

SAS070-2024

A Fine Quotation: Macro Quoting in SAS®

Mark Jordan, SAS Institute Inc.

Abstract

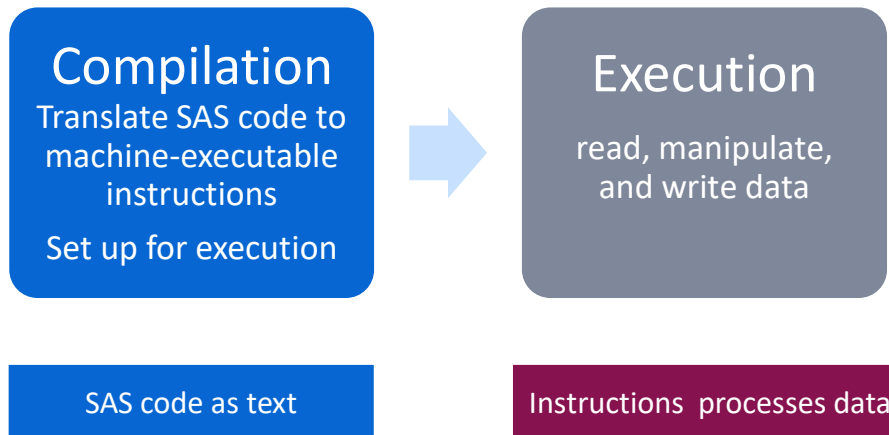
The SAS macro facility provides a powerful, programmable text processing tool for flexible, data-driven programming. But when your macro definition doesn't generate the SAS code you expected, troubleshooting can be a challenge. The problem is often caused by underquoting or overquoting macro variable values in your code. In this paper, I explain quoting in macro programs and demonstrate the effects of underquoting and overquoting. I'll cover effective troubleshooting techniques and some every day "rules of thumb" you can use to avoid quoting problems in your macro programs.

Introduction

In this paper I'll discuss:

- A brief review of code tokenization in SAS
- A review of macro triggers and macro processing
- An overview of compile-time and execution-time macro quoting functions
- How macro quoting works
- Examples of macro quoting gone wrong
- Troubleshooting and correction techniques for macro quoting
- Quoting thumb rules for macro programmers

