

# SAS<sup>®</sup> GLOBAL FORUM 2015

The Journey Is Yours

## Session ID #3268 (E-Poster)

---

To %Bquote or not to %Bquote? That is the question..  
(which drives SAS<sup>®</sup> Macro programmers around the bend.)

Andrew Howell  
ANJ Solutions Pty Ltd  
Melbourne, Australia



# Why do we need macro functions?

- To protect special characters
  - Some text may be misinterpreted by the macro processor, such as:
    - » An unbalanced quote or bracket
    - » Text which can affect macro operations
      - » + - \* / < > = ~ ^ ~ ; , # blank AND OR NOT  
EQ NE LE LT GE GT IN % &
- To run SAS functions within macros
- To run SAS statements within macros

*.. and many more reasons..*



# When to protect, when to not protect?

- Some text requires protection only until a macro variable has been resolved. (*“Compilation”*)
- Some text continues to require protection after a macro variable has been resolved. (*“Execution”*)
- *It depends on your requirements..*
- This SAS Macro E-Poster offers a walk-through of some of the more common compilation & execution macro functions, with examples of what does (and does not) work, and why...



# Summary of macro functions

What to protect?	Compilation	Execution
	Protect text until the macro is resolved	Continue to protect text after the macro variable is resolved.
Text, but not macro triggers & %	%STR( )	%BQUOTE( )
Text & macro triggers	%NRSTR( )	%NRBQUOTE( ), %SUPERQ( )  %UNQUOTE( ) – <i>unprotects</i>
Other functions %SYSFUNC( ), %SYSCALL( )	%SUBSTR( ), %SCAN( )	%QSUBSTR( ), %QSCAN( )

*%QUOTE( ) and %NRQUOTE( )- intended as execution equivalents of the %STR( ) and %NRSTR( ) compilation functions - are still supported, but cannot manage unbalanced text: quotes, brackets, etc.*

*They were superseded by their successor “%B” macro execution functions: %BQUOTE( ) and %NRBQUOTE( ).*



# %STR( ) – Compilation macro function

- Protects text (not including macro triggers % & ) during compilation
- Once resolves, no longer protects text during macro execution
- Unbalanced quotes, brackets, etc, require a % prefix

Store an individual double quote in the macro variable VAR

*%BQUOTE( ) is covered in later slides; it protects the resolved value of VAR.*

%STR( ) protects the text inside the brackets; unbalanced text requires an additional % prefix.

```
data _null_;
  call symputx("VAR", '"');
run;

%macro CharTest;
  %if %bquote(&var) eq %str(%) %then %put Open Bracket;
  %else %if %bquote(&var) eq %str(%) %then %put Close Bracket;
  %else %if %bquote(&var) eq %str(%) %then %put Single Quote;
  %else %if %bquote(&var) eq %str(%) %then %put Double Quote;
  %else %put Char is neither a bracket nor a quote;
%mend;

%CharTest;
Double Quote
```



# %NRSTR( ) – Compilation macro function

- Protects text (including macro triggers % & ) during compilation
- Once resolves, no longer protects text during macro execution
- Unbalanced quotes, brackets, etc, require a % prefix

& is not protected; it will be treated as a macro trigger and try to resolve the non-existent macro variable **DOGS**.

```
%let CHOICE=Cats&Dogs;  
WARNING: Apparent symbolic reference DOGS not resolved.  
%put My choice is [&Choice];  
WARNING: Apparent symbolic reference DOGS not resolved.
```

**Almost!!** The macro processor still attempts to resolve &Dogs.  
%NRSTR( ) needs to be used when **assigning** the macro, not when **calling** it.

```
%let CHOICE=Cats&Dogs;  
WARNING: Apparent symbolic reference DOGS not resolved.  
%put My choice is [%nrstr(&Choice)];  
My choice is [&Choice]
```

%NRSTR( ) protects the & which will be treated as a text and will not try to resolve a macro variable value

```
%let CHOICE=%nrstr(Cats&Dogs);  
%put My choice is [&Choice];  
My choice is [Cats&Dogs]
```



# %BQUOTE( ) – Execution macro function

- Protects text during macro execution
- Protection include unbalanced quotes, brackets, etcd
- Does not protect macro triggers % &

The comma is not protected.  
This will attempt to pass two parameters to a macro function expecting only one parameter.

