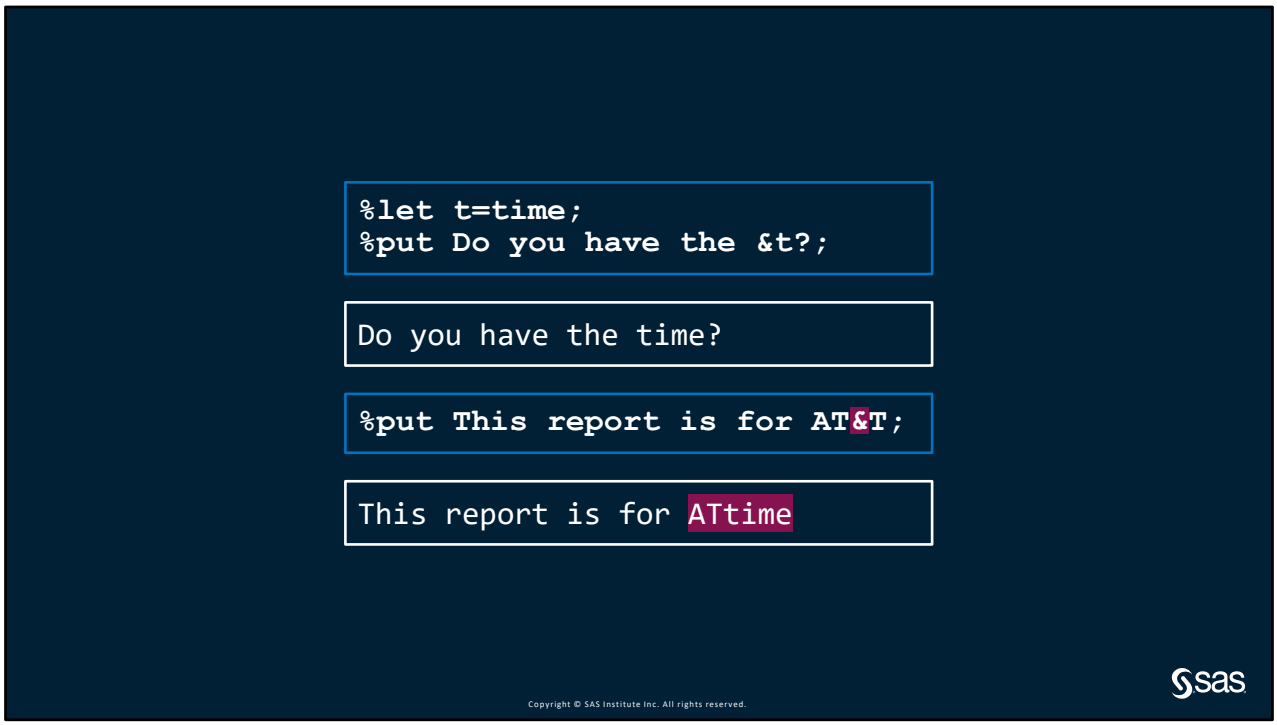
If we can't resolve this dilemma with quotation mark characters, surely there is someway we can help the macro processor make the correct decision about the text in our code? That one-size-fits-all, 'quotes or no quotes' method that worked so well in DATA step won't work here. In macro, quoting is much more nuanced, so we should expect the techniques available to be more varied and nuanced, too.



Macro uses quoting functions instead of quotation marks to ensure that text is treated as data instead of code. The functions we choose are differentiated by WHEN we want the text to be quoted – that is, at what point do we want the macro processor to stop rummaging through the text to see if there is any more executable macro code in there? The category of quoting function we'll need is based on if the text to be quoted is hard-coded in the macro program, or if it will be resolved or produced while the macro processor is processing our text. When quoting the former, we'll be able to see the problematic characters in the code, and we'll need compile-time quoting functions to mask them. If you can't see the problematic text in the code, it's being generated during execution, and we'll need "execution-time" quoting functions to solve that problem.



