Paper 152-2007

Don't %QUOTE() Me on This: A Practical Guide to Macro Quoting Functions

Brian Patterson, The College Board, New York, NY
Mylene Remigio, The College Board, New York, NY

ABSTRACT

For programmers new to the SAS® Macro Facility, understanding the set of Macro quoting functions and their appropriate uses can be a difficult task. This paper describes a decision-based selection methodology for choosing the correct Macro quoting function, based on which character(s) (tokens) are to be masked and at what stage the masked token(s) are to be unmasked. It is also meant to be a refresher for more experienced Macro programmers, who may use the methodology graphic as a handy reference.

INTRODUCTION

The SAS Macro Facility is a text processing resource that allows you to automate and customize the generation of SAS code. There are certain "tokens" or sets of characters that have special meanings to the Macro processor, including (but not limited to) Macro triggers, logical and arithmetic operators, and semicolons. As programmers become more adept in their use of the Macro Facility, most find it necessary to mask or "quote" the meaning of these special characters. This need may arise from experiencing syntax errors, or from subtler logical errors that do not result in an error message, but nonetheless lead to unexpected results.

We see this paper as a practical approach to quoting functions; as such, it will not attempt to cover the technical depth presented within previously published papers. These quoting functions are numerous and may be interchangeable in certain cases, but it is our goal to develop a practical approach to deciding which quoting function is appropriate for each specific case.

METHODOLOGY

In general, we assume that it is ideal to use the least robust Macro quoting function that will accomplish your goal (i.e. the function that masks the fewest tokens for the least amount of time). While more than one quoting function can be used to perform the same task, masking any more characters than is necessary may result in unexpected resolution of Macro variables.

DECISIONS TO BE MADE

QUESTION 1: WHICH TOKENS MUST BE MASKED?

In general the tokens to be masked can be broken down into three categories:

- **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 Macro quoting functions operate similarly on the three **Type I** groups, but may operate differently for that large group than they do for unmatched characters and Macro triggers

Certain tokens generally appear in pairs, most notably the pair of single quotes (''), the pair of double quotes ("") and the pair of parentheses [()]. During compilation, SAS expects those three pairs of tokens to appear in pairs and when they do not, a variety of errors may ensue, none of them particularly pleasant. For example, consider setting up a Macro variable that contains a title to be printed with SAS output. If the title contains a single quote (e.g., an apostrophe), errors occur not in the title's assignment to the Macro variable, but after the Macro variable resolves and the compiler is looking for the matching single quote. Two examples (Ex. 2.A and 2.B) are presented as solutions to this problem.

QUESTION 2: WHEN MUST THESE CHARACTERS (OR TOKENS) BE UN-MASKED?

Depending upon the quoting functions used and other code present, the tokens will remain masked until one of the following occurs:

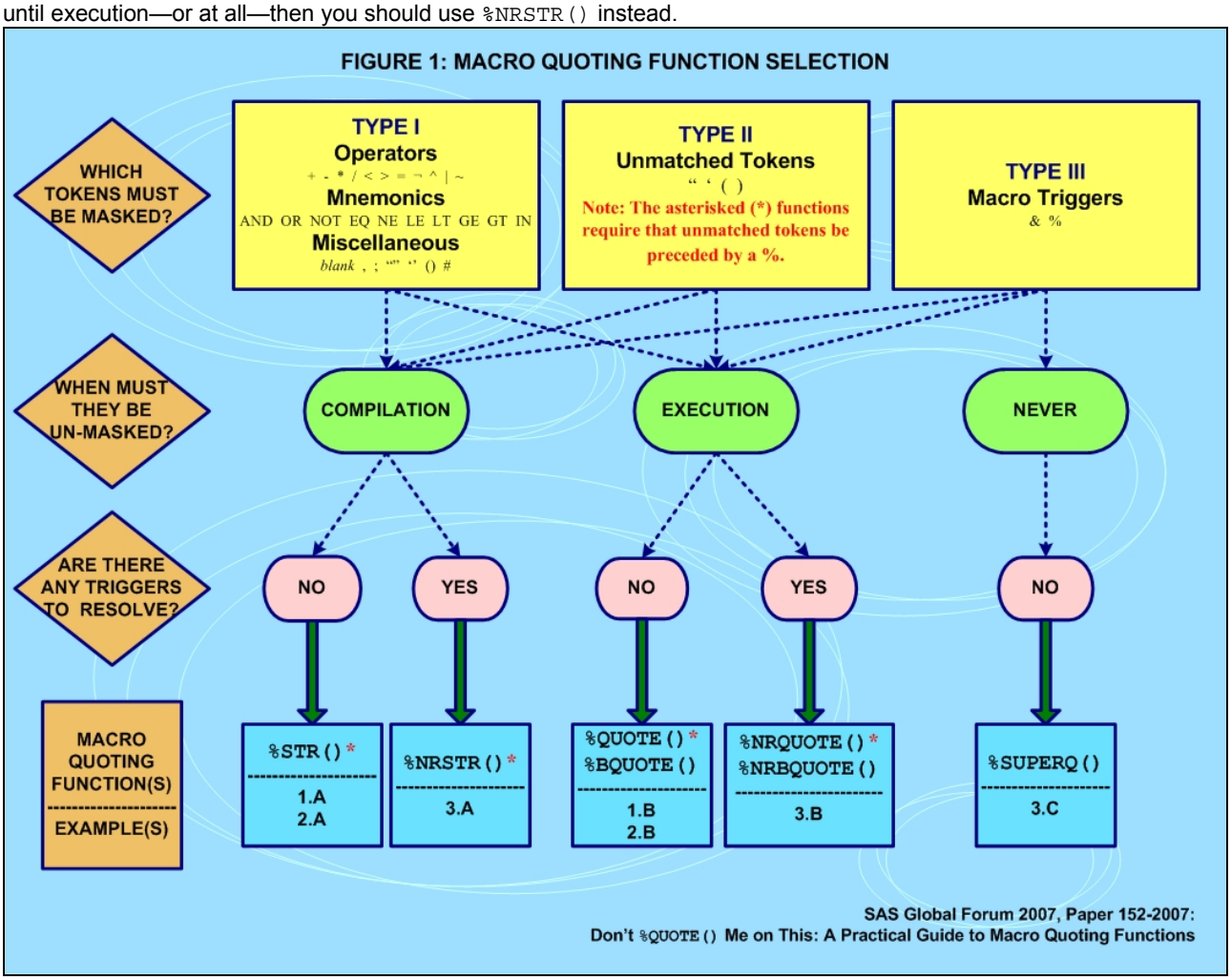
1. The Macro is compiled
2. The Macro is executed
3. The `%UNQUOTE()` function is used

In many cases, the time at which the characters are being unmasked is unimportant; consider this distinction for the sake of only using the weakest quoting function that would accomplish your goal. Hard-coded (static) expressions that contain special tokens generally appear in Macro definitions and therefore should be masked through compilation. On the other hand, dynamic (variable) expressions may or may not contain special tokens and should be masked through execution; parameters passed within a Macro call are good examples of this. The exception to these guidelines applies when you invoke the `%UNQUOTE()` function for a Macro expression that contains masked characters.

If, in the course of coding, you would like to know which tokens are masked within a certain Macro variable, you may simply issue the "`%PUT _USER_;`" command. Depending upon your operating system, the masked or quoted tokens will appear as non-printable characters. As an example, quoted characters often appear as "□" on the Microsoft Windows XP Professional platform.

QUESTION 3: ARE THERE ANY MACRO TRIGGERS WITHIN THE QUOTED TEXT TO RESOLVE?

This question is among the simplest to answer, because if the quoted text contains Macro triggers (e.g. `%` or `&`) that do not actually call Macro functions or point to Macro variables, then you would use the "`NR`" version of the otherwise appropriate quoting function. For example, if you need to mask a semicolon at compilation, the above guidelines would have you use the `%STR()` function, but if you also require that the Macro triggers in the quoted text not resolve until execution—or at all—then you should use `%NRSTR()` instead.



EXAMPLES

Note that these cases are not all-inclusive but rather, are meant to showcase frequently encountered quoting challenges.