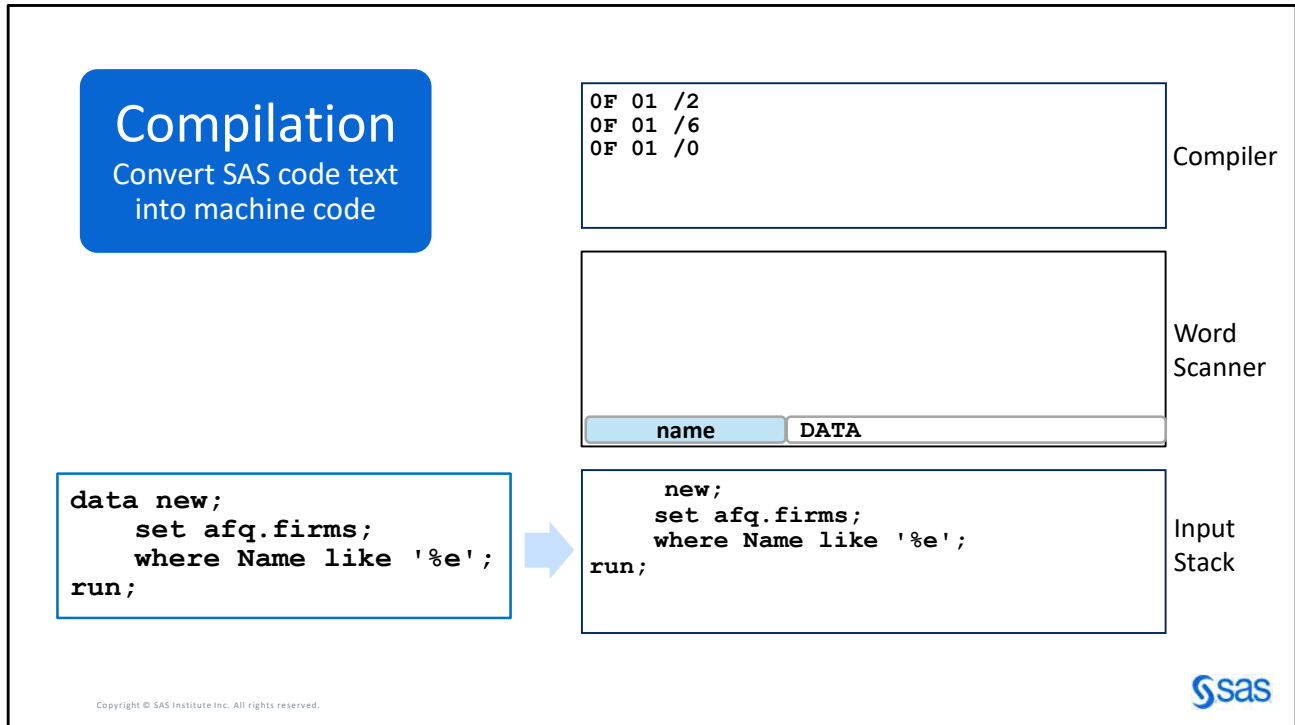
DATA Step Processing



Copyright © SAS Institute Inc. All rights reserved.



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.



When we submit our code, the text in the SAS code editor 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.

```
data _null_;
  a=1+2;
  b='1+2';
run;
```

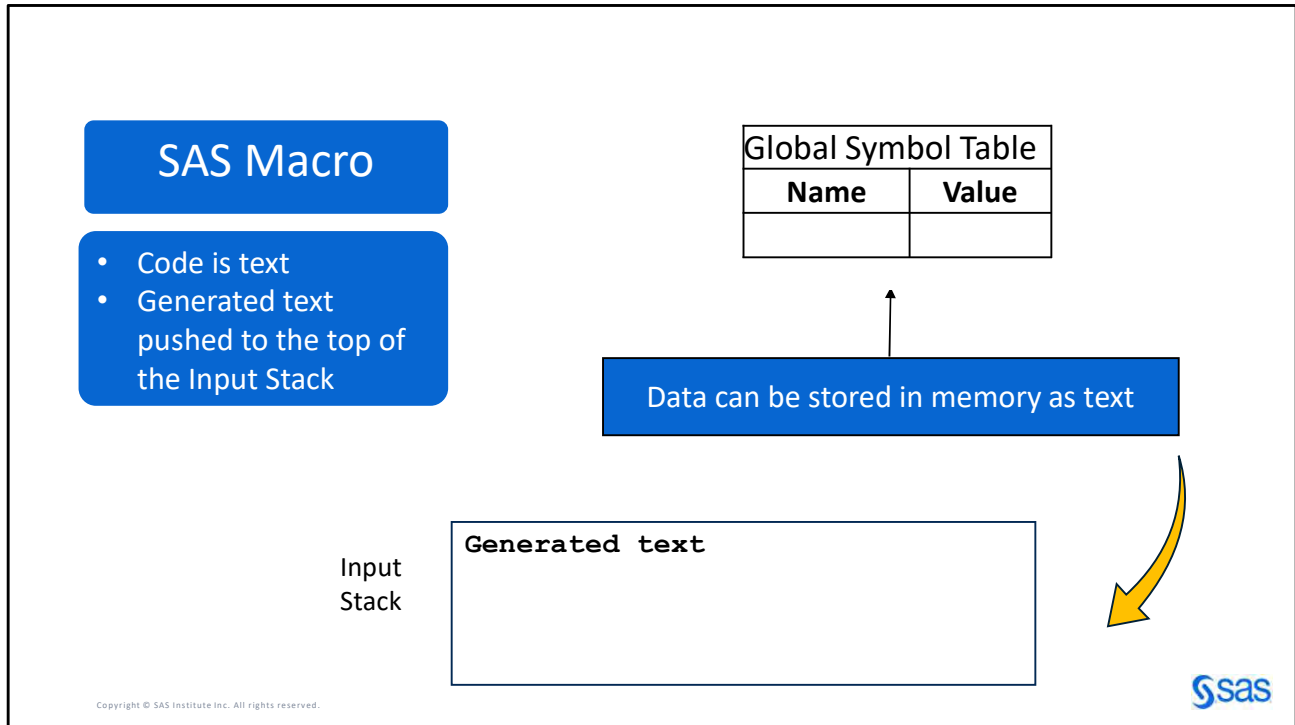
name	a
special	=
number	1
special	+
number	2
special	;
name	b
special	=
literal	1+2
special	;

Word
Scanner

Copyright © SAS Institute Inc. All rights reserved.



