PhUSE 2017

Paper IS06

A Practical Guide to Macro Quoting and Working with Macro Variables

Dawid Militowski, MSD, Hoddesdon, UK

ABSTRACT

Macro quoting in SAS changed over the years and new functions were added as extensions of the old ones. Having many different functions to choose from can be confusing, especially where in many scenarios more than one function can be used to achieve the same result. Complete macro language was introduced in SAS version 5. This paper is a simple guide as to which functions are absolutely necessary to know as they cover most real life scenarios, and which ones are less important. It also provides information on working with quoted macro variables in an efficient way, and on using a data step as a powerful tool for processing such macro variables.

INTRODUCTION

Whenever macro variables or macros are used there is often a need to mask tokens [Patterson & Remigio 2007]. It makes sense to use macro functions that would mask all of those tokens, as using ones that mask only some creates a risk that a macro might not work properly in some cases. There is also a task of accessing and extracting information within macro variables that have quoting applied, in a way that prevents unmatched tokens from producing errors or warnings. Finally, the use of a data step is discussed as a powerful way of working with macro variables. This discussion is followed by 3 practical examples from day to day work.

The paper discusses the following macro functions:

- %NRSTR() – used to mask tokens in compilation phase
- %NRBQUOTE() – used to mask tokens in execution phase
- %SUPERQ() – used to mask tokens until they are explicitly unmasked
- %UNQUOTE() – used to explicitly unmask a macro variable
- %QSCAN() – used to scan a macro variable with or without tokens
- %QSUBSTR() – used to extract a sub string from a macro variable with or without tokens.

Tokens are divided into 3 categories [Patterson & Remigio 2007]:

- Type I:
  - Operators {+ - * / < > = ¬ ^ | ~}
  - Mnemonics {AND OR NOT EQ NE LE LT GE GT IN}
  - Miscellaneous {blank , ; “” ‘ () #}
- Type II: Unmatched Characters {“ ‘ ( )}
- Type III: Macro Triggers {& %}.

The use of a data step is discussed as an alternative and powerful tool to process macro variables in a similar way as any other variable. Specifically the functions that are discussed are:

- SYMGET() – used to load a macro variable into a dataset variable
- CALL SYMPUTX() – used to create a macro variables in a data step
- RESOLVE() – used to unmask tokens within a data step - similar to %UNQUOTE() in macro language.

Outputs from the SAS log window are presented in this paper like:

```
➢ Log output message.
```

MASKING TOKENS

As described in [Patterson & Remigio 2007] there are 3 types of tokens to mask and 7 macro quoting functions to do it. Some of the new functions are extensions of the older ones. This paper primarily focuses on %NRSTR(), %NRBQUOTE() and %SUPERQ(). Those 3 macro quoting functions were introduced to extend the functionality of older functions. All 7 functions can be summarized as follows:

- %NRSTR() is an extension of %STR()
- %NRBQUOTE() is an extension of %QUOTE(), %NRQUOTE(), %BQUOTE()
- %SUPERQ() – is unique and its use is required in some scenarios.

Before going into more detail, one important thing to remember is that unmatched tokens ' " ( ) need to be masked with % when using %NRSTR(), but do not have to be masked this way when using %NRBQUOTE(), for example:

- %NRSTR(This is test%'d)
- %NRBQUOTE(This is test'd).

Now let's look into characteristics of these functions. As they all mask 3 types of tokens, choosing which one to use depends on when tokens are to be resolved:

- At compilation - %NRSTR()
- At execution - %NRBQUOTE()
- Never - %SUPERQ().

%NRSTR() AND %NRBQUOTE()

%NRSTR() is used when creating a macro, that is when it is compiled, for example:

```
%macro test(var=);
    %if &var=%NRSTR(New & old 's ) %then %put %NRSTR(%increase);
    %else %put Wrong result;
%mend test;
```

In this case it is not advisable to use %NRBQUOTE() as it would produce a warning. It should be used when the macro is called, that is when it is executed:

```
%test(var=%NRBQUOTE(New & old 's));
```

- %increase

The above code produces the desired result. In the above scenario %NRSTR() would work as well. To illustrate the difference let's create a macro variable that will be passed as a parameter to the macro:

```
%let newvar=%NRSTR(New & old 's);

%test(var=%NRBQUOTE(&newvar));
```

- %increase

```
%test(var=%NRSTR(&newvar));
```

- Wrong result

In this example only %NRBQUOTE() gives desired results as it resolves the macro variable at execution but still masks tokens inside it, while %NRSTR() is masking '&new var' and prevents resolution of a macro trigger '&'.