EXAMPLE 1.A: MASKING A TYPE I TOKEN THROUGH COMPILATION

Type I Tokens can be masked through compilation using `%STR()` function within Macro definitions. This example is related to the error associated with the fact that 'NE' and 'OR' are state abbreviations that are also logical mnemonic operators for 'not equal to' and 'or'.

```
%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;
%EX_1(WV);
%Ex_1(ID);
```

Result

```
1    %MACRO Ex_1(State);
2       %IF (&State EQ %STR(NE) OR &State EQ %STR(OR) OR &State EQ MO OR
3            &State EQ KS OR       &State EQ WY OR       &State EQ ID)
4       %THEN %PUT The Oregon Trail ran through &State..;
5       %ELSE %PUT The Oregon Trail did not run through &State..;
6    %MEND Ex_1;
7    %EX_1(WV);
The Oregon Trail did not run through WV.
8    %Ex_1(ID);
The Oregon Trail ran through ID.
```

EXAMPLE 1.B: MASKING A TYPE I TOKEN UNTIL EXECUTION

Using `%QUOTE()` is necessary in this example when calling the above Macro because the Type I tokens (passed as parameters) would be misinterpreted during execution.

```
%Ex_1(%QUOTE(NE));
%EX_1(%QUOTE(OR));
```

Result

```
9    %Ex_1(%QUOTE(NE));
The Oregon Trail ran through NE.
10   %EX_1(%QUOTE(OR));
The Oregon Trail ran through OR.
```

EXAMPLE 2.A: MASKING A TYPE II (UNMATCHED PAIR TOKEN) THROUGH COMPILATION

The Type II token included in the following Macro definition must be preceded by a '%', or else the compiler would look for a matching single quote. Also note that the second Macro call of this example employs the `%QUOTE()` function to mask the logical operator "AND" through execution.

```
%MACRO Ex_2(Title);
   %IF (&Title EQ %STR(Charles Dickens%' Complete Works))
   %THEN %PUT  Out of Stock = &Title;
   %ELSE %PUT      In Stock = &Title;
%MEND Ex_2;
%Ex_2(The Unabridged Works of William Shakespeare);
%Ex_2(%QUOTE(Pride and Prejudice));
```

Result

```
1    %MACRO Ex_2(Title);
2       %IF (&Title EQ %STR(Charles Dickens%' Complete Works))
3       %THEN %PUT  Out of Stock = &Title;
4       %ELSE %PUT      In Stock = &Title;
5    %MEND Ex_2;
6    %Ex_2(The Unabridged Works of William Shakespeare);
In Stock = The Unabridged Works of William Shakespeare
7    %Ex_2(%QUOTE(Pride and Prejudice));
In Stock = Pride and Prejudice
```

EXAMPLE 2.B: MASKING A TYPE II TOKEN (UNMATCHED PAIR TOKEN) UNTIL EXECUTION

In the Macro call below, you must use the `%BQUOTE()` Macro function to pass the argument, since it contains an unmatched single quote that must be "blind" quoted. The `%QUOTE()` function would not work in this case because it would encounter problems when trying to match the unmatched token.

```
%Ex_2(%BQUOTE(Charles Dickens' Complete Works));
```

Result

```
8    %Ex_2(%BQUOTE(Charles Dickens' Complete Works));
Out of Stock = Charles Dickens' Complete Works
```

Also note that if you do not know exactly which arguments will be passed but you suspect some values to contain an unmatched token, there is no harm in using `%BQUOTE()` as a substitute for `%QUOTE()`.

```
%Ex_2(%BQUOTE(The Unabridged Works of William Shakespeare));
%Ex_2(%BQUOTE(Pride and Prejudice));
```

Result

```
9    %Ex_2(%BQUOTE(The Unabridged Works of William Shakespeare));
In Stock = The Unabridged Works of William Shakespeare
10   %Ex_2(%BQUOTE(Pride and Prejudice));
In Stock = Pride and Prejudice
```

EXAMPLE 3.A: MASKING A TYPE III TOKEN (MACRO TRIGGER) THROUGH COMPILATION

The argument within the Macro definition is static, so you would use `%NRSTR()` to prevent the Macro processor from attempting (and failing) to resolve the Macro trigger. Note the use of Macro quoting functions within the second and third Macro calls. These parameters must be quoted through execution to ensure proper outcomes.

```
%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));
```

Result

```
1    %MACRO Ex_3(Food);
2       %IF (&Food EQ %NRSTR(Rice & Beans))
3       %THEN %PUT Try the House Special - &Food;
4       %ELSE %PUT &Food - a tasty snack!;
5    %MEND Ex_3;
6    %Ex_3(Shrimp Cocktail);
Shrimp Cocktail - a tasty snack!
7    %Ex_3(%QUOTE("Fried" Green Tomatoes));
"Fried" Green Tomatoes - a tasty snack!
8    %Ex_3(%BQUOTE(Mama's Biscuits));
Mama's Biscuits - a tasty snack!
```