%BQUOTE() protects the comma.  
It is now treated as text, meaning a single parameter is passed to the macro function.

```
%macro YourName(name);  
    %put Your name is: &name;  
%mend;
```

```
%YourName(Andrew Howell);  
Your name is: Andrew Howell
```

```
%YourName(Howell,Andrew);  
ERROR: More positional parameters found than defined.
```

```
%YourName(%bquote(Howell,Andrew));  
Your name is: Howell,Andrew
```





# %BQUOTE( ) – Execution macro function

## ■ Another %BQUOTE( ) example..

The **&State** macro reference is not protected.

When **&State** is resolved to **OR**, and evaluated in the **%IF** statement, it will be treated as a logical “OR” operator.

```
%macro Wrong(State);  
    %if &State eq TX %then %put Texas;  
    %else %put Not Texas;  
%mend;  
%Wrong(TX);  
Texas  
%Wrong(OR);  
ERROR: A character operand was found in the %EVAL  
function or %IF condition where a numeric operand  
is required. The condition was:  
    &State eq TX  
ERROR: The macro WRONG will stop executing.
```

**%BQUOTE()** protects the resolved text.

Once **&State** is resolved to **OR**, it continues to be protected in the **%IF** statement evaluation; it is treated as text.

```
%macro Right(State);  
    %if %bquote(&State) eq TX %then %put Texas;  
    %else %put Not Texas;  
%mend;  
%Right(TX);  
Texas  
%Right(OR);  
Not Texas
```





# %BQUOTE( ) – Execution macro function

## ■ Extending the previous example..

The OR in “eq OR” is not protected; it is treated as a logical “OR” operator.

As above, plus &State is not protected, resulting in another OR which is also treated as a logical “OR” operator.

%BQUOTE() protects the resolved text of &State

%STR( ) protects OR

```
%macro Wrong(State);  
    %if &State eq OR %then %put Oregon;  
    %else %put Not Oregon;  
%mend;  
%Wrong(TX);  
ERROR: A character operand was found in the %EVAL  
function or %IF condition where a numeric operand is  
required. The condition was:  
    &State eq OR  
ERROR: The macro WRONG will stop executing.  
%Wrong(OR);  
ERROR: A character operand was found in the %EVAL  
function or %IF condition where a numeric operand is  
required. The condition was:  
    &State eq OR  
ERROR: The macro WRONG will stop executing.  
  
%macro Right(State);  
    %if %bquote(&State) eq %str(OR) %then %put Oregon;  
    %else %put Not Oregon;  
%mend;  
%Right(TX);  
Not Oregon  
%Right(OR);  
Oregon
```



# %NRBQUOTE( ) – Execution macro function

- Protects text (including macro triggers) during macro execution.

## AT&T is not protected.

The first warning is when &T is passed as a unprotected parameter.

The second warning is when &T is evaluated in the %IF statement, which then results in the macro error when %IF attempt to evaluate the & as text.

%NRBQUOTE( ) now protects AT&T when evaluated in the %IF statement, but still results in two warning when &Choice is resolved to AT&T which then attempts to resolve &T.

Combining %NTSTR( ) protects AT&T when passed as a parameter, and %NRBQUOTE( ) continues protection after &Choice is resolved to AT&T

```
%macro Wrong(Choice);  
    %if &Choice eq ATT %then %put ATT;  
    %else %put Not ATT;  
%mend;
```

```
%Wrong(Dell);  
Not ATT
```

```
%Wrong(AT&T);  
WARNING: Apparent symbolic reference T not resolved.  
WARNING: Apparent symbolic reference T not resolved.  
ERROR: A character operand was found in the %EVAL function or %IF  
condition where a numeric operand is required. The condition was:  
    &Choice eq ATT  
ERROR: The macro WRONG will stop executing.
```

```
%macro Right(Choice);  
    %if %nrquote(&Choice) eq %nrstr(AT&T) %then %put ATT;  
    %else %put Not ATT;  
%mend;
```

```
%Right(Dell);  
Not ATT
```

```
%Right(AT&T);  
WARNING: Apparent symbolic reference T not resolved.  
WARNING: Apparent symbolic reference T not resolved.  
ATT
```

```
%Right(%nrstr(AT&T));  
ATT
```



# %SUPERQ ( )

- Protects all text throughout macro compilation & execution.
- Macro triggers will not resolve, *unless passed to %UNQUOTE( )*
- Reference macro by name only (no preceding “&”)

Macro triggers stored within the macro variable are not protected.

When the macro VAR resolves, %A and &B will also resolve.

