

Paper 005-2013

## Macro Quoting to the Rescue: Passing Special Characters

Mary F. O. Rosenbloom, Edwards Lifesciences LLC, Irvine, CA  
Art Carpenter, California Occidental Consultants, Anchorage, AK

### ABSTRACT

We know that we should always try to avoid storing special characters in macro variables. We know that there are just too many ways that special characters can cause problems when the macro variable is resolved. Sometimes, however, we just do not have a choice. Sometimes the characters must be stored in the macro variable whether we like it or not. And when they appear we need to know how to deal with them. We need to know which macro quoting functions will solve the problem, and even more importantly why we need to use them. This paper takes a quick look at the problems associated with the resolution and use of macro variables that contain special characters such as commas, quotes, ampersands, and parentheses.

### KEYWORDS

%BQUOTE, %LEFT, %NRSTR, %QLEFT, %NRBQUOTE, %SUPERQ, macro variable resolution, special characters

### INTRODUCTION

Generally the inclusion of special characters, such as quotes, commas, ampersands, and parentheses in macro variable values is to be discouraged. These characters can cause problems in a variety of ways when the macro variable is resolved. Unfortunately we do not always have full control over the values taken on by the variables, and when they do contain special characters we need to know how to handle them.

Fortunately macro quoting functions are available. However, the whole topic of quoting in the macro language is difficult to work with, and even more difficult to understand. In this paper we will be discussing a number of situations where macro quoting provides a solution to an otherwise difficult if not intractable programming problem.

The examples will utilize variations of the following problem. The data set HOSPITALVISITS contains the variable SITECODE which takes on the values shown in the list shown to the left. We wish to process a series of PROC PRINT steps for each value of SITECODE. Since in the actual program there are a series of PROC steps for each value of SITECODE, we intend to automate the process by using the macro language. A macro variable is being used to store the value of the data set variable (SITECODE), and the macro variable will be used in a

```
sitecode
SiteA
SiteB,C
SiteD (with E)
SiteF "aka G"
Sites H&J
```

```
%let string1 = SiteA;

title1 "Visits at &string1";
proc print data=HospitalVisits;
  where sitecode="&string1";
run;
```

WHERE clause to subset the data. This is a very straightforward use of a macro variable to supply character substitution.

Even this simple use of text substitution can fail when special characters become involved. For the problem described in this paper the values used in the WHERE are obtained from the data itself, and then processed using a macro list (Fehd and Carpenter, 2007). The code used in the subsequent examples reflects the usage of this list. A %DO loop steps through the individual elements, one at a time, with each element in the list designated as &&STRING&I.

```
%macro show1;
%do i = 1 %to 5;
  title1 "Visits at &&string&i";
  proc print data=hospitalvisits;
    where sitecode="&&string&i";
  run;
%end;
%mend show1;
```

Special characters are not an issue when stored as data set values, and are generally not an issue when being transferred into macro variables. Problems do arise however when the macro variables, which contain special characters, are resolved. The following variants on this example demonstrate

common problems, as well as provide an opportunity to discuss solutions.

```
%let string1 = SiteA;
%let string2 = SiteB,C;
%let string3 = SiteD (with E);
%let string4 = SiteF "aka G";
%let string5 = Site H&J;
```

For the purposes of this discussion, consider these macro variables and the special characters that they contain. &STRING2 contains a comma, &STRING3 matched parentheses, and &STRING4 matched double quotes. &STRING5 contains an ampersand; &J appears to the parser as a macro variable reference, however the macro variable &J has not been defined. Other special characters,

and combinations of normally matched characters, but without a match, e.g. parentheses and quotes, would provide further variations to those discussed here.

## COMPILATION-TIME VS. EXECUTION-TIME MACRO QUOTING

When SAS<sup>®</sup> code is processed, the macro processor goes through the code looking for special characters, such as arithmetic operators, ampersands and percent signs (macro triggers), quotes, and other characters like mnemonics, and attempts to resolve these. This process takes place both during the compilation of the statement or macro and again during the execution phase. The macro quoting functions, which are discussed below, come in two basic flavors; those that are applied during the compilation process, and those that are applied during the execution of the macro element. The differences between these two general types of quoting functions are subtle and do not usually impact how we select which quoting function to utilize. All of the quoting functions are execution time functions with the exception of %STR and %NRSTR. Compilation time functions operate against the values included in the code, while execution time functions operate against resolved values.