EXAMPLE 3.B: MASKING A TYPE III TOKEN (MACRO TRIGGER) UNTIL EXECUTION

This example requires that you use `%NRQUOTE()` in order to prevent the Macro processor from trying to resolve the Macro trigger within the passed parameter.

```
%Ex_3(%NRQUOTE(Rice & Beans));
```

Result

```
9    %Ex_3(%NRQUOTE(Rice & Beans));
Try the House Special - Rice & Beans
```

Also note that if you expect Type II AND Type III tokens to be passed within the parameters, you may use `%NRBQUOTE`, which combines both `%NRQUOTE()` and `%BQUOTE()`:

```
%Ex_3(%NRBQUOTE(Shrimp Cocktail));
%Ex_3(%NRBQUOTE("Fried" Green Tomatoes));
%Ex_3(%NRBQUOTE(Mama's Biscuits));
```

Result

```
10   %Ex_3(%NRBQUOTE(Shrimp Cocktail));
Shrimp Cocktail - a tasty snack!
11   %Ex_3(%NRBQUOTE("Fried" Green Tomatoes));
"Fried" Green Tomatoes - a tasty snack!
12   %Ex_3(%NRBQUOTE(Mama's Biscuits));
Mama's Biscuits - a tasty snack!
```

EXAMPLE 3.C: MASKING A TYPE III TOKEN (MACRO TRIGGER) FOREVER

Admittedly this is one of the more esoteric areas of the Macro quoting discussion, but `%SUPERQ()` is unique in that, in the rare case that you need to use it, there are no substitutes. It takes as its argument the name of a Macro variable—and this is important—without the '&' before the Macro variable reference. Using `%SUPERQ()` results in all tokens masked within the Macro variable, regardless of Type. The example below shows a common problem–masking a Macro trigger contained within a Macro variable.

```
DATA _NULL_;
   CALL SYMPUTX('Title','Revenue:  %Breakdown by Region');
RUN;
%LET NewTitle = %SUPERQ(Title);
***  Warning Given     ***;
%PUT Title = &Title;

***  No warning given  ***;
%PUT NewTitle = &NewTitle;
```

Result

```
1    DATA _NULL_;
2       CALL SYMPUTX('Title','Revenue:  %Breakdown by Region');
3    RUN;

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


4    %LET NewTitle = %SUPERQ(Title);
5     ***  Warning Given     ***;
6    %PUT Title = &Title;
WARNING: Apparent invocation of macro BREAKDOWN not resolved.
Title = Revenue:  %Breakdown by Region
7
8    ***  No warning given  ***;
9    %PUT NewTitle = &NewTitle;
NewTitle = Revenue:  %Breakdown by Region
```

CONCLUSION

We hope that this paper has given you a practical method for determining the Macro quoting function that applies in the many and varied situations that you may encounter. The examples are meant to show how to identify the simplest and least robust Macro quoting function that will achieve the intended result, while also showing the complexity of these issues.

REFERENCES

Carpenter, Art. 2003. "Macro Quoting Functions, Other Special Character Masking Tools, and How to Use Them." *Proceedings of the Twelfth Annual Northeast SAS User Group Conference*. Washington, DC.

O'Connor, Susan. 2003. "Secrets of Macro Quoting Functions – How and Why." *Proceedings of the Twelfth Annual Northeast SAS User Group Conference*. Washington, DC.

SAS Institute Inc. 2006. *SAS Macro Programming: Advanced Topics (Course Notes)*. Cary, NC: SAS Institute Inc.

Whitlock, Ian. 2003. "A Serious Look at Macro Quoting." *Proceedings of the Twenty-Eighth Annual SAS User Group International Conference*, Seattle. WA. http://www2.sas.com/proceedings/sugi28/011-28.pdf.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Brian Patterson
The College Board
45 Columbus Ave
New York, NY 10023-6992
BPatterson@collegeboard.org

Mylene Remigio
The College Board
45 Columbus Ave
New York, NY 10023-6992
MRemigio@collegeboard.org