Paper 1130-2017

Protecting Your Programs from Unwanted Text Using Macro Quoting Functions

Michelle Buchecker, ThotWave Technologies LLC

ABSTRACT

Face it, your data can occasionally have characters that wreak havoc on your macro code. Characters such as the ampersand in AT&T, or the apostrophe in McDonald's for example. This paper is designed for programmers who know most of the in's and out's of SAS Macro code already. Now let's take your macro skills a step farther by adding to your skillset. Specifically, %BQUOTE, %STR, %NRSTR, %SUPERQ what is up with all these quoting functions? When do you one over the other? And why would you need %UNQUOTE? The macro language is full of subtleties and nuances and the quoting functions represent the epitome of all of this. This paper will show you in which instances you would use the different quoting functions. Specifically, what is the difference between the compile time versus execution time functions. In addition to looking at the traditional quoting functions, you will learn how to use %QSCAN and %QSYSFUNC among other functions that apply the regular function and quotes the result.

INTRODUCTION

SAS programmers know that if you want to treat characters you type in to the program as constant text, then you need to put quote marks around that text. However, when it comes to macro programming, this technique doesn't work. Why? Because the macro facility is a text processing facility and will assume those quotes are just some other constant character, like an "a" or a "q".

Instead, you will need to use a macro quoting function that protects the text from being interpreted in an incorrect way.

The macro facility has several quoting functions. This paper examines those most commonly used.

PREVENTING TYPED IN TEXT TO HAVE A "SAS MEANING"

The first type of issue we are going to address is code that you type in yourself on your keyboard. Later we'll take a look at SAS code that your macro code generates.

PROTECTING MACRO TRIGGERS: & AND %

There are times that you want the & and % characters that you type in to mean an ampersand or percent sign and not a macro trigger. For example, you are working with a company named AT&T.

To tell SAS "hey this & or % shouldn't be treated as a macro trigger" you need to use the macro quoting function %NRSTR.

For example, if you coded:

```
%let t=TEMPT;
%put Company is AT&T;
```

The result would be:

```
Company is ATTEMPT
```

To protect that ampersand, you should instead code:

```
%let t=TEMPT;
%put Company is %nrstr(AT&T);
```

The result would be:

```
Company is AT&T
```

PROTECTING OTHER TEXT LIKE SEMICOLONS

There is a macro quoting function, **%STR**, dedicated to protect, ie: remove the normal meaning from, these characters:

+ - * / < > = ¬ ^ ~ ; , blank AND OR NOT EQ NE LE LT GE GT

For example, if you coded:

```
%let state=OR;
%if &state=FL %then . . .
```

SAS would resolve this to:

```
%if OR=FL %then . . .
```

And the OR on the %IF statement would be treated like the logical operator OR (cousin to AND) and not the state abbreviation for Oregon.

A corrected approach would be:

```
%let state=%str(OR);
%if &state=FL %then . . .
```

In this case the text OR is considered to be protected, or quoted.

Could you have used %NRSTR here? Yes, because %NRSTR protects the same characters that %STR protects, as well as the & and %. But consider if you had a text string where you needed some of the text protected, but still wanted macro triggers to execute. In this case, you need to exclusively use %STR. Because of this, I'm in the habit of only using %NRSTR when I need to protect & and %.

For example, if you coded:

```
%let distance=miles;
%let title=%nrstr(50 &distance driven;);
```

The macro variable &distance would not get resolved and the title would be:

```
50 &distance driven;
```

PROTECTING QUOTES

When it comes to protecting actual quotation marks things get even trickier. To help minimize the confusion, I'm going to call the physical quotation marks 'tick marks'. So a tick mark is the key on your keyboard.

You may need to protect tick marks when you have an apostrophe in your code. For example, you are working with a company named McDonald's.

If you say:

```
%let name=McDonald's;
```

Then the macro compiler interprets that tick mark as being the start of a quoted string and all subsequent characters until the next tick mark are part of that string. At this point you have introduced an unbalanced quotation mark.

The %STR function can work here but it requires a bit more work. You need to add an extra percent sign in front of the tick mark to tell the macro facility “hey, this tick mark doesn’t have a match, so don’t look for one”.

Specifically, the code would be:

```
%let name=%str(McDonald%'s);
```

WANTING TYPED IN TEXT TO HAVE SAS MEANING BUT NOT WANTING THE RESULT TO HAVE SAS MEANING

In the previous examples all of the protected text is what we as programmers would type in from the keyboard to create macro variables. Many times though, the values of the macro variables are determined by DATA step variables. So in this case, you are using the value of the data to create the value of the macro variable. In other words, you aren’t typing the value from your keyboard, but the data is supplying it.

In this case we are going to have to use a different set of macro quoting functions.

Let’s take this example:

```
Call symput('company' !! _n_, name);
```

Where **name** is a variable in the data set representing company names of AT&T and McDonald’s. The DATA step will run fine and creates macro variables of **company1=AT&T** and **company2=McDonald’s**. So no quoting functions are needed here to **create** the macro variables.

