PhUSE 2014

Paper CC02

Dequote me on that: Using the dequote function to add some friendliness to SAS macros

John Hendrickx, Danone Nutricia Research, Utrecht, The Netherlands

ABSTRACT

SAS macros need to be user-friendly (certainly if you're going to use the macro yourself). This paper shows how to use the dequote function to make macros easier to use. Macro calls can use single or double quotes to mask text that would otherwise require macro quoting. The macro itself is written to use dequote to remove quotation marks and unmask special characters. The macro becomes easier to explain, easier to use, easier to read.

INTRODUCTION

The old "alt.sysadmin.recovery" newsgroup faq[1] has a wealth of information – great tips for example, on the usage of duct tape and latex gloves if anything goes wrong in a "luser education campaign". Yes, SAS macro developers can learn a lot from the FAQ – unless of course, you have to eat your own dog food: unless you have to use the macros yourself! In that case it's time for some tender, loving care!

And it's hard to be more friendly than by shielding everyone from SAS macro quoting! Don't take my word on it, here's a second opinion: "*One of the more perplexing things for any SAS® programmer to learn is the effective use of macro quoting functions*" (Dunn, 2009). Of course, for macro programmers, there's no escaping a thorough knowledge of macro quoting, but for macro users, thing can be made completely intuitive! Well, at least more intuitive than something like %nrbquote().

Let's take a brief look at macro quoting, then how to use regular quotation marks – like you learned in grade 6 – but in macros this time. After that, we'll move on to getting a longer list of options into a macro!

MACRO QUOTING

Macro parameters can contain special characters – commas, percentage signs, ampersands, the semicolon, to name a few[2]. Only we don't always want them to be special – sometimes a semicolon needs to be just a plain text semicolon, not the termination of a SAS statement. Hence macro quoting, a set of macro functions that mask these special characters so that they are treated as plain text. The macro quoting functions are %str, %nrstr, %quote, %nrquote, %bquote, %nrbquote, %superq.

Why so many?

Let's start with %quote and %bquote. The difference between the two is that %bquote will mask an unbalanced single or double quotation mark whereas %quote will not. If you use the %quote function, you must precede the unbalanced quotation mark by '%'.

```
%put %bquote(John's paper);
%put %quote(John%'s paper);
```

So %quote and %nrquote can be seen as legacy functions, it will always be more convenient to use %bquote and %nrbquote respectively. That leaves %str, %nrstr, %bquote, %nrbquote, %superq.

---

[1] http://www.faqs.org/faqs/sysadmin-recovery/

[2] The complete list of special characters and mnemonic operators that can require macro quoting is:
`& % ' " ( ) + – * / < > = ¬ ^ ~ ; , # blank`
`AND OR NOT EQ NE LE LT GE GT IN`
(SAS(R) 9.4 Macro Language: Reference, Second Edition)
I will refer to these as 'special characters', noting that this includes the mnemonic operators that consist of two or more characters

A second distinction is compile time versus execution time. The %str and %nrstr functions mask special characters when the macro is compiled, the other macro functions work when the macro executes. Outside of a macro, the %str and %bquote functions can often be interchanged, as can %nrstr and %nrbquote. But inside a macro, %str will work whereas %nrbquoute will spill an error message all over your screen. Take the following example from Whitlock (2003).

```
%macro semicolon;
    %local x;
    %let x = %str(;);
    %put abc %upcase(&x) def;
%mend semicolon;
```

This macro will run just fine because the semicolon in macro variable `&x` is masked at compile time through the use of the %str function. The semicolon is enclosed in special unprintable characters (see Dunn, 2007) but at execution time these are removed and the log reports "`Macro variable X resolves to ;`" (providing the symbolgen system option is used).

The following version on the other hand, will not work:

```
%macro semicolon;
    %local x;
    %let x = %bquote(;);
    %put abc %upcase(&x) def;
%mend semicolon;
```

The %bquote function does mask the semicolon but the %upcase function causes `&x` to become unmasked. An error ensues at compile time, since the %put statement resolves to:

```
%put abc ; def;
```

This is of course incorrect SAS syntax and it's no surprise that a big red error message results. Okay, just a red error message, that's annoying enough.

So you should use %str or %nrstr inside a macro and you can use %bquote or %nrbquote elsewhere. But wait a minute. There are %quote and %bquote, %nrquote and %nrbquote. The version with the 'b' can deal with unbalanced single or double quotation marks but the %quote and %nrquote function require that unbalanced quotation marks be preceded by a percentage sign. Aren't there %bstr and %nrbstr functions that can do the same, but at compile time rather than at execution time? Will a large percentage of my time be used dealing with unbalanced quotation marks in %str or %nrstr?