## THE QUOTES PROBLEM

One of the most common situations that requires macro quoting involves macro variables with quotes embedded in their values. Of the five macro variable values shown above, only the fourth will fail when used the the WHERE clause. The inserted value becomes: "SiteF "aka G"", and the LOG shows "SiteF "aka G". Notice that the double quotes embedded within the macro variable have become confused with those already present. As an aside, when two quotes or double quotes appear together within code, as they do here, they are 'seen' as a single character after a second pass of the parser. The result is mismatched quotes.

The macro quoting function %BQUOTE alone solves most of these types of special character problems, but not all of them. There are still traps aplenty for the unwary. %BQUOTE masks special characters and mnemonics, and will

```
title " Visits at %bquote(&&string&i)";
proc print data= hospitalvisits;
  where sitecode="%bquote(&&string&i)";
```

also mask quotes and parenthesis, even if they are not resolved in pairs. But, this function will still not mask the macro triggers, % and &, that may be embedded in the text.

A related problem with quotes is often experienced when a quoted string appears inside of another quoted string. This can occur in a number of SAS programming situations, and is especially problematic when passing strings to a Windows operating system.

This X statement directs Windows to create a directory. Since Windows requires paths to be surrounded by double quotes, we can use single quotes to surround the text associated with the X statement. Since we almost never need to nest quoted strings more than two deep, using a combination of single and double quotes solves the problem – unless a macro variable is needed. One way or the other, the macro variable must be inside of a single quote, and the macro variable will necessarily remain unresolved.

```
x 'md "c:\temp\output"');
```

Fortunately there are ways to either avoid the use of the single quotes or to prevent their use from masking the macro variable. The %STR quoting function can be used to temporarily mask the inner quotes by preceding the quotes with a % masking character. In this way single quotes are not needed. This general concept has broader application than just for the X statement. A number of other statements and options can at times require the use of quotes within quotes.

```
x "%str(md "%&projloc\output%")";
```

These can include the TITLE, FILENAME, and LIBNAME statements, as well as, the STYLE override option.

In the current versions of SAS some of the statements that have required quoting, such as the X, LABEL, and TITLE statements, no longer always require quotes. For the X statement shown above, the outer quotes are not required, so a simpler solution for this specific problem would be to simply rewrite it without the single quotes.

```
x md "&projloc\output";
```

Quotes within quotes can also be a problem in the FILENAME statement. Again this is only an issue when macro variables are also utilized. In the following example we want to write all the names of all of the RTF files found in the specified directory to the LOG. In this case we have used the %BQUOTE function to mask the single quotes until after the macro variable has been resolved. After the macro variable has been resolved the masking

```
filename list pipe
  %unquote(%bquote('dir "&projloc\*.rtf" /o:n /b %bquote('));

data _null_;
infile list;
input ;
put _infile_;
run;
```

characters for the single quotes must be removed, and this is accomplished with the %UNQUOTE function.

## AMPERSANDS AND PERCENT SIGNS

Because the ampersand (&) and the percent sign (%) have special meaning to the macro language these characters can be especially problematic when embedded within macro variables. Consider the site name stored in &STRING5. In this case there is no macro variable &J, instead we would like H&J to be treated as text as if there was no macro trigger involved. We need to mask the &. Unfortunately the & is seen as a macro trigger. Since the macro processor cannot find &J in the global symbol table, a warning is sent to the LOG when the macro variable &STRING5 is defined.

```
%let string5 = Sites H&J;
```

```
WARNING: Apparent symbolic reference J not resolved.
```

Once the macro parser determines that it cannot resolve &J, it is marked and the parser continues with the next statement. Ultimately, therefore, &string5 takes the value of 'Sites H&J'. Whenever we use &STRING5, as it will be in %show1, the ampersand will again be recognized as a macro trigger, and the token &J will again be passed to the

```
%macro show1;
%do i = 1 %to 5;
  title "&&string&i"; /* show how the single quote is
                    generated for the fourth exp*/
  proc print data=hospitalvisits;
    where sitecode="&&string&i"; /* works for first three */
  run;
%end;
%mend show1;
%show1
```

macro processor for resolution. At each instance, every time &STRING5 is used, another warning will be generated in the LOG. Of course if we are very unlucky, the macro variable &J will already exist, no warnings will be generated, and we will get incorrect results.