The problem comes when you wish to use the macro variables. If you then coded:

```
%let t=TEMPT;
%put &company1;
```

This would resolve to **ATTEMPT**. We need &company1 to resolve. So we can’t use %NRSTR because that says treat macro triggers as text and not their normal meaning. So we need a macro quoting function that says “let the macro variable resolve, but protect the results.”

There are two macro quoting function that fall into this category: %SUPERQ and %BQUOTE. There are some subtle but important differences between these two functions.

1. SUPERQ does not use an ampersand in front of the macro variable name
2. SUPERQ only allows a single macro variable as the argument, no additional text. BQUOTE allows additional text. Example: %bquote(&company1 isn’t my carrier) (NOTE BQUOTE protects apostrophes and other text that are not &’s or %’s).
3. BQUOTE allows any resolved value that has a macro trigger of & or % to further resolve. SUPERQ just resolves the macro variable once with no further additional resolution.

The first two differences list are more syntactical in nature. The last one is a processing difference.

So in our above problem let’s see how BQUOTE and SUPERQ will differ.

Using %superq:

```
%let t=TEMPT;
%put Company is %superq(company1);
```

Results in

```
Company is AT&T
```

Using %bquote:

```
%let t=TEMPT;
%put Company is %bquote(&company1);
```

Results in

```
Company is ATTEMPT
```

This happened since BQUOTE permits any macro triggers of a resolved macro variable to resolve further and then protects the results. So not a good use here.

When can BQUOTE be of use then? Well, take this example:

```
%let t=TEMPT;
%put %bquote(&company2 Mexican food spinoff was an &company1);
```

Results in

```
McDonald's Mexican food spinoff was an ATTEMPT
```

So here you can see that the apostrophe is protected, additional text can be inside the parenthesis, and that a macro variable is allowed further resolution.

PASSING VALUES OF MACRO VARIABLES TO A RDBMS

In many cases when you are using SQL pass-through, the code you are passing to the Relational Database Management System (RDBMS) requires the text to be in **single** quotes. Since SAS treats all text in single quotes as literal text, it won't resolve the macro variable as only macro variables in double quotes, or no quotes at all, get resolved.

So you need a way to resolve the macro variable in SAS and pass the results in with single quotes. This is a good use of the %UNQUOTE function.

Earlier we looked at a number of quoting functions that protecting the text and removed the normal meaning from the text and protected the text and treated it as literal text. The %UNQUOTE function unprotects literal text and restores it's normal meaning.

Let's say you have a macro variable named **flag** whose value is **yes** and you need to pass that in to a WHERE clause in SQL pass-through code filtering a DBMS table variable named **status**. The code would look like:

```
where status=%unquote(%str(%')&status%str(%'))
```

So we use the %STR function discussed earlier to protect the single tick marks and treat them as constant text for SAS. Then the macro variable resolves without being inside of quotation marks. The %UNQUOTE function then unprotects the tick marks restoring them back to their normal meaning of being quotation marks.

USING PROTECTION FUNCTIONS

By default, macro functions return an unquoted result, even if the argument was masked. The Qxxx functions will protect the returned value while using the desired function.

For example, assume we have the macro variable **company1=AT&T International** and **company2=McDonald's Incorporated**. I now want to use a SCAN function to pull off the first word, but then immediately protect the results to protect that & and apostrophe:

```
%let name1=%qscan(&company1, 1, %str( ));
%let name2=%qscan(&company2, 1, %str( ));
```

This code performs the scan extracting the first word where words are separated by spaces. Notice the use of the %STR function to protect that space, i.e.: remove its normal meaning of a space in the code which is simply ignored, and treat it as a literal space. The resulting values in name1 and name2 are quoted/protected and so can be safely used in other parts of code.

Other protection functions include %QSUBSTR, %QUPCASE, and %QSYSFUNC.

SUMMARY

Table 1: :

Need	Function(s)
Protecting macro triggers: & and %	%NRSTR
Protecting other text like semicolons	%STR
Protecting apostrophe	%STR with % in front of apostrophe
Protecting results of resolved macro variables	%SUPERQ, %BQUOTE
Protecting results when using a macro function	%Q*macro-function-name*
Restoring meaning to a macro quoted value	%UNQUOTE

Table 1: Quoting Functions

CONCLUSION

Understanding macro quoting functions is important to determining how to protect both your typed in code, or generated code and treat it as literal text without putting the physical tick marks around it. If you don't utilize these functions, your code may produce undesired results, including misinterpreting data as code.

REFERENCES

Mock, Jerry. 1985, "Macro Quoting Functions", *Proceedings of the 10th Annual SAS Users Group International Conference*, Reno, NV, SAS Institute, Inc., Available at http://www.sascommunity.org/sugi/SUGI85/Sugi-10-203%20Mock.pdf

CONTACT INFORMATION

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

Michelle Buchecker
ThotWave Technologies, LLC.
mbuchecker@thotwave.com