Paper 2221 - 2015

Quotes within Quotes: When Single (‘) and Double (“) Quotes are not Enough

Arthur L. Carpenter, California Occidental Consultants, Anchorage, AK

ABSTRACT

Although it does not happen every day, it is not unusual to need to place a quoted string within another quoted string. Fortunately SAS® recognizes both single and double quote marks and either can be used within the other. This gives us the ability to have two deep quoting. There are situations, however where two kinds of quotes are not enough. Sometimes we need a third layer or more commonly we need to use a macro variable within the layers of quotes. Macro variables can be especially problematic as they will generally not resolve when they are inside single quotes. However this is SAS, and that implies that there are several things going on at once and that there are several ways to solve these types of quoting problems.

The primary goal of this paper is to assist the programmer with solutions to the quotes within quotes problem with special emphasis on the presence of macro variables. The various techniques are contrasted as are the likely situations that call for these types of solutions. A secondary goal is to help the reader understand how SAS works with quote marks and how it handles quoted strings. Although we will not go into the gory details, a surface understanding can be useful in a number of situations.

KEYWORDS

Quotes, Macro variable resolution, Apostrophe, Single quotes, Double quotes, parsing, %BQUOTE

INTRODUCTION

SAS uses quote marks as parsing characters. Since SAS is a parsed language, these characters help the SAS parser to determine how to read and interpret the code that we write. Quotes are central to this process as they are used to mark constant text that is to receive either special handling or that is to be interpreted by another entity, such as the operating system. When a quote mark is encountered by the parser, all the text between that mark and its subsequent sibling are marked for special handling internally.

When quoted strings are nested within other quoted strings, the parser is required to essentially make multiple passes of the code to correctly set aside the strings within strings. After the first pass the outer level of marks are used and noted. Then interior quote marks are interpreted. Clearly this implies that a great deal of care must be taken by the programmer to ensure that the correct interpretation is applied.

One of the simplest situations of quotes within quotes is that of the use of an apostrophe, say in a title. Although it is no longer required that we quote title text, we cannot simply write the title using one single quote as the apostrophe. If we do so, the quote mark will be interpreted as the start of a quoted string and SAS will search for the closing quote. This will, of course, cause all sorts of mayhem later. Nor can we use single quotes to surround the text containing the apostrophe. The parser will see the second quote (the apostrophe) as the end of the quoted string. The remainder of the title is unquoted (s TRUCK), and then, and perhaps worse, the third quote starts a new quoted string that will mask the semicolon and other statements until a matching quote is found (or until the string becomes too long). Instead we solve this problem by using double quotes to mask the interior single quote. In this way double quotes can be used to mask single quotes or single quotes can be used to mask double quotes.

```
title1 Tom's Truck;
```

```
title1 'Tom's Truck';
```

```
title1 "Tom's Truck";
```

In much earlier versions of SAS the double quote was not available and a different solution to this type of problem was employed. Here the outside double quotes are replaced by a single quote and the apostrophe is replaced by two single quotes. This works because when the parser sees two single (or double) quotes immediately following each other, the parser resolves them into one quote mark *after* the closing quote has been determined. In this example the title text is marked in the first pass and the two adjacent single quotes

```
title1 'Tom''s Truck';
```

are then translated into a single quote mark, but by now the parser has stopped looking for quote marks and the single quote is seen as an apostrophe. Later we will see how this behavior can still be useful to us in the current versions of SAS.

Problem Code

There are a number of coding situations that require quotes within quotes. For the most part these coding situations can be represented by the four types of statements shown here.