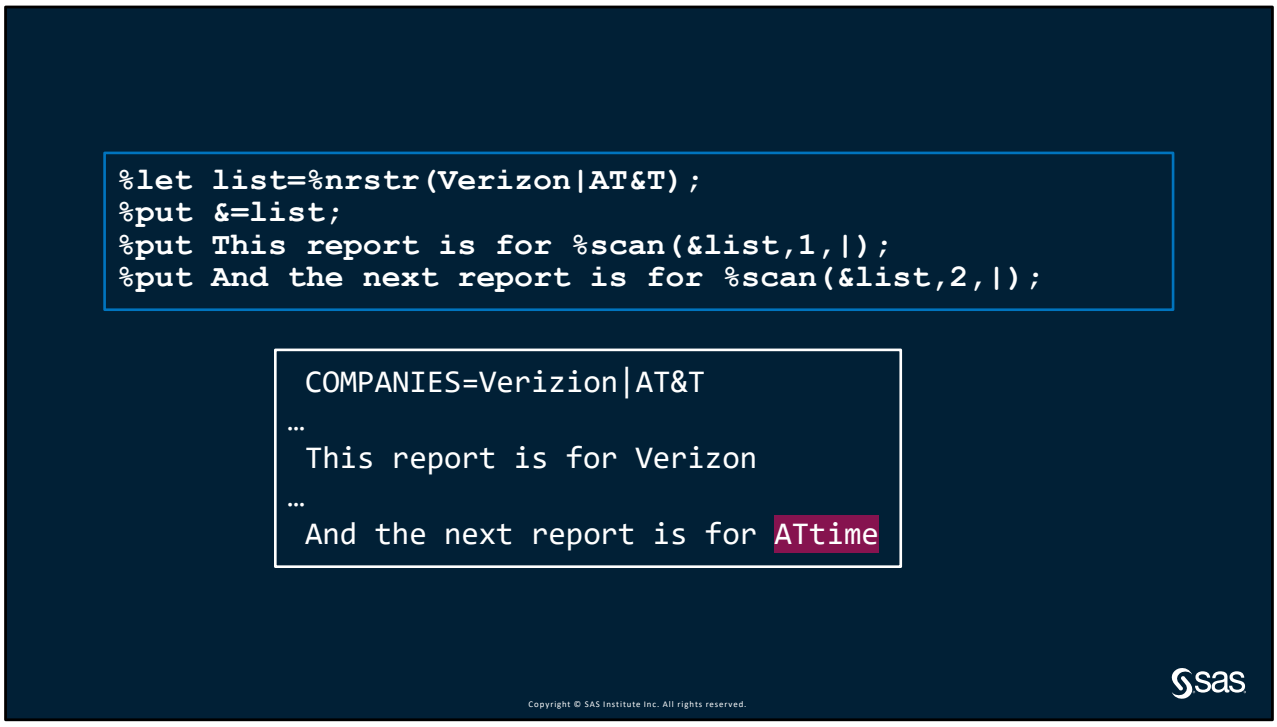
As we write our programs, we'll need to decide if we want macro triggers to resolve before the text gets quoted, or if we just want the macro processor to leave our text as-is. This is kind of analogous to using double or single quotes in the DATA step. If your DATA step would have used double quotes to allow macro triggers, in macro the %STR, or "string", function is a good choice. If instead you would have used single quotes to make sure your text was processed unmolested, then the %NRSTR, or "non-resolving string", function will do the trick. Whichever you pick, it's important to remember to precede each instance of unbalanced quotation marks or parentheses with a percent sign so that those characters are handled properly.