Since single quotes will mask both ampersands and percent signs we do not get this warning when creating macro variables in the DATA step. However when it is subsequently used, an attempt is made to resolve the &J, and a warning is issued.

```
data a;
call symputx('string5', 'Sites H&J');
run;

* Warning when used;
%put |&string5|;
```

```
6 * Warning when used;
7 %put |&string5|;
WARNING: Apparent symbolic reference J not resolved.
|Sites H&J|
```

Clearly in this case we do not want the &J to be seen as a macro reference and we would like to avoid the resulting warning from appearing in the LOG.

If we attempt to create the macro variable using %LET, a similar result and warning is observed. In this case there is nothing to mask the ampersand and an attempt is made to resolve the &J both when the macro variable is created and when it is used.

```
%let string5 = Sites H&J;
%put |&string5|;
```

We need to examine quoting functions that can be used to prevent the resolution of &J. Ideally we would like to mask &J in such a way that it is not even seen as a macro trigger. The three macro quoting functions shown here include %NRBQUOTE, %NRSTR, and %SUPERQ. The capabilities and performance of these functions are not the same, and it is worth looking at each individually.

The %NRBQUOTE function is the NoRescan version of the %BQUOTE function. Both behave the same except in how they treat resolved ampersand and percent signs.

This function will attempt to resolve the &J **once** and if unsuccessful the ampersand is then masked so that it will no longer be seen as a macro

trigger. Here an attempt is made to resolve the & during the assignment of the %LET statement. Then in a subsequent use of the macro variable the &J is masked and the warning is not reissued.

```
75 %let string5c = %nrblockquote(&string5);
WARNING: Apparent symbolic reference J not resolved.
76 %put |&string5c|;
|Sites H&J|
```

We can prevent even the first attempt at resolving the &J by using the %NRSTR quoting function. This compile time function immediately masks the &J. Since the ampersand is immediately masked, this means that &NRSTR cannot be

```
80 %let string5d = %nrstr(&string5);
81 %put |&string5d|;
|&string5|
82 %put |%nrstr(&string5)|;
|&string5|
```

```
%let string5b = %nrstr(Sites H&J);
%put |&string5b|;
```

used on a macro variable that needs to be resolved. Here we attempt to use %NRSTR with &STRING5. As we can see &STRING5D contains the unresolved macro variable &STRING5.

In addition the %NRSTR function, and the %STR function, are more limited than the %BQUOTE and

%NRBQUOTE functions in the special characters that they can mask (they cannot mask things like resolved quotes and resolved parentheses).

Some of the limitations of the %NRSTR function can be overcome, while still immediately masking the &J, by using the %SUPERQ quoting function. The %SUPERQ function masks resolved values, such as ampersands. Unlike the other quoting functions, the argument of %SUPERQ is assumed to be the name of a macro variable. This means that if the argument contains an ampersand it will be resolved first and the result is assumed to be the name of a macro variable. %SUPERQ would not be used to quote text unless that text is already contained in a macro variable.

```
85 %let string5e = %superq(string5);
86 %put |&string5e|;
|Sites H&J|
```

It is not inappropriate to use ampersands within the argument of %SUPERQ, but remember that the resolved value must be the name of a macro variable. Delayed resolution references are possible, but they are specified differently when they appear as an argument of %SUPERQ. When using %NRBQUOTE one might specify an argument as %NRBQUOTE (&&STRING&I), the same delayed resolution reference using %SUPERQ would be specified as %SUPERQ (STRING&I).

Usually in a situation, such as this one with the &J, you will want to mask the & as soon as possible and generally you will want to use the %NRSTR function to prevent any warnings to be written to the LOG. When the macro reference may contain resolved characters that are not masked by %NRSTR, the %SUPERQ quoting function is available.

In general, you cannot see the special characters that need quoting by simply reading the SAS code. More often it is the resolved value of the macro variable that will contain characters that need to be masked. When this is the case, execution-time quoting is needed and %NRSTR will not be sufficient. For our HOSPITALVISITS dataset, described above, the macro variables are loaded with site data which contain special characters. Our coding becomes more complicated when the site code is stored in a macro variable, as it was in the introduction. Now we must turn to the execution-time quoting functions. In the previous example, we needed to mask the &J before the macro was compiled, and this was done with the %NRSTR function. However, since the %NRSTR quoting function prevents the resolution of the macro variable &STRING5D by masking the &, we cannot use it when the value itself is stored in a macro variable. In this example %NRBQUOTE is used successfully to resolve &STRING5C, while %NRSTR does not resolve &STRING5D. Notice that the use of %NRBQUOTE allows the resolution of &STRING5C, attempts to resolve &J once (resulting in the warning), and then stores the value with the &J masked. This means that the warning will not be displayed on subsequent uses of the macro variable &STRING5C. In some companies, it is not permissible to have a WARNING statement in the SAS log, when this is the case the %SUPERQ quoting function often provides an alternative to the %NRBQUOTE function.