%STR() AND %QUOTE()

%STR() and %QUOTE() are arguably the most popular macro functions used for masking of special characters and mnemonic operators. Let's discuss in more detail why in this paper they are not recommended. The previous example could be re-coded to use %STR():

```
%macro test1(var=);
    %if &var=%STR(New & old 's) %then %put True;
%mend test1;
%test1(var=%NRBQUOTE(New & old 's));
```

The problem with this approach is that during execution errors are printed:

- ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: &var=New & old 's
- ERROR: The macro TEST1 will stop executing.

Finally, here is the same scenario, this time using %QUOTE(). % needs to be added in this case to mask unmatched tokens:

```
%macro test2(var=);
    %if &var=%NRSTR(New & old 's) %then %put True;
%mend test2;
%test2(var=%quote(New & old %'s));
```

Running the above code results in similar errors as the example using %STR(). Both examples show that when macro triggers need to be masked at compilation and execution, respectively %STR() and %QUOTE() cannot be used. %NRSTR() and %NRBQUOTE() mask macro triggers and in all other scenarios provide exactly the same functionality as %STR() and %QUOTE(), therefore it is recommended that %NRSTR() and %NRBQUOTE() are preferentially used.

%SUPERQ()

In some ways %SUPERQ() is the simplest function to understand. It masks all tokens within the macro variable/parameter until they are explicitly unmasked. It's useful when all macro triggers are to be masked until the user specifically wants them to be resolved. To illustrate, let's look at some scenarios.

First let's create a macro variable with macro triggers that can be treated as macro variables or macros:

```
data _null_;
    CALL SYMPUTX('test1','%CI and &x1');
run;
```

This step is necessary as %SUPERQ() only takes a macro variable as an argument. Let's also create a simple macro to print:

```
%macro test3(var=);
    %put &var;
%mend test3;
```

Now let's call the macro using all 3 macro quoting functions:

```
%test3(var=%NRSTR(&test1));
```

- &test1

```
%test3(var=%NRBQUOTE(&test1));
```

- WARNING: Apparent invocation of macro CI not resolved.
- WARNING: Apparent symbolic reference X1 not resolved.

```
%test3(var=%SUPERQ(test1));
```

- %CI and &x1

%NRBQUOTE() gives warnings, while both %SUPERQ() and %NRSTR() produce results, though those results differ. As stated above the reason for the difference is that %NRSTR() treats the input function argument &test1 as text. It is not desirable as the task is not to mask word '&test1' but the value of the macro variable &test1. Note that if the value '%CI and &x1' was to be processed before the macro variable is unmasked, then using %NRSTR() would prevent it.

ADDITIONAL NOTES

There are some cases when using %NRSTR() is the only option while relying on macro language. If there was a string to be passed like "text and &test1" then neither %NRBQUOTE nor %SUPERQ() would work:

```
%test3(var=%NRSTR(text and &test1));
```

%SUPERQ() can only handle a macro variable as an input and if it was used in the above example, it would try to resolve the whole string. Caution is advised when using %SUPERQ() or %NRSTR() in cases where the macro triggers are to be resolved inside the code. The masked macro variables need to be explicitly unmasked by using a specific code in those cases.

There is also a very interesting scenario when using single and double quotes, which complicates matters further. While a practical scenario may be unlikely, consider the below example:

```
%macro test4(var=);
    %put &var;
    %put %NRBQUOTE(&var);
%mend test4;
```

```
%test4(var=%NRSTR(text and &test1)); *Works fine;
%test4(var=%NRSTR('text and &test1')); *Produces warnings;
```

- WARNING: Apparent invocation of macro CI not resolved.
- WARNING: Apparent symbolic reference X1 not resolved.

The code with a single quotation gives warnings as single quotes prevent macro quoting from being applied to a macro variable &test1. Code without %NRBQUOTE works as it just displays text, while code with %NRBQUOTE resolves during execution, therefore giving warning.

This subject is elegantly described in detail by [Shan Lee 2007].

USE OF A DATA STEP

In many ways using a data step feels more familiar and intuitive. It relies primarily on 3 functions that are used for working with macro variables in a data step. Beyond those 3 functions, values can be processed in the same way as any variable in a dataset. While a regular dataset could be created in this process, a common approach is to use 'data _null_' step.