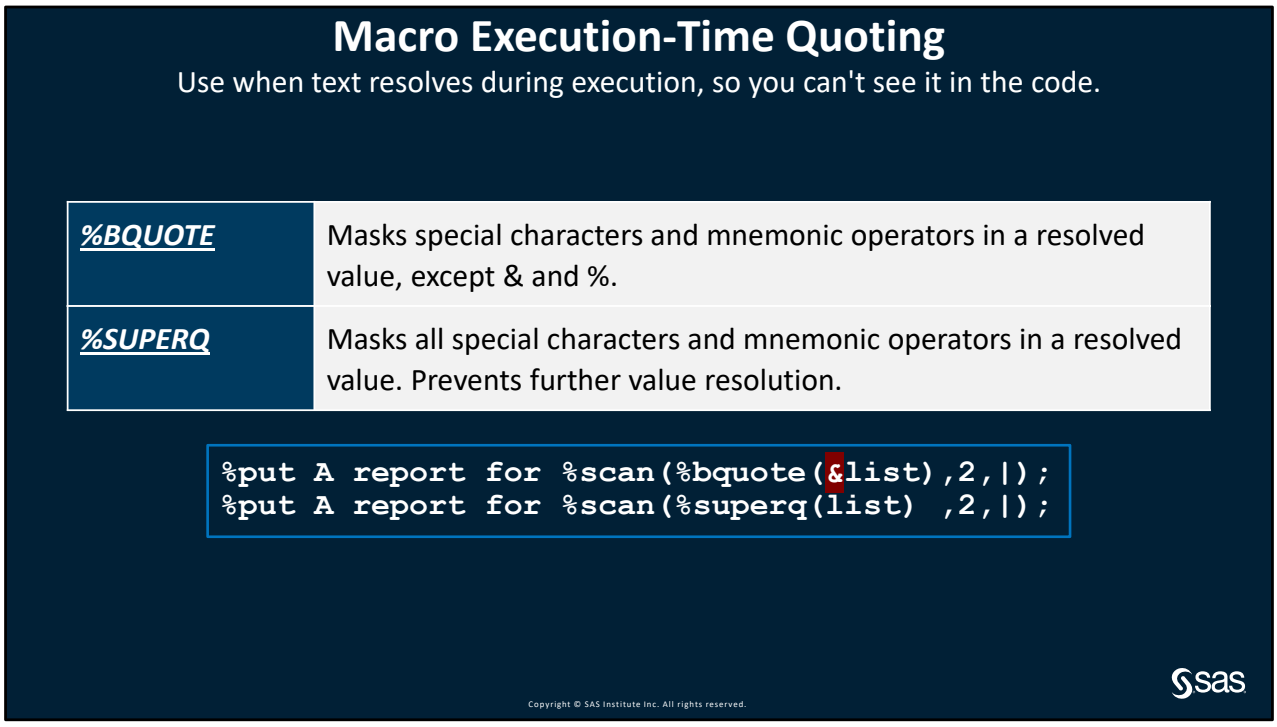
If I submit this macro code, I get the expected result in the log. If I later submit this macro code, the result isn't what I wanted. Macro is interpreting the ampersand as code, and does a symbol table lookup for the variable named T. I just wanted it to treat the ampersand like a text character and write it to the log.

"The wisdom of the wise, and the experience of ages, may be preserved by quotation."
Isaac D'Israeli

D'Israeli says, our text may be preserved by a quotation – or in this case, a quoting function. Because I can see the problematic character in my code, I should use %STR in this instance.



Here's a different dilemma. I intend to generate pipe-delimited list of company names and generate a report for each one. I'll eventually write a loop for this, but I'll test my code first. When I run the program, the last %PUT statement didn't get me the result I expected for AT&T. Which is weird, because that first %PUT statement worked just fine, demonstrating that the value of **list** was properly quoted when I stored it. It must have gotten unquoted somehow…



Well, when the problem statement executed, the offending character was not visible in the code. I'll need an execution-time quoting function for this, I think. Once again, I'll need to decide if I want macro triggers to resolve before the text is quoted, or if I just want the macro processor to leave my text alone. To allow macro triggers to resolve, I usually use %BQUOTE. But when I need to ensure my text is never mistaken for code, I'm a fan of %SUPERQ. Note that, unlike all other macro functions, %SUPERQ does not accept a text string as its parameter – it only accepts a macro variable name. Here, the ampersand in the function call will first resolve the **list** variable text, and then %BQUOTE will quote the result. There is no ampersand in %SUPERQ's parameter when called. SUPERQ retrieves the value straight from the symbol table variable itself – the text is not pre-resolved by the macro processor.