## LISTS and NESTED FUNCTIONS

The overall problem can easily become more complicated in a number of ways, and two of the more common complications are discussed here. The first involves the use of macro list processing and the second the use of nested functions.

There are several techniques for processing a list of values in the macro language (Fehd and Carpenter, 2007). The following example utilizes a list of macro variables and steps through the list one at a time. The values of interest have again been stored in the macro variables &STRING1 through &STRING5. These will be processed in a %DO loop and will be addressed as &&STRING&I. So that we can demonstrate the use of nested functions, blanks have been added to the text values.

```

%* add a leading space to foil the WHERE;
%let blk = %str( );
%let string1 = &blk.SiteA;
%let string2 = &blk.SiteB,C;
%let string3 = &blk.SiteD (with E);
%let string4 = &blk.SiteF "aka G";
%let string5 = &blk.Sites H&J;

```

The values are to be used in a WHERE statement and the leading blank will cause problems with the matching of the macro variable value to the stored value in the data, which does not have a leading blank. Although leading and trailing blanks are normally removed when a macro variable is defined using the %LET, they will be preserved if the macro variable has been derived from a data step variable using CALL SYMPUTX or if inserted with a quoting function as was done above.

The intuitive solution is to use the %LEFT autocall macro function to simply perform a left justification of the value, and then quote the result with the %NRBQUOTE function. However simply adding the %LEFT function can have

```
where sitecode="%nrbquote(%left(&&string&i))";
```

unexpected consequences. Remember that the &&STRING&I macro variable is resolved first, the %LEFT is applied to the result, and that resolved value must comply with the

expectations of the %LEFT function. One of those expectations is that there is exactly one argument. While this solution will generally work, it fails for one of our site codes. Recall that the resolved value of &STRING2 contains a comma. In this case, the comma is interpreted as a function argument delimiter for the %LEFT function, which

```

ERROR: More positional parameters found than defined.
NOTE: The SAS System stopped processing this step because of errors

```

causes the compilation of the macro expression to fail and an

error to be generated.

Other possible combinations of functions could be considered. Each of these could be used in the WHERE

%nrstr(%left(&&string&i))
%nrbquote(%left(&&string&i))
%left(%nrbquote(&&string&i))
%qlleft(%nrbquote(&&string&i))
%qlleft(&&string&i)
%superq(string&i)
%qlleft(%superq(string&i))

statement with varying degrees of success. Two of these seven combinations of functions have already been discussed. We can try each of these in a small test loop and use the results to further understand how these functions work together.

The following tables show how each of these combinations of functions work with each of the five site codes in the HOSPITALVISITS data. Of these seven combinations only two work for all five of the test strings.

SiteA	Resolves To:	Errors?	Warnings?	OK?
<code>%nrstr(%left(&amp;&amp;string&amp;i))</code>	<code>%left(&amp;&amp;string&amp;i)</code>	No	No	❶
<code>%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteA	No	No	✓
<code>%left(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteA	No	No	✓
<code>%qleft(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteA	No	No	✓
<code>%qleft(%nrstr(%left(&amp;&amp;string&amp;i))</code>	SiteA	No	No	✓
<code>%superq(string&amp;i)</code>	SiteA	No	No	
<code>%qleft(%superq(string&amp;i))</code>	SiteA	No	No	✓

SiteB,C	Resolves To:	Errors?	Warnings?	OK?
<code>%nrstr(%left(&amp;&amp;string&amp;i))</code>	<code>%left(&amp;&amp;string&amp;i)</code>	Yes	No	
<code>%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>		Yes	No	❷
<code>%left(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteB,C	No	No	✓
<code>%qleft(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteB,C	No	No	✓
<code>%qleft(%nrstr(%left(&amp;&amp;string&amp;i))</code>		Yes	No	❷
<code>%superq(string&amp;i)</code>	SiteB,C	No	No	❸
<code>%qleft(%superq(string&amp;i))</code>	SiteB,C	No	No	✓