- X statement
  The path in the DIR command is passed to the operating system and under the WINDOWS operating system it must be quoted using double quotes. Primarily this is to allow for the use of blanks within either folder or file names.

```
x 'dir "c:\My Loc\*.sas" /o:n /b > "c:\My Loc\pgmlist.txt"';
```

- DM statement
  The DM statement passes one or more Display Manager specific commands from within your program to the Display Manager for execution. The commands must be quoted and under Windows any path information must be double quoted.

```
dm 'log; file "c:\My Loc\logdump1.log"';
```

- CALL DEFINE statement
  The CALL DEFINE routine in PROC REPORT's compute block often requires quoted text especially when modifying styles. Some style attributes, such as FLYOVER, also require quoted text.

```
call define(_col_,'style', 'style={flyover="My Loc Mean WT"}');
```

- FILENAME statement
  The PIPE device type can be used on the FILENAME statement to create a virtual file (instead of a physical file as done with the X statement above). Again under Windows paths must be quoted using double quotes.

```
filename tmpdat pipe 'dir "c:\My Loc\*.sas" /o:n /b';
```

Each of these statements work and there are no problems or issues associated with them. However if we introduce a macro variable within the inner quoted string, there very well could be a problem. Macro variables tend not to resolve when they are within a string defined by single quotes. In each of these examples there is a double quoted string within a single quoted string. Regardless of whether double quotes are within the single quotes or the single quotes are within the double, if a macro variable is within the inner string it will be within a string delineated by single quotes. This can prevent macro variable resolution, but as we will see – not always.

In the examples that follow the macro variable &TEMP has been assigned the value MY LOC and each of previous statements has been modified accordingly. In each case the macro variable lies within single quotes.

```
%let temp = My Loc;
```

THE X STATEMENT

Normally we expect any macro variables to be resolved before the statement that contains them is executed. However since the macro variable &TEMP is enclosed by single quotes it will not be resolved. We can see this by using the macro %PUT statement to show the statement after any resolution. As we anticipated, the LOG shows that the macro variable remains unresolved, which of course will cause the X statement to fail.

```
x 'dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\pgmlist.txt"';
```

```
50   %put x 'dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\pgmlist.txt"';
x 'dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\pgmlist.txt"'
```

It turns out that this is the easiest of the four situations to solve (and easiest to understand). In the current versions of SAS the X statement, like the TITLE statement, no longer requires the text to be quoted. This means that we can remove the single quotes that surround the DIR command. The %PUT shows that the macro variable does indeed resolve to the correct value. When working with statements such as the X statement, that traditionally required quotes, check to see if they are still required.

```
x dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\pgmlist.txt";
```

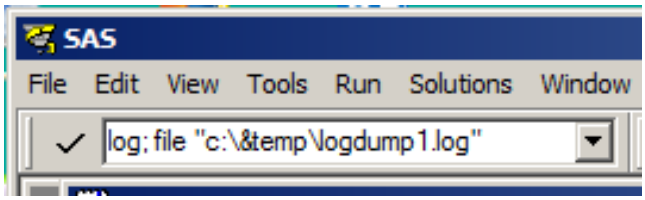
```
54    %put x dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\pgmlist.txt";
x dir "c:\My Loc\*.sas" /o:n /b > "c:\My Loc\pgmlist.txt"
```

THE DM STATEMENT

A simple test shows that unlike the X statement, the DM statement still requires the commands to be quoted. This is actually not too surprising considering that a series of DM commands are concatenated using a semicolon. Without the quotes, the *extra* semicolons would themselves confuse the parser. After adding the macro variable, our DM statement surrounds the macro variable with single quotes. But surprisingly this works the way we intended for it to! What happened? How could the macro variable &TEMP be resolved? It turns out that, because this is a DM command, and DM commands can contain macro variables, we get two chances at resolution. The DM statement passes the command to the Display manager for execution. The outer quotes have been used by the parser to identify the DM commands and are therefore no longer needed. At this point the macro variable has not yet resolved, but since &TEMP is no longer quoted and the commands themselves now need to be interpreted by the Display Manager, the macro variable is free to resolve. As is usual the macro variable resolves before the FILE command executes.

```
55    dm log;
ERROR: Error in the DM statement.  DM must be followed by a quoted string.
```

```
dm 'log; file "c:\&temp\logdump1.log"';
```



There is a nuance here that might interest some readers. This is not yet the whole story on this particular DM statement. If we were to reverse the quotes in the DM statement the macro variable still resolves – even though it is contained within single quotes. This is because the FILE command itself is expecting a quoted string [well technically, the quotes are to allow for blanks in one or more of the names]. This means the quotes are used as parsing characters within the command and the macro variable actually gets a third chance to resolve before the FILE command executes.

```
dm "log; file 'c:\&temp\logdump1.log'";
```

In early versions of SAS9 the macro variable &TEMP would not have resolved in the previous DM statement. In those versions we had to utilize variations of the techniques shown for the FILENAME example below.

