Paper 48-2018

How Best to Use Macro Quoting Functions?

Arthur Li, City of Hope National Medical Center, Duarte, CA

ABSTRACT

The Macro quoting functions is one of the most difficult concepts to grasp when one learns to write a macro program. One of the reasons for its complexity is closely related to understanding the macro processing, such as macro compilation and execution phases. Furthermore, the situations for utilizing macro quoting functions are not obvious for programmers. In this talk, in addition to reviewing the macro processing, we will examine the most useful macro quoting functions via various examples.

INTRODUCTION

The macro language is a character-based language. That means even numeric values are treated as characters by the macro processor. The purpose of using the macro language is used to generate SAS codes based on our instruction. For example, when we write characters such as % and & signs, which are used to signal the macro processor to perform tasks such as creating macro variables, retrieving values of macro variables, and calling macro. But in some situations, & and % signs need to be treated simply as texts. The mnemonics, such as OR or GE, are operators in macro language expressions. In some situations, these special characters or mnemonics only need to be interpreted as texts as well. Therefore, we need to use the macro quoting functions to mask the special meanings so that the macro processor will be able to interpret them correctly.

WHY DO WE NEED TO USE THE QUOTING FUNCTIONS?

Here're a couple of situations where the macro quoting is needed. For example, to create a macro variable, we can use the %LET statement. The following example attempts to store "`proc print data=foo; run;`" in the macro variable &PRINT_FOO.

Code Chunk 1:

```
%let print_foo = proc print data=foo; run;;
%put &print_foo;
```

SAS Log:

```
216  %let print_foo = proc print data=foo; run;;
217  %put &print_foo;
proc print data=foo
```

The macro variable &PRINT_FOO is not created as we intended, and it only stores the value of "`proc print data=foo`" since the macro processor treats the first semicolon, which is after "foo", as the end of the %LET statement. In this situation, we need to mask the meaning of the semicolons and let the macro processor treat the semicolons simply as text.

In the following example, the macro variable &NAME stores the value of "Doe, John". In the second %LET statement, &NAME needs to be resolved first in the %SUBSTR function. However, the resolution of &NAME contains a comma which creates a function call that contains four arguments. We need to mask the meaning of commas to prevent the macro processor from misinterpreting the comma in the resolved values as a separator for the %SUBSTR function.

Code Chunk 2:

```
%let name = Doe, John;
%let initial = %substr(&name, 1, 1);
%put &initial;
```

SAS Log:

```
228  %let name = Doe, John;
229  %let initial = %substr(&name, 1, 1);
ERROR: Macro function %SUBSTR has too many arguments.
230  %put &initial;
```

TYPES OF MACRO QUOTING FUNCTIONS

During the macro processing, we need to let the macro processor use the macro quoting functions to mask the following characters and mnemonics:

```
& % blank , ; + -  * / < > =  ¬ ^ ~  # | ' " ( )
        AND OR NOT EQ NE LE LT GE GT IN
```

These quoting functions can be categorized into *compilation* and *execution* quoting functions.

The compilation quoting functions are %STR and %NRSTR. This type of quoting function is used when a user types special characters in the open code, such as in the %LET or %PUT statements. They can also be used in a programming logic statement such as %IF statement. And we use these functions when a user supplies parameter values on a macro call. During the macro compilation phase, using the compilation quoting functions will treat characters as text in the macro program.

Other than the %STR and %NRSTR functions, other macro quoting functions are mostly used during the macro execution. During the macro execution, the execution macro functions mask resolved values from macro variables.

The macro quoting functions generally come in pairs. The function that begins with NR, such as %NRSTR masks all the special characters listed above. The function that does not begin with NR, such as %STR, masks the special characters except for ampersands (&) and percent signs (%). The letters NR are short for “non-resolved”. The function that starts with NR prevents macro or macro variable resolution.

There are also functions that with letter B (for “by itself”), such as %BQUOTE and %NRBQUOTE, which are useful for quoting unmatched quotation marks or parentheses.