The answer is, "afraid so", unbalanced quotation marks in %str or %nrstr need to be preceded by a '%'. The "%' can also be used to mask unbalanced parentheses. A percentage sign itself must be represented by a double percentage sign.

```
%put %str(An unbalanced parenthesis %)); /*An unbalanced parenthesis )*/
%put %str(Give me 20%%); /*Give me 20%*/
```

So %str and %nrstr can be used inside a macro to mask special characters until execution time whereas other macro functions can't. Other than this, there's no big difference between %str and %bquote except that the latter can deal with unbalanced quotation marks. (Not so for %nrstr versus %nrbquote, see below).

Next point, the distinction between the %nr and the non-%nr versions of macro functions. Basically, the %nr versions include "&" and "%" in the characters they will mask whereas the non-%nr versions do not mask these characters. For example:

```
%let a=me;
%put %str(You and &a); /*You and me*/
%put %nrstr(You and &a);/*You and &a*/
```

If the %str function is used, then &a is expanded. If macro variable &a is not defined then a warning ensues. If the %nrstr function is used instead, then &a is masked.

You might expect %bquote to produce the same result as %str (provided this occurs outside a macro) and for %nrbquote to be the same as %nrstr. Unfortunately, it's a bit more complicated. Yes, the %bquote function produces the same result as %str. %nrstr on the other hand will never expand macro variables whereas %nrbquote will do so.

```
%put %bquote(You and &a);  /*You and me*/
%put %nrbquote(You and &a);  /*You and me*/
```

The %nrbquote function tries to expand any macro variables. In most situations, %bquote and %nrbquote will produce the same results.

The %nrstr function on the other hand, is very different from the %str function and can be used for the very useful purpose of delayed evaluation of a macro variable, either in open code or within a macro. Using the %nrstr function will place macro variable between invisible quotation marks and it will be treated as plain text. No errors will be produced if a macro variable masked using %nrstr does not exist (yet).

```
%let a=me;
%let Waitforit=%nrstr(You and &a);
%put &waitforit; /*You and &a*/
```

What's very useful is that a macro variable masked by %nrstr can be unmasked and updated. To do this, the macro variable needs to be unquoted, and rather astonishingly, a completely appropriate and intuitive name for this function: %unquote. The following example shows how %unquote can be used to unquote a macro variable, which will then evaluate to its current value, not to the value when %nrstr was used.

```
%put %unquote(&waitforit); /*You and me*/
%let a=I;
%put %unquote(&waitforit); /*You and I*/
```

It's not just the %unquote function though that will unquote a macro variable that has been masked using %nrstr. Most macro functions will do this as well. For example, the %scan function:

```
%put %scan(&waitforit,1,\); /*You and I*/
```

If the macro variable is not to be unmasked yet at this point, the %qscan function can be used instead. The %qscan function works in the same way as %scan but applies macro quoting to the results (using the same invisible characters as %nrstr). A quoted macro variable will remain a quoted macro variable.

```
%put %qscan(&waitforit,1,\); /*You and &a*/
```

One last macro function to discuss is the %superq function. This expands a macro string exactly as it is, any special characters are not interpreted or expanded. Example 1.A from Patterson and Remigio (2007) can be used to illustrate how useful the %superq function can be. Their sample macro is as follows:

```
%MACRO Ex_1(State);
    %IF (&State EQ %STR(NE) OR &State EQ %STR(OR) OR &State EQ MO OR
        &State EQ KS OR &State EQ WY OR &State EQ ID) %THEN
        %PUT The Oregon Trail ran through &State;
    %ELSE %PUT The Oregon Trail did not run through &State;
%MEND Ex_1;
```

To use this macro, the macro argument needs macro quoting:

```
%EX_1(%QUOTE(OR));
```

In this modified version, applying the %superq function to the positional argument &state makes it unnecessary to use macro quoting when calling the macro. Unlike other macro functions, %superq requires that the macro be specified without the ampersand sign, i.e. `%superq(State)`, not `%superq(&State)`. The macro variable is expanded but any special characters are treated as plain text.

```
%MACRO Ex_1a(State);
    %IF (%superq(State) EQ %STR(NE) OR %superq(State) EQ %STR(OR) OR %superq(State)
EQ MO OR
        %superq(State) EQ KS OR %superq(State) EQ WY OR %superq(State) EQ ID) %THEN
        %PUT The Oregon Trail ran through %superq(State);
    %ELSE %PUT The Oregon Trail did not run through %superq(State);
%MEND;

%EX_1a(OR);
```

The attentive reader cannot but allow that I have convincingly illustrated how confusing macro quoting can be, as have the authors referenced above and others I have not cited. Using the %superq function in a macro can make the macro a little bit more user-friendly by shielding users from the dark art of macro quoting. But %superq can't make all forms of macro quoting unnecessary for users of your macros. The next section shows how the dequote function can also be used to make life a little easier (and SAS programs a little more readable).