SiteD (with E)	Resolves To:	Errors?	Warnings?	OK?
<code>%nrstr(%left(&amp;&amp;string&amp;i))</code>	<code>%left(&amp;&amp;string&amp;i)</code>	No	No	
<code>%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteD (with E)	No	No	✓
<code>%left(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteD (with E)	No	No	✓
<code>%qleft(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteD (with E)	No	No	✓
<code>%qleft(%nrstr(%left(&amp;&amp;string&amp;i))</code>	SiteD (with E)	No	No	✓
<code>%superq(string&amp;i)</code>	SiteD (with E)	No	No	
<code>%qleft(%superq(string&amp;i))</code>	SiteD (with E)	No	No	✓

SiteF "aka G"	Resolves To:	Errors?	Warnings?	OK?
<code>%nrstr(%left(&amp;&amp;string&amp;i))</code>	<code>%left(&amp;&amp;string&amp;i)</code>	No	No	
<code>%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteF "aka G"	No	No	✓
<code>%left(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteF "aka G"	Yes	No	❹
<code>%qleft(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	SiteF "aka G"	No	No	✓
<code>%qleft(%nrstr(%left(&amp;&amp;string&amp;i))</code>	SiteF "aka G"	No	No	✓
<code>%superq(string&amp;i)</code>	SiteF "aka G"	No	No	
<code>%qleft(%superq(string&amp;i))</code>	SiteF "aka G"	No	No	✓

Sites H&J	Resolves To:	Errors?	Warnings?	OK?
<code>%nrstr(%left(&amp;&amp;string&amp;i))</code>	<code>%left(&amp;&amp;string&amp;i)</code>	No	Yes	
<code>%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	Sites H&J	No	Yes	✓
<code>%left(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	Sites H&J	No	Yes	✓
<code>%qleft(%nrstr(%left(%nrstr(%left(&amp;&amp;string&amp;i)))</code>	Sites H&J	No	Yes	✓
<code>%qleft(%nrstr(%left(&amp;&amp;string&amp;i))</code>	Sites H&J	No	Yes	✓
<code>%superq(string&amp;i)</code>	Sites H&J	No	No	
<code>%qleft(%superq(string&amp;i))</code>	Sites H&J	No	No	✓

❶ Clearly using the %NRSTR function will not be successful as it masks both the % and & before any attempt is made to resolve the value.

❷ Both the %LEFT and %QLEFT functions expect a single argument, and the unmasked comma causes these functions to detect what appears to be two arguments and this causes an error.



- ③ Using the %NRBQUOTE function on the argument to the %LEFT or %QLEFT function will mask the comma. This solves the problem of the comma, but now the string with embedded quotes again causes a problem!

```
where sitecode="%left(%nrbquote(&&string&i))";
```

After its execution the %LEFT un masks any special characters that had been masked by the %NRBQUOTE function. Consequently

the double quotes are now exposed to the parser where they cause problems for the WHERE statement.

This highlights a behavior that is shared by several macro functions. The %LEFT function always results in an unquoted string, even if the string was previously quoted. This is also true for each of the autocall macro functions that return text, including %LEFT, %TRIM, %SCAN, %SUBSTR, %UPCASE, and %VERIFY. Fortunately, we have another option. Each of these functions has a 'Q' analog that returns quoted values (regardless of whether or not they were quoted prior to calling the function). The analog function for %LEFT is %QLEFT.

%QLEFT is also an autocall macro function, and it has the additional feature of producing a result with all of the special characters and mnemonic operators masked, including the macro triggers % and &. %QLEFT can work

```
where sitecode="%qleft(%nrbquote(&&string&i))";
```

alone without %NRBQUOTE for all of the site codes that we send to it, with the exception of &string2, again because of the

comma. When it is paired, as it is here, with %NRBQUOTE, all of the macro variables resolve as desired (although not without warnings).

- ④ %SUPERQ alone does not left justify the resultant text. %SUPERQ is evaluated and maintains the spaces that we added when we defined the macro variables. Because of this, none of the observations are selected in the PROC PRINT.

```
where sitecode="%superq(string&i)";
```

We remedy this by adding %QLEFT to the mix, which left-justifies the quoted results and then re-quotes the result. This combination produces the correct data values, and thus the correct observations are selected by the WHERE statement.

```
where sitecode="%qleft(%superq(string&i))";
```

Of all of these combinations of functions only two return the result that we need and only the last one, using %QLEFT and %SUPERQ does so without any warnings.

## SOMETIMES SPECIAL CHARACTERS JUST WON'T WORK

Our original example showed a macro which called PROC PRINT using a WHERE clause to subset the data. The WHERE clause and the TITLE statement contained references to &&string&i. But what if we also wanted to save the PROC PRINT output to an .rtf file using ODS, where the filename contained the site name, too?