When we look at the tokenization of these two assignment 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 code to 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 for tokenization.

Macro Processing

Compiler

```
%let company=Verizon;
```

Macro Processor

Word
Scanner

special	%
name	let

Input
Stack

```
company=Verizon;  
data new;  
  set afq.firms;  
  where Name = "&company";  
run;
```

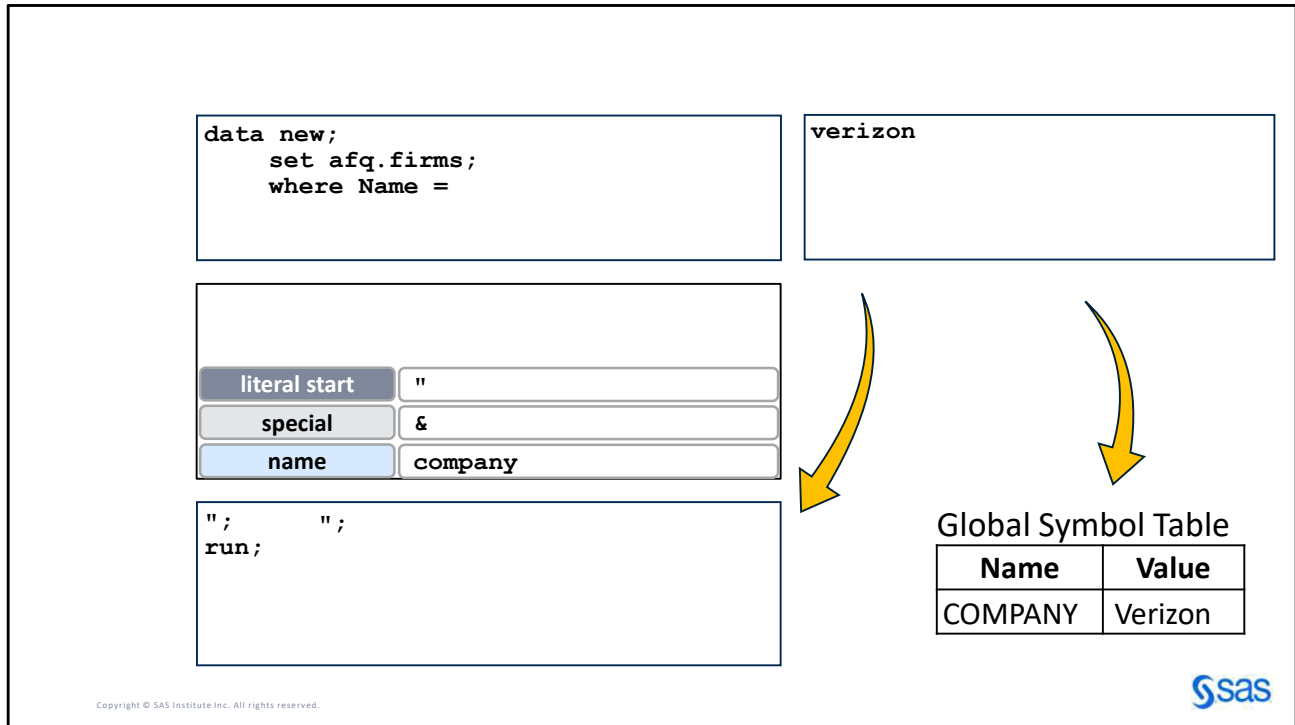
Global Symbol Table

Name	Value
COMPANY	Verizon

Copyright © SAS Institute Inc. All rights reserved.



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

```
data new;  
  set afq.firms;  
  where Name =
```

literal start	"
name	verizon
literal end	"

```
;  
run;
```

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 for the macro processor to tell a literal value from code.