DEQUOTING

My spelling checker is of course redlining "dequoting", there's no such word. But unquote as already taken as a SAS (macro) function, so SAS came up with "dequote" as the name of this datastep function. What the dequote function does is simply to remove quotation marks.

```
data _null_;
    MyQuotedString='"Can I quote you on that?"';
    MyDeQuotedString=dequote(MyQuotedString);
    put (MyQuotedString MyDeQuotedString) (=/) ;
run;

MyQuotedString="Can I quote you on that?"
MyDeQuotedString=Can I quote you on that?
```

The dequote function isn't a macro quoting function but it can be used as such by it in a %sysfunc or %qsysfunc function. Dequote has many nice properties. For example, if a string isn't quoted then the dequote function does nothing.

```
%put %qsysfunc(dequote(Hi there)); /*Hi there*/
```

As in title or footnote statements, single quotes will prevent macro variables or macros from being expanded. Embedded quotes are allowed, provided they've been doubled.

```
%put %qsysfunc(dequote('Hi, there''s milk (2.5% fat) &cookies (forgot the
space!)'));
/*Hi, there's milk (2.5% fat) &cookies (forgot the space!)*/
```

If double quotes are used then macros or macro variables will be expanded.

```
%macro fat;
    percent cream
%mend;
%let cookies=and chocolate chip cookies;
%put %qsysfunc(dequote("Hi, there's milk (2.5 %fat) &cookies (forgot the
space!)"));
/*Hi, there's milk (2.5 percent cream) and chocolate chip cookies (forgot the
space!)*/
```

The dequote function (wrapped in `%qsysfunc` for use in macros) is flexible and leverages the user's knowledge of the ins and outs of quotation marks in titles and footnotes. It's an excellent tool for making macros a little easier to use.

USER-FRIENDLINESS

We've seen that macro quoting is used to let SAS know that a macro argument should be treated as plain text. Used in a macro call, this means that the comma, the semi-colon, the ampersand should be treated as part of a specification and that SAS should not take the actions it would usually take. The macro functions %str, %nrstr, %nrbquote can be used in these situations.

But another option is to let the user use regular, single or double quotation marks instead. The quotation marks will mask the special characters when the macro is executed. Within the macro, the dequote function is used to, well, dequote the macro parameter enclosed in quotation marks. The macro works the same, but it becomes easier to explain, easier to use, and easier to read.

To illustrate this, take for example the %Ex_3 macro from Patterson and Remigio (2007):

```
%MACRO Ex_3(Food);
    %IF (&Food EQ %NRSTR(Rice & Beans)) %THEN
        %PUT Try the House Special - &Food;
    %ELSE %PUT &Food - a tasty snack!;
%MEND Ex_3;
```

```
%Ex_3(Shrimp Cocktail);
%Ex_3(%QUOTE("Fried" Green Tomatoes));
%Ex_3(%BQUOTE(Mama's Biscuits));
%Ex_3(%NRQUOTE(Rice & Beans));
```

Using macro quoting, this example runs quite well. But it can be made a lot more user-friendly (albeit perhaps less programmer-friendly), by making use of the dequote function:

```
%MACRO Ex_3a(Food);
    %IF (%qsysfunc(dequote(&Food)) EQ %NRSTR(Rice & Beans)) %THEN
        %PUT Try the House Special - %qsysfunc(dequote(&Food));
    %ELSE %PUT %qsysfunc(dequote(&Food)) - a tasty snack!;
%MEND;

%Ex_3a(Shrimp Cocktail); /*Shrimp Cocktail - a tasty snack!*/
%Ex_3a('"Fried" Green Tomatoes'); /*"Fried" Green Tomatoes - a tasty snack!*/
%Ex_3a("Mama's Biscuits"); /*Mama's Biscuits - a tasty snack!*/
%Ex_3a('Mama''s Biscuits'); /*Mama's Biscuits - a tasty snack!*/
%Ex_3a('Rice & Beans'); /*Try the House Special - Rice & Beans*/
```

To use the macro properly, all that users have to understand is how use quotation marks in SAS. If they can do titles and footnotes correctly, they can use the macro. And even a non-SAS user could grasp what's happening here, in the sense that a string is being passed to a user-written program. %Nrbquote on the other hand, requires a good deal of explanation, as we have seen.