Summary: Macro Quoting Functions

	Macro Triggers (& %) Not Masked	Macro Triggers (& %) Masked
Constant text (compile time)	%STR	%NRSTR
Resolved values (execution time)	%BQUOTE	%SUPERQ

So, in summary, if you can see the problematic text in the code, use the compile-time functions %STR and %NRSTR. If the problematic text is not visible in the code, use the execution-time quoting functions, %BQUOTE or %SUPERQ.

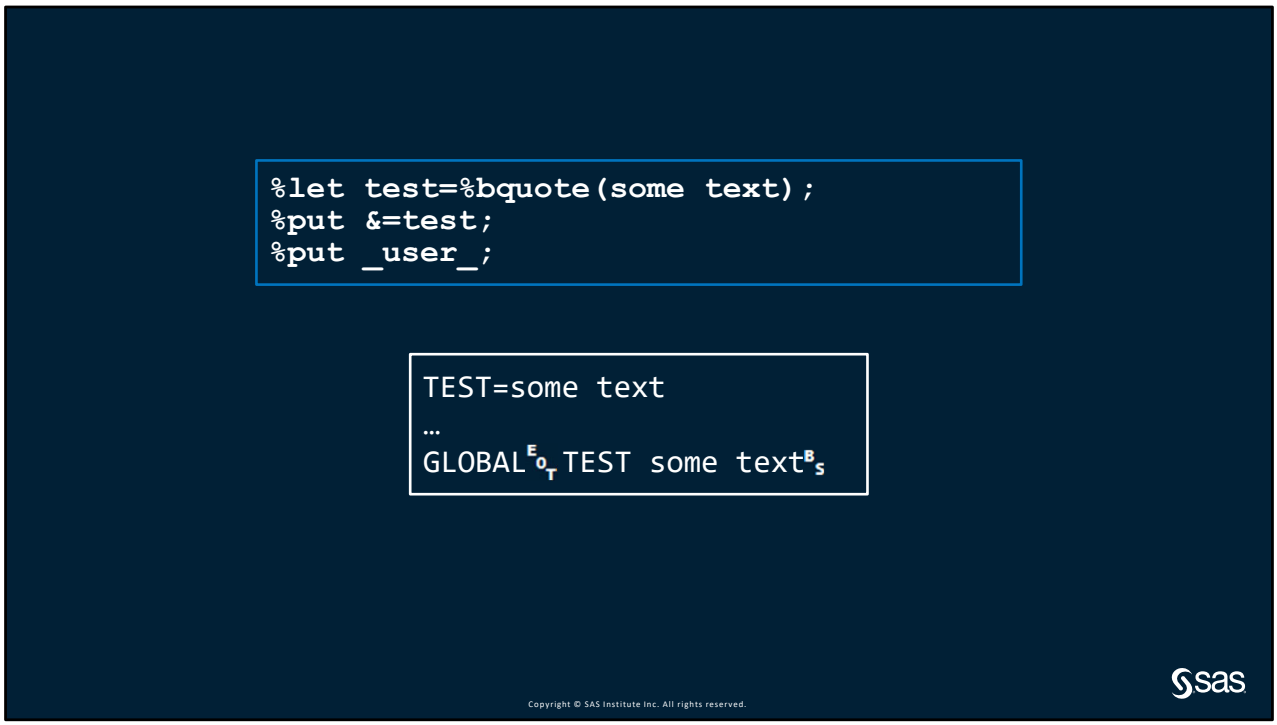
```
%put A report for %scan(%bquote(&companies),2,|);
%put A report for %scan(%superq(companies),2,|);
```

```
 A report for ATtime
…
 A report for Attime
```


Now WAIT a minute! What about *this* code? I thought %SUPERQ was the answer to all of my execution-time quoting problems. What's going on here?