COMPILATION QUOTING FUNCTIONS

The %STR and %NRSTR functions mask special characters and mnemonics during the macro compilation phase.

%STR(*character-string*)
%NRSTR(*character-string*)

Both functions mask the special character or mnemonics that are listed above, except that the %STR function does not mask & and %. When masking ' " ( ), we need to place a percent sign (%) in front of these characters.

THE %STR FUNCTION

The example in *Code Chunk 1* can be modified by using the %STR function. The %STR function in *Code Chunk 3* masks the meaning of semicolons and the macro processor treats the semicolons within the %STR function as texts.

Code Chunk 3:

```
%let print_foo = %str(proc print data=foo; run;);
&print_foo
```

SAS Log:

```
235
236  %let print_foo = %str(proc print data=foo; run;);
237  &print_foo

NOTE: There were 1 observations read from the data set WORK.FOO.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.10 seconds
      cpu time            0.01 seconds
```

Alternatively, you can quote the semicolons only. For example:

Code Chunk 4:

```
%let print_foo = proc print data=foo%str(;) run%str(;);
%put &print_foo;
```

The following macro FIRST_LAST is used to separate first and last name of a given full name. The full name is separated by using a comma, which is argument value for the FRIST_LAST macro. The two %LET statements are used to create two macro variables by using the %SCAN function to extract the first and last name of the macro variable &NAME, which is the parameter of the FIRST_LAST macro. The default separator, which is including a comma, is used as the separator for the %SCAN function.

Code Chunk 5:

```
%macro first_last(name);
    %let first = %scan(&name, 2);
    %let last = %scan(&name, 1);
    %put &first &last;
%mend;

%first_last(Doe, John)
```

SAS log:

```
263  %first_last(Doe, John)
ERROR: More positional parameters found than defined.
```

Without masking the value during the macro call, the error was generated because commas are also used as the separators between parameters values during the macro calls. Since there is only one parameter that is defined in the FIRST_LAST macro definition, “Doe, John” is treated as two values during the macro call. We can use the %STR function to mask the entire value during the macro call so that the comma will be interpreted as text, not as the separators for the function parameters. For example:

Code Chunk 6:

```
%first_last(%str(Doe, John))
```

SAS log:

```
265  %first_last(%str(Doe, John))
John Doe
```

If we want to treat the comma (,) as the only delimiter instead of using all default delimiter for the %SCAN function, we need to use the %STR function to enclose the comma in the %SCAN function as well; otherwise, having two consecutive commas will cause the %SCAN function to use the NULL value as the delimiter, which means no characters will be used as the delimiter for the %SCAN function. The following program modifies the program above by using a comma (,) as the delimiter.

Code Chunk 7:

```
%macro first_last1(name);
    %let first = %scan(&name, 2, %str(,));
    %let last = %scan(&name, 1, %str(,));
    %put &first &last;
%mend;
```

Using the %STR function is also useful when we want to preserve the leading blanks in the %PUT statement in the SAS log like the following example:

Code Chunk 8:

```
%macro first_last2(name);
    %let first = %scan(&name, 2, %str(, ));
    %let last = %scan(&name, 1, %str(, ));
    %put %str(    &first &last);
%mend;
%first_last2(%str(Doe, John))
```

SAS log:

```
312  %first_last2(%str(Doe, John))
    John Doe
```

The mnemonic, such as GE, OR, are generally served as the purpose for comparison in macro expressions. In some applications, we need to treat the mnemonics as text as well. In the following example, the COMPARE_BRAND compares the parameter &BRAND with the value "GE."