The special characters contained in the macro variables could cause syntax errors once resolved into the code. Here the file name in the ODS statement uses macro variables to form the name. And the name itself must be quoted.

```
ods rtf path="C:\output" file="%bquote( Report for &string4 &sysdate9..rtf)";
```

When resolved &STRING4 contains double quotes, which will create an ambiguity for the parser.

```
ods rtf path="C:\output" file=" Report for SiteF "aka G" 19JUL2011.rtf";
```

This is not a macro processing problem, but rather a problem with resolving double quotes within double quotes. There is a solution to this, however. In this case, we can use the TRANSLATE function with %QSYFUNC to exchange the double quotes for single quotes.

```
%let string4a=%qsysfunc(translate(&string4,%str('),%str(')));
```

The resulting ODS statement will parse correctly.

```
ods rtf path="C:\output" file=" Report for SiteF 'aka G' 19JUL2011.rtf";
```

A number of other solutions

are available to solve the quotes within quotes problem. In addition to those discussed earlier in this paper in the section "The Quotes Problem", alternate approaches can be found in Carpenter (2012).

## IN THE MACRO CALL

Another place that special characters can cause problems is in the macro call. In the preceding examples we wanted to print a portion of the data set. The PROC PRINT would commonly be in a macro. In this case we want to pass the value to be used in the WHERE statement into the macro. Having carefully learned the lessons in the previous example we use the %QLEFT and %NRBQUOTE (or %SUPERQ instead of %NRBQUOTE) quoting functions within the macro.

```
%macro printit(whr=);
  %put *****;
  title "%qleft(%nrbquote(&whr))";
  %put %qleft(%nrbquote(&whr));
  proc print data=hospitalvisits;
    where sitecode="%qleft(%nrbquote(&whr))";
  run;
%mend printit;
```

Using quoting functions within the macro however will not be enough. We still need to call the macro and pass the site code into the WHERE statement. The second of the five site codes, shown here, is unsuccessful, and results in an error. The problem is not in the macro, but in the macro call itself. The comma is seen as a parameter separator, and a

```
%printit(whr= SiteB,C)
```

positional parameter (which does not exist) is assumed. This results in the error. We need to mask the special characters in the macro call as well as within the macro.

```
415 %printit(whr= SiteB,C)
ERROR: All positional parameters must precede keyword parameters.
```

but in the macro call itself. The comma is seen as a parameter separator, and a

The solution is a straight forward extension of what we have already done. To make the following examples a bit more interesting we have used CALL EXECUTE (Fehd and Carpenter, 2007, and Michel, 2005), which is another way of processing a list of values, to generate the macro calls. The CALL EXECUTE routine can be used to write to a stack or queue, in this case we will write macro calls to the stack. After the DATA step has finished executing, the items in the stack are executed. This methodology completely avoids the need to build the list of macro variables and the resulting code does not use either %DO loops or the &&STRING&&I syntax.

The DATA step which uses the CALL EXECUTE is fairly simple, and our first attempt recreates the macro call shown above. As we might anticipate, this approach is unsuccessful for two of the five site codes (SiteB,C and SiteF "aka G"). In the first case the macro call fails and in the second the quote marks cause confusion.

```
data _null_;
  set hospitalvisits;
  call execute('%printit(whr='||sitecode||')');
run;
```

Our solution is to build the macro call with the special characters masked. We cannot apply the macro quoting functions to the DATA step variable SITECODE, because the timing would be wrong (SITECODE must be resolved in the DATA step first). We can, however apply the quoting functions to the call itself. The CALL EXECUTE statement now includes the %QLEFT and the %NRBQUOTE

```
call execute('%printit(whr=%qleft(%nrbquote('||sitecode||'))');
```

functions. For the first SITECODE the CALL EXECUTE produces a macro call that includes the %QLEFT and %NRBQUOTE or %SUPERQ functions. This macro call is successful for each of the five cases except the fourth – the one with the quotes.

```
%printit(whr=%qleft(%nrbquote(SiteA))
```

The timing of the compilation and execution of functions within a macro called by CALL EXECUTE are not necessarily the same as those of macros called through other methods. We can delay the process; make it more like the timing in non-CALL EXECUTE situations, by hiding the macro call until after the stack is read. We can do this by adding a %NRSTR function. In earlier examples the %NRSTR masked all the % signs and their associated function

```
call execute('%nrstr(%printit(whr=%qleft(%nrbquote('||sitecode||'))))')
```

calls. In this case, when using CALL EXECUTE,

this is a bit different. Here the masking due to the %NRSTR is unquoted and the % revealed when the macro call is read out of the stack.