"A fine quotation is a diamond in the hand of a man of wit and a pebble in the hand of a fool."
Joseph Roux

I think we need a better understanding of how this macro quoting stuff works under the covers!



A quoted macro value includes non-printable leading and trailing "delta characters" that the macro processor recognizes as a sign to leave the intervening text alone. In the macro world, these are the equivalent of those DATA step single quotes. At times during processing, the delta characters are automatically removed. In most cases the removal is appropriate, such as when the value is printed or is included in SAS code sent to the compiler. In fact, in SAS Studio, I haven't yet found a way to view those characters! But in Enterprise Guide or the SAS Windowing Environment, if you %PUT macro variables using a special list like _USER_, the values get dumped to the log as-is, with those non-printable delta characters represented, so we can better see how it works.

```
options symbolgen;
%put The list is %superq(list);
%put This report is for %scan(%superq(list),2,|);
%put This report is for
%scan(Verizon|%nrstr(AT&T),2,|);
options nosymbolgen;
```

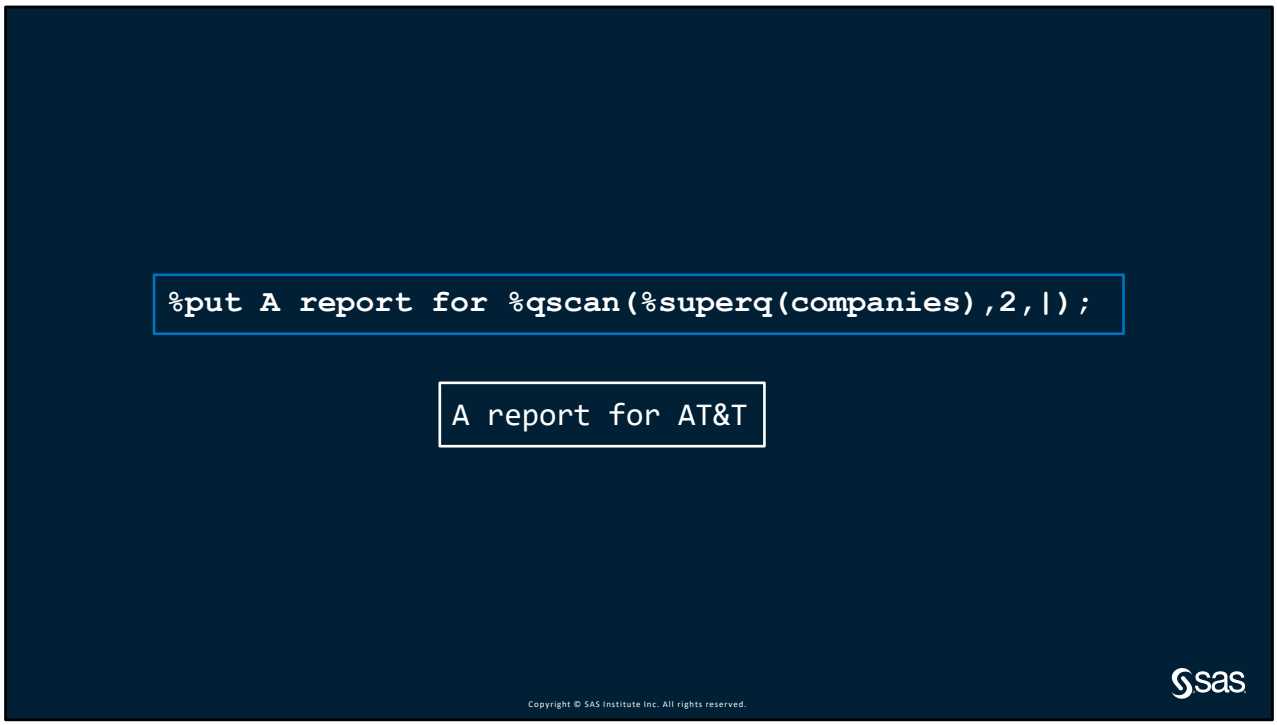
```
 The list is Verizon|AT&T
…
 SYMBOLGEN:  Macro variable T resolves to time
 This report is for ATtime
…
 SYMBOLGEN:  Macro variable T resolves to time
 This report is for ATtime
```