Code Chunk 9:

```
%macro compare_brand(brand);
    %if &brand = GE %then %put Brand is General Electric;
    %else %put Brand is &brand;
%mend;

%compare_brand(Coke)
```

SAS log:

```
321  %compare_brand(Coke)
Brand is General Electric
```

The result above is not what we expected. The problem is that GE is treated as a comparison operator by the macro processor. The expression "&brand = GE" in the %IF statement is equivalent to `((&brand = ) GE)`. That is the resolved value from the &BRAND variable that compares with a null value by using the = operator first, then compares with the second null value with the GE operator. To fix the problem, we can use the %STR function to enclose "GE". For example:

Code Chunk 10:

```
%macro compare_brand1(brand);
    %if &brand = %str(GE) %then %put Brand is General Electric;
    %else %put Brand is &brand;
%mend;

%compare_brand1(Coke)
```

SAS log:

```
489  %compare_brand1(Coke)
Brand is Coke
```

What about when we call this macro by provide GE as its argument? For example:

Code Chunk 11:

```
%macro compare_brand1(brand);
    %if &brand = %str(GE) %then
        %put Brand is General Electric;
    %else %put Brand is &brand;
%mend;

%compare_brand1(GE)
```

SAS log:

```
315  %compare_brand1(GE)
Brand is GE
```

The result from the code above is not correct because we also need to mask GE when we make a macro call. For example:

Code Chunk 12:

```
%macro compare_brand1(brand);
    %if &brand = %str(GE) %then
        %put Brand is General Electric;
    %else %put Brand is &brand;
%mend;

%compare_brand1(%str(GE))
```

SAS log:

```
317  %compare_brand1(%str(GE))
Brand is General Electric
```

THE %NRSTR FUNCTION

The %NRSTR function masks all the characters that the %STR function masks. In addition, the NRSTR function masks & and %. In the following example, without using the %NRSTR function, the macro variable &ICE_CREAM will not be created successfully since the %STR function doesn't mask the ampersand (&) and the macro processor attempts to resolve the macro variable &JERRY in the %LET statement.

Code Chunk 13:

```
%let ice_cream = %str(Ben&Jerry);
%put &ice_cream;
```

SAS log:

```
694  %let ice_cream = %str(Ben&Jerry);
695  %put &ice_cream;
WARNING: Apparent symbolic reference JERRY not resolved.
Ben&Jerry
```

In this situation, we need to use the %NRSTR function, which masks % and & and the macro processor will not treat &JERRY as a macro variable.

Code Chunk 14:

```
%let ice_cream = %nrstr(Ben&Jerry);
%put &ice_cream;
```

SAS log:

```
697  %let ice_cream = %nrstr(Ben&Jerry);
698  %put &ice_cream;
Ben&Jerry
```

THE SITUATION FOR USING %NRSTR FUNCTION, BUT NOT THE %STR FUNCTION

The %NRSTR function masks more characters which doesn't mean that we can use it to replace the %STR function in all the situations. The following example demonstrates the need for using the %STR function instead of using the %NRSTR function. The macro variable &PRINTLAST stores the values for printing the last created data set. Since the %NRSTR function masks the ampersand (&), the automatic macro variable &SYSLAST will not be resolved. Hence, the last generated data set will not be printed.

Code Chunk 15:

```
%let printlast = %nrstr(proc print data=&syslast; run;);
&printlast
```

SAS log:

```
731  %let printlast = %nrstr(proc print data=&syslast; run;);
732  &printlast
NOTE: Line generated by the macro variable "PRINTLAST".
1      roc print data=&syslast; run;
                      -
                      22
                       -
                       200
ERROR: File WORK.SYSLAST.DATA does not exist.

ERROR 22-322: Expecting a name.

ERROR 200-322: The symbol is not recognized and will be ignored.
```

Instead of using the %NRSTR function, using the %STR function only masks the semicolons, but not the ampersand sign (&). The macro variable &SYSLAST is resolved correctly when creating the macro variable &PRINTLAST. Therefore, the last generated data is printed without any error.