Firstly, to create a macro variable, the function CALL SYMPUTX should be used. It is an extension to CALL SYMPUT, with two functions. Firstly, it left justifies and trims trialing blanks from a numeric value. Secondly, it allows user to specify the scope of the created macro variable. Let's look at some of the characteristics of CALL SYMPUTX:

```
%let x2=abc;
data _null_;
   *This code works as both % and & are masked as text.;
    CALL SYMPUTX('test1','%CI and &x1');

   *As in any case when using macro variables in a data step, double quotes will cause the macro triggers to
   resolve and produce warnings.;
    CALL SYMPUTX('test2',"%CI and &x1");

   *To pass both triggers while only a macro variable is to be resolved, more complicated code needs to be
   used:;
    CALL SYMPUTX('test3','%CI and'||"&x2");

   *The value stored in a variable can be passed into a macro variable.
   Specifying the third parameter in CALL SYMPUTX as 'G' ensures that the macro variable is created as a
   global macro variable;
    test='%CI and &x1';
    CALL SYMPUTX('test4',test,'G');
run;
```

On the topic of local and global variables it is very important to remember that if a macro variable exists in multiple symbol tables, the value stored in the most local one is used. It might then be advisable to use 'F' as a third option in CALL SYMPUTX, to ensure that the variable is created in the most local symbol table.

Another action that can be performed in a data step is to load a macro variable content into a dataset variable by calling the function SYMGET(). It can then be processed using a data step functionality and afterwards be output using CALL SYMPUTX, for example:

```
%let title=New title;
Data _null_;
    length title title1 $100;
    title =SYMGET('title');
    title1="ABC"||title;
    CALL SYMPUTX('title1',title1,'G');
run;
```

It is advisable to specify the lengths of the variables being used to avoid potential truncation.

PROCESSING QUOTED VARIABLES WITH TOKENS

When using a data step there is no difference in processing a macro variable with or without tokens. A macro variable is loaded into a dataset variable as a string and can be processed as such, no matter what kind of tokens it holds.

In the macro language there are 2 main functions that can be used when processing macro variables with tokens - %QSCAN() and %QSUBSTR(). They both allow processing of macro variables that cannot be dealt with by using %SCAN() and %SUBSTR(). Let's look at examples:

```
%let string=%NRSTR(abc %'12 34.);
    %put %QSCAN(&string,2); *A correct result;
    %put %SCAN(&string,2); *An incorrect result;
    %put %QSUBSTR(&string,2,5); *A correct result;
    %put %SUBSTR(&string,2,5); *An incorrect result;
```

Even though %SCAN() and %SUBSTR() are not producing any immediate warnings or errors they effectively unmask the single quotation mark, leaving an unbalanced quote affecting all code that follows. This problem in the code is not easy to debug.

As is shown in an example later in the paper, %QSCAN() and %QSUBSTR() do not replace %SCAN() and %SUBSTR() in all scenarios, so in fact they cannot be regarded as an extension of them.

UNMASKING VARIABLES

Macro variables can be unmasked before any processing is done. This can be achieved both by the use of macro language or a data step. In macro language the function %UNQUOTE() is used. A simple example illustrates how it works:

```
data _null_;
    CALL SYMPUTX('test1','&x1 and &x2');
run;

%let x1=5;
%let x2=15;

%let new=%SUPERQ(test1);
%put &new;
        ➢ &x1 and &x2

%put %UNQUOTE(&new);
        ➢ 5 and 15

%let new1=%NRSTR(&test1);
%put &new1;
        ➢ &test1

%put %UNQUOTE(&new1);
        ➢ 5 and 15
```

It does not matter which function was used for quoting. Even a macro variable masked with %NRSTR() is resolved correctly.

A similar result can be achieved using a data step. The function RESOLVE() is used. As it processes variable in a dataset, a macro variable value needs to be first loaded into a dataset variable using SYMGET():

```
Data _null_;
    title = SYMGET('test1');
    put title;
    title2 = RESOLVE(title);
    put title2;
run;
        ➢ &x1 and &x2
        ➢ 5 and 15
```

The results are the same as when using macro language.

ADVANTAGE OF USING DATA_NULL_

Even though in the above example there is no difference between macro language and a data step, the following example shows that using a data step is a much better way of processing unmasked macro variables.