I can use the SYMBOGEN option to make the macro processor confess each time it resolves a macro variable. By submitting these code variants, I can see that the %SCAN function must be producing unquoted text, because &T in the output is being misinterpreted as a macro variable reference. When I think about it, I can see that the delta characters would probably need to be stripped during text processing in order to maintain proper character counts, etc. But having the unquoted text returned in this case just won't do…

Functions + Quoting

Function	*"Q" Version*	*Purpose (Q versions produce quoted text)*
%CMPRES	*%QCMPRES*	Compresses multiple blanks, strips leading and trailing blanks.
%LEFT	*%QLEFT*	Left-aligns an argument by removing leading blanks.
%LOWCASE	*%QLOWCASE*	Converts a value to lowercase.
%SCAN	*%QSCAN*	Returns the specified word from a string.
%SUBSTR	*%QSUBSTR*	Produces a substring.
%SYSFUNC	*%QSYSFUNC*	Executes Base SAS functions.
%TRIM	*%QTRIM*	Trims trailing blanks.
%UPCASE	*%QUPCASE*	Converts a value to uppercase.

Fortunately, there are quoting versions of every text processing function available. Some are implemented as actual macro functions, other functions ship as part for the AUTOCALL library. From a programmer's perspective, we use them all the same. Each function does what you would expect with the arguments you would expect, but the text they return is already quoted, so it will never inadvertently trigger additional processing in the macro processor.



%QSCAN is just what I need to make my code work as expected!

"Quotation, like much better things, has its abuses."
Isaac D'Israeli

On a rare occasion, you will run into a "darned if you do, darned if you don't" situation with macro quoting. In those cases, the same macro text may need to be quoted in one part of the program and unquoted in another. I'll use a somewhat contrived example so we can clearly see the problem.

```
data _null_;
   call symputx('FIRMLIST','"Lands'' End", "Armani",'||
                '"Chipotle", "Ruth''s Chris Steakhouse",||
                '"Mattel"');
run;
%put &=firmlist;
```

```
FIRMLIST="Lands' End", "Armani","Chipotle", "Ruth's Chris Steakhouse","Mattel"
```


First, I'll use a DATA step to store a comma-delimited list of company names in a macro variable named **firmlist**. The DATA step knows nothing about delta characters, so the value just stored is as-is.

```
title "Firms: &firmList";
proc print data=afq.Firms;
   where Name in(&firmList);
run;
```

```
title "Firms: &firmList";
WARNING: The TITLE statement is ambiguous due to invalid options or unquoted
text.
NOTE: Line generated by the macro variable "FIRMLIST".
"Firms: "Lands' End", "Armani","Chipotle", "Ruth's Chris Steakhouse","Mattel"
NOTE: The meaning of an identifier after a quoted string might change in a
future SAS release.  Inserting white space between a quoted string and the
succeeding identifier is recommended.`
```