The Macro Dilemma



Global Symbol Table

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



No problem can be solved from the same level of
consciousness that created it.
Albert Einstein

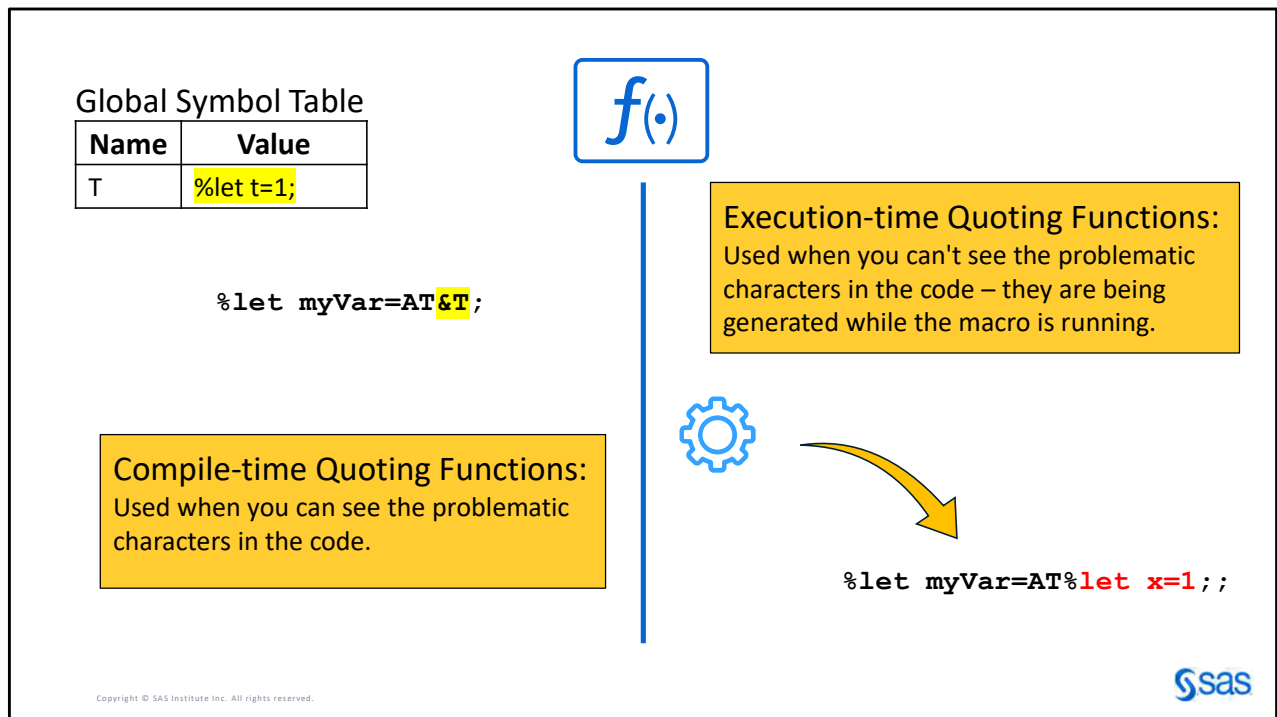


```
"%let x=1;"
```

Copyright © SAS Institute Inc. All rights reserved.



If we can't resolve this dilemma with quotation mark, surely there is somehow we can help the macro processor make the correct decision? 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.



Instead of quotation marks, macro uses quoting *functions* 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 depends 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.

Compile-Time Quoting Functions

For text you can see in your code

```
where company="&company";
```

%STR

- Masks all special characters and mnemonic operators in a text string except & and %.
- Unbalanced quotation marks [' "] and parentheses [()] must be preceded by a % character.

```
where company='AT&T' ;
```

%NRSTR

- Masks all special characters and mnemonic operators in a text string.
- Unbalanced quotation marks [' "] and parentheses [()] must be preceded by a % character.

Copyright © SAS Institute Inc. All rights reserved.



As we write our programs, we'll need to decide if we want macro triggers to resolve 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, the macro %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 %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.

```
%let t=time;  
%put Do you have the &t?;
```

Do you have the time?

```
%put This report is for AT&T;
```

This report is for ATtime

Copyright © SAS Institute Inc. All rights reserved.



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

```
%put This report is for AT&T;
```

```
%STR()
```

Copyright © SAS Institute Inc. All rights reserved.