When unmasking macro variables caution is advised as all masked tokens get unmasked and have potential to create unbalanced quotes or an open parenthesis. That is one of the reasons why it is advisable to use a data step instead of macro language. As all unmatched tokens need matching, if a macro variable is unmasked in macro language it results in either unbalanced quotes or an open parenthesis:

```
data _null_;
    CALL SYMPUTX('test1','% CI and "( . , &x1');
run;

%let x1=abc;
%let new=%SUPERQ(test1);
%put &new;
```

- % CI and "(. , &x1

```
%put %UNQUOTE(&new);
```

The last line does not produce a result in the log window, because it results in an unbalanced double quote. If such a string needs to be unmasked, for example if it holds macro variables that need to be resolved, then the following data step can be used:

```
Data _null_;
    length test test1 $80;
    test = SYMGET('new');
    put test;
    test1 = RESOLVE(test);
    put test1;
run;
```

- % CI and "(. , &x1
- % CI and "(. , abc

The above code produces the desired results and the string can be processed within a data step. That way potential problems with unbalanced quotes or an open parenthesis are avoided.

PRACTICAL EXAMPLES

EXAMPLE 1: PROCESSING A SERIES OF FOOTNOTES FROM A SINGLE MACRO VARIABLE HOLDING MULTIPLE FOOTNOTES

Here, a series of footnotes for a report are being passed through a single macro variable. Let's create such a variable:

```
%let foot_note=%NRSTR(a: First footnote| b: Second footnote| C: Third footnote|
CI: confidence interval ; XXX:unknown & % '; SE: standard error ; SOC: standard of
care);
```

The task in this example is to extract and process the last footnote stored in the macro variable &foot_note. As footnotes are separated by | and contain all kinds of tokens, it is best to use %QSCAN() function:

```
%let last=%QSCAN(&foot_note,-1,|);
%put &last;
```

- CI: confidence interval ; XXX:unknown & % '; SE: standard error ; SOC: standard of care

The below code might seem like over programming at first glance, but is in fact necessary. Using only %QSCAN() might give us easily the first and last footnote, but getting middle ones would require extra coding. A cleaner way is to find the position of the last footnote and then using %QSUBSTR(), select the last and all other footnotes:

```
%let pos=%index(&foot_note,&last);
%put &pos;

%let end_notes1=%QSUBSTR(&foot_note,1,&pos-1);
%let end_notes2=%QSUBSTR(&foot_note,&pos); *Duplicated with %QSCAN() code above;


%put &end_notes1.;
```

- a: First footnote| b: Second footnote| C: Third footnote|

```
%put &end_notes2.;
```

- CI: confidence interval ; XXX:unknown & % '; SE: standard error ; SOC: standard of care

From this point &end_notes2 can be processed separately. The takeaway point of this example is that a macro variable that contains all kinds of tokens has been processed successfully. This could not be achieved using %SCAN() and %SUBSTR().

EXAMPLE 2: PROCESSING A MACRO VARIABLE THAT HOLDS REPORT TITLE IN A DATA STEP

The below example shows how a macro variable can be loaded into a variable and then easily processed using a data step functions. The task is to process report titles for display in an rtf file. Titles should be displayed in separate lines and this is achieved by the use of RTF tag '\line':

```
data _null_;
    CALL SYMPUTX('title',"This is title(|Subtitle one'|Second subtitle &");
run;

%let title1=%SUPERQ(title);
%put &title1;
```

- This is title(|Subtitle one'|Second subtitle &

```
data _null_;
    length title $ 3000;
    title = SYMGET('title1');
    title=tranwrd(title, '|', '\line ');
    CALL SYMPUTX('title2',title,'G');
run;

%put %SUPERQ(title2);
```

- This is title(\line Subtitle one'\line Second subtitle &

It is simple to make this transformation in a data step without worrying about tokens. An additional advantage is that using a data step is more intuitive than using macro language. Using a data step also makes code more transparent, especially in complex scenarios.

EXAMPLE 3: CREATING MULTIPLE VARIABLES BASED ON VALUES STORED IN A SINGLE MACRO VARIABLE

As mentioned previously, it is not always possible to replace %SCAN() with %QSCAN(). The below example involves creating a new variable whose name is partly derived from an element in a set of time points stored in a macro variable. Just to be safe it would seem logical to use %QSCAN() just in case any special characters are encountered. However, consider below when %SCAN() is used:

```
%let time_points=3 6 9 12;

data new;
    rate1_scan_%scan(&time_points.,2)=1;
run;
```

This code runs correctly. Now, let's use %QSCAN() function instead:

```
data new;
    rate1_QSCAN_%QSCAN(&time_points.,2)=1;
run;
```

This code produces error message:

- NOTE: Line generated by the macro function "QSCAN".
- rate1_QSCAN_
- ------------
- 180
- ERROR 180-322: Statement is not valid or it is used out of proper order.