THE CALL DEFINE STATEMENT

The DEFINE routine is used inside of the REPORT step's compute block to conditionally change attributes associated with the row, column, or report item. These can include changes to style attributes and many of these attribute changes require quoting of the attribute value. In the REPORT step used in this example the DEFINE routine is used to add flyover tip text to the mean values of WEIGHT. The entire third argument of the CALL DEFINE must be quoted as must be the text associated with the FLYOVER attribute. As in the previous two examples the macro variable is within single quotes.

```
title1 'Flyover on Weight';
proc report data=sashelp.class nowd;
column sex weight;
define sex / group;
define weight / analysis mean f=6.2;
compute weight;
   call define(_col_,'style', 'style={flyover="&temp Mean WT"}');
endcomp;
run;
```

Within PROC REPORT the compute block goes through a multistep

parsing process. During the SETUP phase, and well before compute block execution which takes place in the REPORT ROW phase, the code that contains the STYLE= option is scanned a separate time as is the text associated with the FLYOVER option. Like with the DM statement shown above, as a result these multiple scanning passes, the macro variable will resolve correctly regardless of whether the single quotes are on the outside or inside of the quotes within quotes. No special handling is required.

THE FILENAME STATEMENT

The FILENAME statement expects the *external-file* to be quoted. In this usage of the statement we are using the PIPE device type, so instead of an external file the quoted string is a *process,* which in this case is the DOS DIR command. We cannot simply remove these outer quotes as we could to solve the similar problem in the X statement shown above. As an additional constraint under Windows, the OS expects the path information to be enclosed in double quotes. Thus the outer quotes must be single quotes.

```
filename tmpdat pipe 'dir "c:\&temp\*.sas" /o:n /b';
```

To solve this problem we must hide the outer single quotes from the parser until after the macro variable has been resolved. This is not particularly hard to do but we need to understand what we are doing and why. There is more than one way to delay the application of the single quotes. Think of it as a timing issue. Normally the single quote will be applied as it is first seen (left to right). Since the macro variable is to the right of the first single quote this is a problem. If however we can force the single quote to be seen only after the macro variable is resolved, then the problem goes away.

Fortunately there is more than one way to hide the single quote.

Using Macro Quoting Functions

Macro quoting functions can be used to temporarily mask parsing characters such as quotes. Macro quoting functions act by inserting invisible masking characters that pass information to the macro parser about how to parse the masked characters. There are a number of macro quoting functions to choose from, and the one that I tend to use, because it quotes a wider variety of characters in more situations, is %BQUOTE. The %STR function can also be used, however since it does not automatically mask quotes, a special secondary masking character (the % sign) must proceed each quote mark that is to be masked. Here a single quote is masked using the %STR function.

```
%str(%')
```

In the FILENAME problem below, the %BQUOTE is going to be used to mask the single quotes. The first attempt at masking the two single quotes using %BQUOTE fails. The code shown here successfully masks the single quotes, and allows the macro variable to resolve, however the FILENAME statement itself fails to parse correctly. The problem is that now the parser is misinterpreting the DIR command as an unquoted option. The interpretation of the DIR command also needs to be delayed until the single quote has been unmasked (restored), and this needs to happen after the macro variable has been resolved.

```
filename tmpdat pipe %bquote(')dir "c:\&temp\*.sas" %bquote(/o:n /b');
```

We can further delay the parsing of the entire string by first passing the entire string to the macro facility. There are several ways to do this, but the easiest is by using the %UNQUOTE function. Primarily the %UNQUOTE function is used to remove macro masking characters that have been inserted by functions such as %BQUOTE. Since %UNQUOTE is executed by the macro facility, all the code within its parentheses are first passed to the macro facility for processing. Consequently here the entire DIR command is passed to the macro facility with the single quotes masked. The macro variable &TEMP is resolved and then the single quotes are unmasked. At this point the resultant string (shown here), including the single quotes, is passed back out of the macro facility where the DIR command can then be parsed. Other macro functions, such as %LEFT and %TRIM, will also remove macro quoting, however the %UNQUOTE function is more specific for this task, and does not have some of the limitations of these other macro functions.

```
filename tmpdat pipe %Unquote(%bquote(')dir "c:\&temp\*.sas" %bquote(/o:n /b'));
```

```
'dir "c:\My Loc\*.sas" /o:n /b'
```