Next, I'll use the list in some SAS code. The program uses the list in the TITLE statement and in the PROC PRINT WHERE statement. When I run this code, the compiler complains. The text resolved inside the TITLE statement's double quotes also contains double quotes, that that's messing up the tokenization.

```
%let firmList=%bquote(&firmList);

title "Firms: &firmList";
proc print data=afq.Firms;
   where Name in(&firmList);
run;
```

```
MPRINT(FIRMLIST):   proc print data=afq.Firms;
MPRINT(FIRMLIST):   where Name in("Beef 'O' Brady's","Crate & Barrel","Lands'
End","Ruth's Chris Steakhouse");
MPRINT(FIRMLIST):   run;
NOTE: No observations were selected from data set AFQ.FIRMS.
NOTE: There were 0 observations read from the data set AFQ.FIRMS.
      WHERE 0 /* an obviously FALSE WHERE clause */ ;
```


Let's store the list as quoted text using %BQUOTE. When I run this code, the TITLE statement works fine, but now the delta characters in **list** don't sit well when the compiler processes the WHERE statement.

Unquoting

Use when quoted text needs to be executed it as code.

%UNQUOTE	Unmasks all special characters and mnemonic operators

Now, there are various approaches we could have used to avoid overquoting. But if by some off chance you wind up with over-quoted macro text, you can use the %UNQUOTE function to remove the delta characters yourself.

```
%let firmList=%bquote(&firmList);

title "Firms: &firmList";
proc print data=afq.Firms;
   where Name in(%unquote(&firmList));
run;
```

Firms: "Lands' End", "Armani","Chipotle", "Ruth's Chris Steakhouse","Mattel"

Obs	Name	Type	MktCap
3	Lands' End	Clothing	$275,880,000.00
4	Armani	Clothing	$478,629,900.00
8	Chipotle	Restaurant	$42,089,200,000.00
9	Ruth's Chris Steakhouse	Restaurant	$594,320,000.00
10	Mattel	Toys	$7,037,825,000.00

When I process the text resolved from **list** with the %UNQUOTE function before the compiler sees it, the result complies and executes without error.

Other Macro Quoting Functions

For backwards compatibility or rare niche problems

%QUOTE	• Masks all special characters and mnemonic operators in resolved text except & and %. • Unbalanced quotation marks [' "] and parentheses [()] must be preceded by a % character.
%NRQUOTE	• Masks all special characters and mnemonic operators in resolved text. • Unbalanced quotation marks [' "] and parentheses [()] must be preceded by a % character.
%NRBQUOTE	• Masks all special characters and mnemonic operators in resolved text. • Unbalanced quotation marks [' "] and parentheses [()] do not need to be marked.

There are other macro quoting functions, mostly useful for backwards compatibility in older SAS programs. I list them here for completeness, but don't fell like you'll need them very often. I haven't in my own work.

The demonstration code was created and executed on SAS on Demand for Academics. Access to SAS on Demand for Academics is free for *anyone* to use for learning more about SAS. If you already have a SAS profile, just log in with those credentials.
Log in to SAS On Demand for academics here and choose SAS Studio as your interface:
https://welcome.oda.sas.com/login

To grab the programs and data, copy this code into the editor window then run the code:

```
filename macro url "https://raw.githubusercontent.com/SASJedi/sas-macros/master/unpackzip.sas";
%include macro;
options dlcreatedir;
libname x "~/Ask the Expert";
libname x "~/Ask the Expert/A Fine Quotation - Macro Quoting"; libname x clear;
options nodlcreatedir;

%let path=~/Ask the Expert/A Fine Quotation - Macro Quoting; filename localzip "&path/local.zip" ;
proc http url=
 "https://github.com/SASJedi/presentations/raw/master/A%20Fine%20Quotation%20-%20Macro%20Quoting.zip"
  out=localzip;
run;
%unpackzip(&path,local.zip)/* Activate some useful macros */%include "&path/autocall/*.sas";
```

After getting the programs and data, run the setup.sas program found in the folder named "A Fine Quotation - Macro Quoting" to set up the system. Now you're ready to run the demo code yourself!

Questions?

Mark Jordan
Principal Technical Training Consultant
Williamsburg, Virginia, USA
mark.jordan@sas.com

sas.com