## SUMMARY

Using macros to write data-driven code is a powerful and flexible tool. But, these types of programs must be able to handle all types of special characters embedded in the data, some of which may not appear until after the program is in production. This paper has examined several techniques for handling these special characters.



## ABOUT THE AUTHORS

Mary Rosenbloom is a statistical programmer at Edwards Lifesciences in Irvine, California. She has been using SAS for over 15 years, and is especially interested in using macros to generate data-driven code, DDE, and program validation methods.

Art Carpenter's publications list includes five books, and numerous papers and posters presented at SUGI, SAS Global Forum, and other user group conferences. Art has been using SAS<sup>®</sup> since 1977 and has served in various leadership positions in local, regional, national, and international user groups. He 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

Mary F. O. Rosenbloom  
Edwards Lifesciences, LLC  
One Edwards Way  
Irvine, CA 92614

(949) 250-2281  
mary.rosenbloom.sas@gmail.com

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

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

## ACKNOWLEDGMENTS

Mary would like to thank Drs. Cody Hamilton and Steven Lewis of Edwards Lifesciences for supporting her continued SAS learning and publishing efforts. She would also like to thank Art Carpenter for helping to turn this Code Doctor question into a paper, and for his enthusiastic and patient mentoring.

## REFERENCES

Carpenter, Arthur L., 2004, [Carpenter's Complete Guide to the SAS<sup>®</sup> Macro Language, 2<sup>nd</sup> Edition](#), Cary, NC: SAS Institute Inc.

Carpenter, Arthur L., 2004, "[Storing and Using a List of Values in a Macro Variable](#)", Proceedings of the 12<sup>th</sup> Annual Western Users of SAS Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc. Also in the proceedings of the Thirtieth SAS User Group International Conference (SUGI), 2005, Cary, NC: SAS Institute Inc., paper 028-30, and in the proceedings of the Pharmaceutical SAS User Group Conference (PharmaSUG), 2005, Cary, NC: SAS Institute Inc.

Carpenter, Arthur L., 2012, [Carpenter's Guide to Innovative SAS<sup>®</sup> Techniques](#), Cary, NC: SAS Institute Inc.

Fehd, Ronald and Art Carpenter, 2007, "[List Processing Basics: Creating and Using Lists of Macro Variables](#)" by Ronald Fehd and Art Carpenter which was presented at the 2007 SAS Global Forum (Paper 113-2007). The discussion of the paper looks at different approaches used in the automation of programs by using various kinds of macro variable lists. This paper appears in proceedings of a number of conferences, including: SGF(2007), WUSS (2008), MWSUG (2009), SESUG (2009).

Michel, Denis, 2005, "[CALL EXECUTE: A Powerful Data Management Tool](#)", presented at SUGI 30 (Paper 027-30), this paper includes a number of references to other CALL EXECUTE papers.

## TRADEMARK INFORMATION

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.

## APPENDIX – SAMPLE CODE

The following code can be used to test the various examples used in this paper.

```
***define the macro variables;
%let string1 = SiteA;
%let string2 = SiteB,C;
%let string3 = SiteD (with E);
%let string4 = SiteF "aka G";
%let string5 = Sites H&J;

***look at the value of the last macro variable;
%put &string5;
```

```

***create the hospital data set;
data hospitalvisits;
length sitecode $15;
sitecode = 'SiteA';           output;
sitecode = 'SiteB,C';         output;
sitecode = 'SiteD (with E)';  output;
sitecode = 'SiteF "aka G"';   output;
sitecode = 'Sites H&J';       output;
run;

*****;
***initial macro;
%macro show1;
%do i = 1 %to 5;
  title1 "&&string&i";
  proc print data=hospitalvisits;
    where sitecode="&&string&i"; /* works for first three */
  run;
%end;
%mend show1;
%show1

*****;
***macro with NRBQUOTE;
options mprint;
%macro show2;
%do j = 1 %to 5;
  title1 "%nrquote(&&string&j)";
  proc print data=hospitalvisits;
    where sitecode="%nrquote(&&string&j)";
  run;
%end;
%mend show2;
%show2

*****
*** Ampersands and percent signs;
data a;
call symputx('string5', 'Sites H&J');
run;

* Warning when used;
%put |&string5|;

* Warning when created and when used;
%let string5 = Sites H&J;
%put |&string5|;

* Warning when first used (in the LET);
%let string5a = %nrquote(Sites H&J);
%put |&string5a|;

* No warning masked in the LET;
%let string5b = %nrstr(Sites H&J);
%put |&string5b|;

* Warning when first used;
%let string5c = %nrquote(&string5);
%put |&string5c|;
%put |%nrquote(&string5)|;

* No warning masked in the LET;
%let string5d = %nrstr(&string5);
%put |&string5d|;
%put |%nrstr(&string5)|;