Code Chunk 16:

```
%let printlast = %str(proc print data=&syslast; run;);
&printlast
```

SAS log:

```
734  %let printlast = %str(proc print data=&syslast; run;);
735  &printlast

NOTE: There were 1 observations read from the data set WORK.FOO.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.09 seconds
      cpu time            0.00 seconds
```

EXECUTION QUOTING FUNCTIONS

The macro execution quoting functions are used when we want to mask the resolved value of a macro variable during macro invocation, the result from a macro function or in macro programming logic. It is useful to use execution quoting functions to mask the characters that were entered by users but unknown to us and these characters might cause programming errors.

THE % BQUOTE AND %NRBQUOTE FUNCTIONS

The %BQUOTE and %NRBQUOTE functions are used to mask special characters and mnemonic operators in a resolved value at macro execution.

%BQUOTE(*character-string* | *text-expression*)
%NRBQUOTE(*character-string* | *text-expression*)

Both functions mask special characters or mnemonics in a character string or resolved value of a text expression. These special characters or mnemonics include the following:

& % blank , ; + - * / < > = ¬ ^ ~ # | ' " ( )
AND OR NOT EQ NE LE LT GE GT IN

Among all the symbols above, the %BQUOTE function does not masks ampersand (&) and percent (%) signs.

In the previous example (Code Chunk 12), we used the %STR function to quote GE when we call COMPARE_BRAND1 macro. If we pass this macro to a user, we can't control what the user will use when calling the macro. A better approach is to use the %BQUOTE function to quote &BRAND within the macro definition. For example:

Code Chunk 17:

```
%macro compare_brand2(brand);
    %if %bquote(&brand) = %str(GE) %then
        %put Brand is General Electric;
    %else %put Brand is &brand;
%mend;

%compare_brand2(GE)
```

SAS log:

```
333  %compare_brand2(GE)
Brand is General Electric
```

In the following example, the ADDPLUS macro adds either + or +1 to a given phone number. The macro variable &FIRST is used to store the first character of the parameter value and then used to compare with the value of either 1 or + sign. If a phone number is provided with a plus operator (+), the plus operator (+) will be stored in &FIRST. After

resolving &FIRST in the %IF statement, the expression "`&first EQ 1`" will be equivalent to "`+ EQ 1`" which will lead to an error because the macro processor treats the plus sign (+) as an operator which requires a numeric operand. In this situation, using the %BQUOTE function will treat the resolved value from the &FIRST macro variable simply as text.

Code Chunk 18:

```
%macro addplus(number);
    %let first = %substr(&number, 1, 1);
    %if %bquote(&first) EQ 1 %then %do;
        %let newnumber = +&number;;
    %end;
    %else %if %bquote(&first) NE %str(+) %then %do;
        %let newnumber = +1&number;;
    %end;
    %else %let newnumber = &number;
    %put &newnumber;
%mend;
%addplus((909)319-2541)
%addplus(+1(909)319-2541)
%addplus(1(909)319-2541)
```

SAS log:

```
159  %addplus((909)319-2541)
+1(909)319-2541
160  %addplus(+1(909)319-2541)
+1(909)319-2541
161  %addplus(1(909)319-2541)
+1(909)319-2541
```

The use of %NRBQUOTE is similar to %BQUOTE except that the %NRBQUOTE function is applied in the situation to mask & and % signs in the resolved value of an argument. The following example, ADD_AMPERSAND, illustrates the need for using the %NRBQUOTE function. If &NUMBER contains an ampersand, the macro doesn't add an ampersand; otherwise, it will add an ampersand to the provided number.

Code Chunk 19:

```
%macro add_ampersand(number);
    %let first = %substr(&number, 1, 1);
    %if %nrbquote(&first) EQ %nrstr(&) %then %do;
        %let newnumber = &number;;
    %end;
    %else %let newnumber =%nrstr(&)&number;
    %put &newnumber;
%mend;
%add_ampersand(&123)
%add_ampersand(123)
```

SAS log:

```
402  %add_ampersand(&123)
&123
403  %add_ampersand(123)
&123
```

Macro facility also has %QUOTE and %NRQUOTE functions. The uses of these two functions are similar to %BQUOTE and %NRBQUOTE functions. Both of these functions are used during the macro execution phase. The %BQUOTE and %NRBQUOTE functions do not require that quotation marks without a match be marked with a preceding % sign, and %QUOTE and %NRQUOTE do. With the availability of the %BQUOTE and %NRBQUOTE functions, the %QUOTE and %NRQUOTE functions are rarely used.