Using the DATA Step Quote Function

Double quote marks can be added using the DATA step's QUOTE function. Since the FILENAME statement is a global statement, it is outside the DATA step, however we can still use the QUOTE function if it is called by the %SYSFUNC macro function. Here we are using the QUOTE function to supply the outer quote marks for the parser. Although we ultimately will have a string surrounded by double quotes within a string of double quotes, this will not be an issue because of the timing associated with the parsing of the interior string. Of all the FILENAME statement solutions shown here, this is probably the simplest.

```
filename tmpdat pipe %sysfunc(quote(dir "c:\&temp\*.sas" /o:n /b));
```

Using Repeated Quote Marks

Another way to delay interpretation of the quote marks is by doubling them up. This technique was first briefly described in the introduction. Here double quotes are used to surround the entire string, and interior string is marked with doubled double quotes. Remember that the quote marks are used by the programmer to tell the parser how to handle a set of characters. In this case the interior string is marked in a second pass of the parser, after the macro variable has been resolved.

```
filename tmpdat pipe "dir ""c:\&temp\*.sas"" /o:n /b";
```

As we have seen in other situations, we could not have used single quotes as the outer set of quotes. Doing so would mask the macro variable regardless of the type of interior quotes. Consequently this FILENAME statement will return an error, as the path will not resolve correctly.

```
filename tmpdat pipe 'dir ""c:\&temp\*.sas"" /o:n /b';
```

A SMALL QUIZ

You seem to have read this far. Since the topic seems to have amused you to at least some extent, here is a simple quiz to test your understanding. In the following code NOTE0 produces what we want. Which of the other variables NOTE1 – NOTE3 will contain the same value?

```
%let saying = Nevermore;
data _null_;
note0 = "The raven sayth: 'Nevermore'";
note1 = "The raven sayth: '&saying'";
note2 = 'The raven sayth: ''&saying''';
note3 = "The raven sayth: ""&saying""";
put (note:) (=);
run;
```

ANSWER:
Only NOTE1; during the assignment of the variable's value, the inner single quotes are masked, not seen as parsing characters, so the macro variable resolves
NOTE2; the outer single quotes prevent macro variable resolution by masking the ampersand
NOTE3; The doubled double quotes resolve to double quotes, not single quotes (OK this is picky, but still . . . .).

SUMMARY

Quotes within quotes – not an unusual problem, but fortunately not an insurmountable one. Even when macro variables are involved, their resolution can be obtained through the use of one of several techniques. When applied by the SAS programmer who understands how quotes are used in the parsing process, these techniques become more intuitive and can be applied more successfully.

Remember that quotes are parsing characters. They help the parser distinguish between strings, constant text, and other code structures such as keywords and options. Remember also that there are a number of timing events. Things happen in an order. Understanding that order and understanding how quotes are used in the parsing process will go a long way to help you solve any problems involving quotes within quotes.

ABOUT THE AUTHOR

Art Carpenter's publications list includes; five books, two chapters in *Reporting from the Field*, and numerous papers and posters presented at SAS Global Forum, SUGI, PharmaSUG, WUSS, and other regional conferences. Art has been using SAS since 1977 and has served in various leadership positions in local, regional, and national user groups.

Art is a SAS Certified Advanced Professional Programmer, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

Certified Clinical Trials Programmer Using SAS®9

Certified Advanced Programmer *for* SAS®9

View my paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

REFERENCES

Some of the examples in this paper have been 'borrowed' (with the author's permission) from:
Carpenter, Art, 2012, *Carpenters Guide to Innovative SAS® Techniques*, SAS Press, SAS Institute Inc., Cary NC.

TRADEMARK INFORMATION