The %Ex_3a macro uses %qsysfunc and therefore special characters are masked after dequoting. If %sysfunc is used then the third and fourth examples, `"Mama's Biscuits"`, will produce an error because it contains unbalanced quotation marks. Because %qsysfunc is used, special characters can be included to some extent without using quotation marks:

```
%Ex_3a(Rice & Beans);
%Ex_3a(Rice and Beans);
```

By using the dequote function, the macro is much simpler for users to use. Quotation marks are optional if the macro argument doesn't contain commas or unmatched quotation marks. On the other hand, instructions can be greatly simplified by telling users to always use quotation marks, using the same rules as apply to titles and footnotes. The results will be the same.

EXPANSION AT EXECUTION TIME

The %nrstr function can be used to mask a macro variable until the point where the programmer wants it to be expanded. This can be done as well by allowing single quotes in the macro specification and using the dequote function in the macro.

The following example is based on a macro for confidence interval plots with a time variable by group variable. A "stagger" value is added to the x-axis value of the treatment groups so their CI plot doesn't overlap. The problem here is the Y-axis. Proc gplot doesn't always select good axis intervals automatically. That's not a problem for a one-off graph but if a series of graphs needs to be made, it's much nicer if an appropriate "order=*a* to *b* by *c*" specification on the axis statement can be calculated automatically.

To do this, an axis statement for the Y-axis can be specified as a macro parameter, using single quotes.

```
%ciplot(data=lab,x=avisitn,y=logalat,by=trtan,vunit=0.05,
autovaxis='Axis1 width=1 order=&ordercmd minor=none label=none
value=(height=2pct);',
plotopts="vaxis=axis1 haxis=axis2 legend=legend1 noframe");
```

The order specification is represented by macro variable &ordercmd. Because single quotes are used, &ordercmd is not expanded when the syntax is run.

Internally, the %ciplot macro calculates appropriate values for &ordercmd based on the range of the plot data. The axis1 statement is then created using

```
%sysfunc(dequote(&autovaxis))
```

In this case, %sysfunc must be used, not %qsysfunc. The %sysfunc causes the &autovaxis macro parameter to be unmasked, similar to using the %unquote function. The &ordercmd macro variable has appropriate values at this point in the macro and an appropriate axis statement is generated.

This could also be accomplished using %nrstr. But using quotation marks makes it unnecessary for the user to fully grasp the intricacies of macro quoting. An added benefit is readability of the code.

Quotation marks are also used for the "plotopts" parameter of this macro. This is not strictly necessary in this example, the equal signs do not cause any potential problems. On the other hand, the quotation marks do not affect the proper functioning of the macro, if they're not necessary then they are simply stripped. They group the specification in a way that (to me) makes the specification easier to read. And because the plotopts parameter is dequoted with the macro, any special characters it might contain will be effectively masked.

PARSING LISTS

A more advanced application of the dequote function involves parsing a list of quoted and unquoted strings, for example:

```
%let myString="label 1" a b "label 2" c d e "label 3" x y z;
```

In this case, there are pairs of quoted labels, followed by an unquoted specification. A useful property of the dequote function is that if a string consists of multiple quoted and unquoted substrings, the dequote function will be applied to the first quoted string and the rest of the string will be ignored. In this example, the dequote function will extract just the first label "`label 1`" (without the quotes)

```
%put %qsysfunc(dequote(&myString)); /*label 1*/
```

The first label can be extracted and then deleted from `&myString` as follows:

```
%let labelstr=%qsysfunc(dequote(&myString));
%let instr=%qsubstr(&myString,%length(&labelstr)+3);
```

The next step is to extract the specification. First, any leading spaces on `&instr` are deleted using the `left` function. Then things get hairy[3]. A regular expression is used to extract the specification, which works great but doesn't mean high readability if you're not familiar with the technique.

```
%let instr=%qsysfunc(left(&instr));
%let specstr=%qsysfunc(prxchange(s/^([\w\s]+)\W?.*$/$1/,1,&instr));
```

Let's see, if we focus on the regex, it contains "`s/^([\w\s]+)\W?.*$/$1/`".