THE %SUPERQ FUNCTION
The %SUPERQ function also masks special characters and mnemonics during macro execution, and it prevents further resolution of the value.

%SUPERQ(*argument*)

The *argument* is the name of a macro variable with no leading ampersand or a text expression that produces the name of a macro variable without a leading ampersand. Unlike other macro quoting functions, we can only use the %SUPERQ function to apply on one single macro variable.

The following example uses the %SUPERQ function that masks the comma in the resolved value of &NAME. Without using the %SUPERQ function, the resolution of the &NAME macro variable contains a comma which results to a function call from the %SUBSTR function containing four arguments.

Code Chunk 20:
```
%let name = Doe, John;
%let initial = %substr(%superq(name), 1, 1);
%put &initial;
```

SAS log:
```
1555  %let name = Doe, John;
1556  %let initial = %substr(%superq(name), 1, 1);
1557  %put &initial;
D
```

The following example illustrates using the %SUPERQ function that masks the meaning of OR. Without using the OR function, the macro processor treats OR as a logical operator which requires numeric operands.

Code Chunk 21:
```
%macro check_state(state);
    %if %superq(state) = %str(OR) %then %put State is Oregon;
    %else %put State is &state;
%mend;

%check_state(OR)
```

SAS log:
```
1564  %check_state(OR)
State is Oregon
```

Both %NRBQUOTE and %SUPERQ functions mask % and & signs. When using the %NRBQUOTE function, the macro processor masks the argument after the macro processor resolves macro variable references or values. On the other hand, the macro processor masks the argument of the %SUPERQ function before it resolves any macro variable references or values.

QUOTING FUNCTION RESULTS
Consider the following macro call to the FIRST_LAST macro which was defined previously in *Code Chunk 4:*

Code Chunk 22:
```
%first_last(%str(O%'Hara, Scarlett))
```

SAS log:
```
1772   %first_last(%str(O%'Hara, Scarlett))
ERROR: Literal contains unmatched quote.
ERROR: The macro FIRST_LAST will stop executing.
```

Why does this macro call generate an error message even after “O’Hara, Scarlett” was enclosed in the %STR function? During the macro execution, the macro variable name is successfully created and contains the value

"O'Hara, Scarlett" in the local symbol table. The first %LET statement creates the local macro variable FIRST correctly. However, the second %LET statement fails. When the second %LET statement executes, the macro reference is resolved first in the %SCAN function. The result from the %SCAN function becomes "O'Hara". The masking applies to the apostrophe that is then removed when the macro variable is resolved in the %SCAN function. Since there is no masking for the apostrophe, the apostrophe is treated as a single quote by the macro processor. The semicolon that ends the %LET statement becomes part of the literal. Since there is no ending single quote, the error message is generated.

We can fix the problem by using the %BQUOTE function for the returned value from the %SCAN function, such as `%BQUOTE(%SCAN(&NAME, 1))`. Alternatively, we can also use the %QSCAN function, which achieves the same result as the %SCAN function but also masks the result. For example:

Code Chunk 23:

```
%macro first_last3(name);
    %let first = %qscan(&name, 2);
    %let last = %qscan(&name, 1);
    %put &first &last;
%mend;

%first_last3(%str(O%'Hara, Scarlett))
```

SAS log:

```
1866   %first_last3(%str(O%'Hara, Scarlett))
Scarlett O'Hara
```

In addition to the %SCAN and %QSCAN pairs, there are many macro functions and autocall macros that have "Q" equivalent functions that mask the results, such as %QCMPRES, %QLEFT, %QLOWCASE, %QSUBSTR, %QTRIM, %QUPCASE, and %QSYSFUNC.