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

```
%let list=%nrstr(Verizon|AT&T);  
%put &=list;  
%put This report is for %scan(&list,1,|);  
%put And the next report is for %scan(&list,2,|);
```

```
COMPANIES=Verizon|AT&T  
...  
This report is for Verizon  
...  
And the next report is for ATtime
```

Copyright © SAS Institute Inc. All rights reserved.



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...

Macro Execution-Time Quoting

Use when text resolves during execution, so you can't see it in the code.

%BQUOTE

Masks special characters and mnemonic operators in a resolved value, except & and %.

%SUPERQ

Masks all special characters and mnemonic operators in a resolved value. Prevents further value resolution.

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

Copyright © SAS Institute Inc. All rights reserved.



Well, in this case the offending character wasn't visible in my code. I'll need an execution-time quoting function for this. Once again, I'll need to decide if I want macro triggers to resolve or not. To allow macro triggers to resolve, I usually use %BQUOTE. If not, I usually make sure my text is never mistaken for code by using %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 without the macro processor resolution, so there's no chance of a "misunderstanding".

Summary: Macro Quoting Functions

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

Copyright © SAS Institute Inc. All rights reserved.



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 Atttime
```

Copyright © SAS Institute Inc. All rights reserved.



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

Copyright © SAS Institute Inc. All rights reserved.



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

```

/* Store quoted text */
%let company=%nrstr(AT&T);
%put &=company;

```

COMPANY=AT&T

```

/* Store unquoted text */
data _null_;
    call symput('COMPANY2', 'AT&T');
run;
proc sql;
select Name 'Name'
      ,Value 'Value'
      ,Value 'HEX Value'
format=$hex12.
  from dictionary.macros
 where name like 'COMPANY%'
;
quit;

```

Name	Value	HEX Value
COMPANY	ATT	0141540F5402
COMPANY2	AT&T	415426542020

Copyright © SAS Institute Inc. All rights reserved.



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. If you query the **dictionary.macros** table and display the values in HEX, you will be able to detect those delta characters. In the COMPANY value, you can see there is a '01'x before and a '02'x after the quoted text. The hex codes for the A and T characters are the same as the unquoted value, but the quoted ampersand hex value is different from the unquoted value. Because the quoted version doesn't actually contain an ampersand, it won't trigger the macro processor. Unquoting macro values removes the leading and trailing hex characters and converts the delta characters back to their original values.

```
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

Copyright © SAS Institute Inc. All rights reserved.



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...

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

Copyright © SAS Institute Inc. All rights reserved.



Fortunately, there are quoting versions of every text processing function available. Some are implemented as actual macro functions; other functions ship as part of 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 resulting text they return is already quoted, so it will never inadvertently trigger additional processing in the macro processor.

```
%put A report for %qscan(%superq(companies),2,|);
```

```
A report for AT&T
```

Copyright © SAS Institute Inc. All rights reserved.



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

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

Copyright © SAS Institute Inc. All rights reserved.



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"
```

Copyright © SAS Institute Inc. All rights reserved.



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.
```

Copyright © SAS Institute Inc. All rights reserved.



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
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 */;
```

Copyright © SAS Institute Inc. All rights reserved.



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 with the WHERE statement.

Unquoting

When quoted text needs to be executed it as code:

%UNQUOTE

Unmasks all special characters and mnemonic operators

Copyright © SAS Institute Inc. All rights reserved.



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

Copyright © SAS Institute Inc. All rights reserved.



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

Other Macro Quoting Functions

For backwards compatibility or rare niche problems

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

Copyright © SAS Institute Inc. All rights reserved.



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



Macro Quoting and Troubleshooting

This demonstration illustrates using macro quoting functions in macro code and troubleshooting macro quoting issues using SAS system options.

Copyright © SAS Institute Inc. All rights reserved.



See the demo code included in the ZIP package.

This presentation was developed and demoed using SAS on Demand for Academics – and if you haven't already, you should register for a free account there yourself! It's a great place to practice and learn to code in SAS. To grab the programs and data for this presentation, download the zip file from <https://bit.ly/AFineQuotation>

After unzipping the programs and data to a location accessible to your SAS session, run the **setup.sas** program found in the folder named "A Fine Quotation - Macro Quoting" to set up your system, and you're ready to run the demo code yourself.

Contact Information



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

Mark Jordan
SAS Institute, Inc.
mark.jordan@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.