- In `prxchange`, "`s/`" initiates the substitution expression, the first slash "`/`" ends the "find" expression, the second "`/`" ends the "replace" expression.
- A match in the "find" expression, when placed in parentheses, is represented by "`$1`" in the replace expression.
- A caret "`^`" in the "find" expression means start of the string, "`$`" means end of string.
- The "`\w`" will match letters, digits and an underscore, the "`\s`" will match spaces (and tabs, but that won't happen here).
- "`[\w\s]`" will match either letters, digits, underscore and/or spaces whereas "`[\w\s]+`" signifies one or more matches.
- "`\W`" will match anything but letters, digits or an underscore and this is used to match a quotation mark in `&instr`.
- The question mark "`?`" indicates that this match is optional, for use in a loop when there are no more quotation marks or other "`\W`" characters to match.
- In a regex, a period matches anything and an asterisk indicates zero or more matches.

So, I'm asking to match a series of one or more spaces, letters, digits, starting at the start of the string, up to the first non-letter or digit character, followed by whatever, and to replace the string with the match found (of spaces, letters, digits). Just less verbose.

As always in SAS, there are other ways of doing this. But regex's are my hammer and `&instr` sure looks like a nail to me!

Moving on, the final step is to remove the specification string that has just been extracted from `&instr`.

[3] http://www.retrologic.com/jargon/H/hairy.html, meaning 1 or 2 ("Annoyingly complicated" or "Incomprehensible").

```
%let instr=%qsubstr(&instr,%length(&specstr)+1);
```

At this point, the first label and the first specification have been extracted. Now, let's program this properly in a loop:

```
%macro parseQuote(instr);
    %local i;
    %let i=1;
    %do %while (%length(%trim(&instr)) gt 0);
        %let labelstr=%qsysfunc(dequote(&instr));
        %let instr=%qsubstr(&instr,%length(&labelstr)+3);
        %let instr=%qsysfunc(left(&instr));
        %let specstr=%qsysfunc(prxchange(s/^([\w\s]+)\W?.*$/$1/,1,&instr));
        %let instr=%qsubstr(&instr,%length(&specstr));
        %let instr=%qsysfunc(left(&instr));
        %put i=&i;
        %put labelstr=&labelstr;
        %put specstr=&specstr;
        %put instr=&instr;

        %let i=%eval(&i+1);
        %if &i gt 20 %then %return;
    %end;
%mend;
%parseQuote(&myString);
```

A counter was included in the %do %while loop to prevent an infinite loop from occurring. Hopefully unnecessary but you never know. When run, the macro writes the following to the SAS log:

```
i=1
labelstr=label 1
specstr=a b
instr="label 2" c d e "label 3" x y z
i=2
labelstr=label 2
specstr=c d e
instr="label 3" x y z
i=3
labelstr=label 3
specstr=x y z
instr=
```

So the macro worked as intended, producing labels without quotation marks and corresponding specifications.

This approach could be used to specify a set of test specifications and the accompanying headers for a series of tests for data with multiple treatment groups. Or to specify multiple test, columns with multiple p-values. The number of label pairs is only limited by an (entirely optional) check on the number of iterations of the loop. The specification can be short or long, as long as it consists only of letters, digits or an underscore (\w for short).

CONCLUSION

The dequote function can help make macros easier to use by allowing the use of single or double quotes for macro parameters rather than macro quoting functions. Using quotation marks in macro specifications follows the same rules as titles, footnotes, that users are already familiar with. Special characters are effectively masked, but knowledge or usage of macro quoting is not necessary. Users with just a basic knowledge of SAS will more readily understand when a macro variable will be expanded immediately and when expansion will be delayed. An added benefit of using quotation marks is that programs become clearer and easier to read.

REFERENCES

Dunn, Toby. 2008. "Macro Quoting". *SESUG 2008 Proceedings, Paper #CS-049.*

Patterson, Brian, Remigio, Mylene. 2007. "Don't %QUOTE() Me on This: A Practical Guide to Macro Quoting Functions". *SAS Global Forum 2007, Paper 152-2007.*

SAS(R) 9.4 Macro Language: Reference, Second Edition

Whitlock, Ian. 2003. "A Serious Look Macro Quoting". *SUGI 28, paper 11-28*.

CONTACT INFORMATION

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

John Hendrickx
Danone Nutricia Research
Uppsalalaan 12
Utrecht, The Netherlands NL-3584CT
John.Hendrickx@Danone.com