UNQUOTING

When special characters or mnemonics are masked with macro quoting functions, they will be unmasked if they are resolved in the macro functions, such as the %SCAN, %SUBSTR, or %UPCASE functions. Furthermore, they are also unmasked when they are passed to the DATA step compiler. Generally speaking, they are remained masked as long as they are remained in the macro facility. If the texts are masked by the quoting function, the generated SAS statements from the macro processor appear to be correct, but the compiler often does not recognize them as valid syntax. In the situation where you want to explicitly unmask them, such as using the unmasked value within the same macro, you can use the %UNQUOTE function.

%**UNQUOTE**(*character-string* | *text-expression*)

The macro SCALE in the following example divides a given variable, names, by 100. In the %DO loop, an individual variable name is extracted by using the %QSCAN function and then is used as part of the new variable name.

Code Chunk 24:

```
data example;
    input a b c;
datalines;
100 200 300
;

%macro scale(varname);
    %local i new;
    data;
        set example;
        %do i = 1 %to %sysfunc(countw(&varname));
            %let new = %qscan(&varname, &i);
            val&new = &new/100;
        %end;
    run;
%mend;
%scale(a b)
```

SAS log:

```
2108  %scale(a b)
NOTE: Line generated by the macro variable "NEW".
1                  vala
                   ---
                   180
NOTE: Line generated by the macro variable "NEW".
1                  valb
                   ---
                   180

ERROR 180-322: Statement is not valid or it is used out of proper order.

NOTE: The SAS System stopped processing this step because of errors.
```

The error was generated from the program above because the masking is placed on the value that is returned from the %QSCAN function. Therefore, it causes the generated result to be tokenized incorrectly. To fix the problem, we can use the %UNQUOTE function like the example below.

Code Chunk 25:

```
%macro scale(varname);
    %local i new;
    data;
        set example;
        %do i = 1 %to %sysfunc(countw(&varname));
            %let new = %qscan(&varname, &i);
            %unquote(val&new) = &new/100;
        %end;
    run;
%mend;
```

CONCLUSION

Grasping macro quoting functions completely is not an easy task. First, one needs to know why the quoting functions are needed in the macro program, which is related to understanding the correct macro syntax. Furthermore, we also need to know which quoting functions to use during the macro compilation and execution phases. A general rule to choose the correct macro quoting function is that if we can see the special character or mnemonics, we need to use the compilation quoting functions; on the other hand, if we cannot see the special character or mnemonics, we need to use the execution quoting functions.

REFERENCES

Burlew, Michele M. 2014, SAS® Macro Programming Made Easy, 3nd Edition, Cary, NC: SAS Institute Inc.

Carpenter, Art L., 2006, Carpenter's Complete Guide to the SAS® Macro Language, 3nd Edition, Cary, NC: SAS Institute Inc.

Rosenbloom Mary F. O. and Carpenter, Art L., 2013, "Macro Quoting to the Rescue: Passing Special Characters", Proceedings of SAS Global Forum 2013.

SAS® Macro Language 2: Developing Macro Application Course Notes, 2009, Cary, NC: SAS Institute Inc.

Tyndall, Russ, 2014, "Macro Quoting Made Easy", https://blogs.sas.com/content/sgf/2014/08/15/macro-quoting-made-easy/#prettyPhoto.

Whitlock, Ian, "A Serious Look Macro Quoting", Proceedings of SUGI28.

ACKNOWLEDGMENTS

I would like to thank Art Carpenter for his valuable programming suggestions and insight.

CONTACT INFORMATION

Arthur Li
City of Hope National Medical Center
Department of Information Science
1500 East Duarte Road
Duarte, CA 91010 - 3000
Work Phone: (626) 256-4673 ext. 65121
E-mail: arthurli@coh.org