The above code may seem correct, but an error is generated because the masking placed on the value selected by %QSCAN() is causing the string to be tokenized incorrectly. This can be corrected by using the %UNQUOTE function. This effectively means that %QSCAN() on its own cannot replace %SCAN() in this case.
The following code runs correctly and produces the same result as %SCAN():

```
data new;
    rate1_QSCANu_%UNQUOTE(%QSCAN(&time_points.,2))=1;
run;
```

There is a similar issue when %QSUBSTR() is used instead of %SUBSTR():

```
%let time_points=3456;

data new;
        rate1_scan_%subSTR(&time_points.,3,1)=1;
        rate1_QSCAN_%QSUBSTR(&time_points.,3,1)=1;
        output;
run;
```

It gives a similar error as above. This kind of scenario is unlikely to occur in normal use but illustrates issues that might be encountered when using %QSUBSTR().

SUMMARY

Below are the pros and cons of the methods and functions discussed in this paper. It is good to be aware of those:

Function	Advantages	Disadvantages/Limitations
Macro language		
%NRSTR()	Masks all tokens in the macro at compilation.	When used to mask a macro variable it masks the name of the variable and not its value.
%NRBQUOTE()	Masks all tokens in the macro at execution.	Warnings are produced when trying to mask macros or macro variables, as the function is resolving them.
%SUPERQ()	Masks all tokens no matter if they can be treated as macro variables or macros.	Only a macro variable can be an argument to this function. It needs to be explicitly unquoted to resolve.
%UNQUOTE()	Resolves all quoted tokens and masks macro triggers that cannot be treated as macros or macro variables.	Needs to be used cautiously as unmasking unmatched tokens may lead to wrong results – unbalanced quote or open parenthesis.
%QSCAN()	Can be used on macro variables with tokens.	Cannot be directly used to create variables.
%QSUBSTR()	Can be used on macro variables with tokens.	Cannot be directly used to create variables.

Method	Advantages	Disadvantages/Limitations
Data step functions		
SYMGET()	Loads a macro variable into a dataset variable.	It usually requires assigning the length of the dataset variable, in which the value of a macro variable is loaded. This is to avoid potential truncation when the dataset variable is processed.
CALL SYMPUTX()	Allows easy creation of a number of macro variables. Allows both global and local variables to be created.	Macro variables can be created from variables values in a data set. It is convenient in some cases but it is not easily visible what macro variables are created and what are their values.
RESOLVE()	Best function to resolve macro variables as it masks all tokens and resolves macro variables and macros.	Cannot be directly used on a macro variable – requires SYMGET() to be used first.

CONCLUSION

Working with macro variables can be difficult, but it can be simplified by the use of the newest SAS programming framework. It is advisable to use the newest macro language functions %NRSTR(), %NRBQUOTE() and SUPERQ(). It is important to understand in detail what the differences between those three functions are. Don't forget that macro variables can also be read into a data step and be processed in the same way as any other dataset variables. Use of a data step is recommended when macro variables require extensive processing, as it decreases complexity of programming and improves code transparency.

REFERENCES

Brian Patterson, Mylene Remigio, "Don't %QUOTE() Me on This: A Practical Guide to Macro Quoting Functions", SAS Global Forum 2007,
http://www2.sas.com/proceedings/forum2007/152-2007.pdf

Shan Lee,"Quoting Macro Variable References", PhUSE 2007
http://www.lexjansen.com/phuse/2007/tu/TU04.pdf

RTF Tags
https://www.microsoft.com/en-gb/download/details.aspx?id=10725

ACKNOWLEDGMENTS

The author would like to express appreciation to Carl Herremans and Frederic Coppin for his input during the preparation of this paper.

RECOMMENDED READING

Chang Y. Chung, John King, 'IS THIS MACRO PARAMETER BLANK?', SAS Global Forum 2009
http://changchung.com/download/022-2009.pdf

Beilei Xu, Lei Zhang, 'Take and In-Depth Look at the %EVAL Function' NESUG 17.
http://www.lexjansen.com/nesug/nesug04/pm/pm26.pdf

CONTACT INFORMATION

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

Dawid Militowski
HTA Statistical Programmer
MSD
Hertford Rd, Hoddesdon, EN11 9BU, United Kingdom
Work Phone: +44 1992 452 730
Email: dawid.militowski@merck.com
Web: http://www.msd.com