```

```

%let string5e = %superq(string5);
%put |&string5e|;

*****;
***leading blanks;
options mprint MAUTOSOURCE mlogic symbolgen;

%* add a leading space to foil the WHERE;
%let blk = %str( );
%let string1 = &blk.SiteA;
%let string2 = &blk.SiteB,C;
%let string3 = &blk.SiteD (with E);
%let string4 = &blk.SiteF "aka G";
%let string5 = &blk.Sites H&J;
%put |&string4|;

***macro to demonstrate which cases work;
%macro show3;
%do i = 1 %to 5;
  title1 "&i - %qleft(%nrbquote(&&string&i))";
  %put %qleft(&&string&i);
  %do k = 1 %to 7;
    %put *****;
    %put i=&i k=&k;
    proc print data=hospitalvisits;
    %if &k=1 %then %do;
      title2 "&k - Using nrstr then left";
      where sitecode="%nrstr(%left(&&string&i))";
      %put where sitecode="%nrstr(%left(&&string&i))";
    run;
    %end;
    %else %if &k=2 %then %do;
      title2 "&k - Using nrbquote then left";
      where sitecode="%nrbquote(%left(&&string&i))";
      %put where sitecode="%nrbquote(%left(&&string&i))";
    run;
    %end;
    %else %if &k=3 %then %do;
      title2 "&k - Using left then nrbquote";
      where sitecode="%left(%nrbquote(&&string&i))";
      %put where sitecode="%left(%nrbquote(&&string&i))";
    run;
    %end;
    %else %if &k=4 %then %do;
      title2 "&k - Using qleft and nrbquote";
      where sitecode="%qleft(%nrbquote(&&string&i))";
      %put where sitecode="%qleft(%nrbquote(&&string&i))";
    run;
    %end;
    %else %if &k=5 %then %do;
      title2 "&k - Using qleft only";
      where sitecode="%qleft(&&string&i)";
      %put where sitecode="%qleft(&&string&i)";
    run;
    %end;
    %else %if &k=6 %then %do;
      title2 "&k - Using superq only";
      where sitecode="%superq(string&i)";
      %put where sitecode="%superq(string&i)";
    run;
    %end;
    %else %if &k=7 %then %do;
      title2 "&k - Using superq and qleft";
      where sitecode="%qleft(%superq(string&i))";
      %put where sitecode="%qleft(%superq(string&i))";

```

```

        run;
    %end;
%end;
%end;
%end;
%show3;
options nomprint nomlogic nosymbolgen;
*filename mprint "c:\temp\quoting.sas";
*options mprint mfile;
%show3
*****;
* Translate double quotes;
%let string4 = SiteF "aka G";

%put ods rtf path="C:\output" file="%bquote( Report for &string4 &sysdate9..rtf)";

%let string4a=%qsysfunc(translate(&string4,%str(%'),%str(%")));
%put &string4;
%put &string4a;
%put ods rtf path="C:\output" file="%bquote( Report for &string4a &sysdate9..rtf)";

*****;
options mprint;
%macro printit(whr=);
    %put *****;
    title1 "%qleft(%nrquote(&whr))";
    %put %qleft(%nrquote(&whr));
    proc print data=hospitalvisits;
        where sitecode="%qleft(%nrquote(&whr))";
    run;
%mend printit;

* Call printit - this fails;
%printit(whr= SiteB,C)

* This does not always work;
data _null_;
    set hospitalvisits;
    call execute('%printit(whr='||sitecode||')');
run;

* quote in the macro call - mostly works;
data _null_;
    set hospitalvisits;
    call execute('%printit(whr=%qleft(%nrquote('||sitecode||'))');
run;

* Adding %NRSTR;
data _null_;
    set hospitalvisits;
    call execute('%nrstr(%printit(whr=%qleft(%nrquote('||sitecode||'))));
run;

```