By referencing **VAR** within the %SUPERQ( ) function, all text stored within the **VAR** macro is protected, including (in this example) macro triggers.

```
%macro A;  
    %put Never get A !!;  
%mend;  
%let B=Never get B !!;  
  
data _null_;  
    call symputx('VAR','%A &B');  
run;
```

```
%put WRONG: VAR=&var;  
  
Never get A !!  
WRONG: VAR= Never get B !!
```

```
%put RIGHT: VAR=%superq(var);  
  
RIGHT: VAR=%A &B
```

# %UNQUOTE( )

- Unprotects protected text

**VAR1** is not protected anyway.  
Macros are resolved

**VAR2** is protected.  
Macros are not resolved

**VAR1** is not protected anyway.  
Macros are resolved.  
*[%UNQUOTE( ) has no effect.]*

**%UNQUOTE( )** overrides the **%SUPERQ( )**  
protection. Macros are resolved.

```
%macro A;  
    %put Here is my A macro program !!;  
%mend;  
%let B=Here is my B macro variable;  
  
data _null_;  
    call symputx('VAR1','%A &B');  
run;  
  
%let VAR2=%superq(VAR1);  
  
%put The text in the macro VAR1 is:&var1;  
Here is my A macro program !!  
The text in the macro VAR1 is: Here is my B macro variable  
  
%put The text in the macro VAR2 is:&var2;  
The text in the macro VAR2 is:%A &B  
  
%put Watch what happens when VAR1 resolves: %unquote(&var1);  
Here is my A macro program !!  
Watch what happens when VAR1 resolves: Here is my B macro variable  
  
%put Watch what happens when VAR2 resolves: %unquote(&var2);  
Here is my A macro program !!  
Watch what happens when VAR2 resolves: Here is my B macro variable
```

# %Q..( ) macros – Execution macro functions

- Emulates similar “compilation” macro functions

- %SCAN( ), %SUBSTR( ), etc.

- Continues to protect text after macro resolution

The **FULL1** macro was resolved when assigned. There will be no difference between %SCAN( ) and %QSCAN( ).

The **FULL2** macro is protected during compilation, but %SCAN( ) resolves **FULL2** and any macro references contained within FULL2's resolved value.

The **FULL2** macro is protected, %QSCAN( ) operates on the unresolved value of **FULL2**.

```
%let First=Andrew;  
%let Middle=R;  
%let Last=Howell;
```

```
%let Full1=&Last-&First-&Middle;  
%put &=FULL1;  
FULL1=Howell-Andrew-R
```

```
%let Full2=%nrstr(&Last-&First-&Middle);  
%put &=FULL2;  
FULL2=&Last-&First-&Middle
```

```
%put SCAN1: %scan(&Full1,1);  
SCAN1: Howell
```

```
%put QSCAN1:%qscan(&Full1,1);  
QSCAN1:Howell
```

```
%put SCAN2: %scan(&Full2,1,-);  
SCAN2: Howell
```

```
%put QSCAN2:%qscan(&Full2,1,-);  
QSCAN2:&Last
```

# %SYSFUNC()

- Allows macro execution of SAS Data Step functions

**Note:** &SYSTIME is not “now”.  
&SYSTIME is the time your SAS session began.

```
%put This SAS session began at: &SYSTIME;  
This SAS session began at: 21:18
```

**Incorrect:** When executed by the macro processor, this simply returns the literal text “time()”

```
%put The current time is: time();  
The current time is: time()
```

**%SYSFUNC()** causes the macro processor to execute the TIME() function, returning the number of seconds between midnight and “now”.

```
%put The current time is: %sysfunc(time());  
The current time is: 79450.4696559906
```

**%SYSFUNC()** also allows an optional format to be applied to the function's returned value.

```
%put The current time is: %sysfunc(time(), hhmm5.);  
The current time is: 22:04
```



# %SYSCALL( )

- Allows macro execution of SAS Data Step statements

In this example, %SYSCALL SET emulates the **DATA STEP's SET** statement, effectively creates a macro variable for each table variable – a very handy method to load a line of data into the macro symbol table.

```
%macro Test(dsn, row);  
    %let _id=%sysfunc(open(&dsn));  
    %syscall set(_id);  
    %let _rc=%sysfunc(fetchobs(&_id, &row));  
    %let _rc=%sysfunc(close(&_id));  
    %put _LOCAL_;  
%mend;  
%Test(sashelp.class, 2);  
TEST AGE 13  
TEST DSN sashelp.class  
TEST HEIGHT 56.5  
TEST NAME Alice  
TEST ROW 2  
TEST SEX F  
TEST WEIGHT 84  
TEST _ID 5  
TEST _RC 0
```





# Thank you for your interest in SAS Macros!

## REFERENCES

- SAS 9.4: Macro Language Reference, Third Edition
- SAS Macro 2 Training Course, SAS Education

## RECOMMENDED READING

- Carpenter, Art. 2004. **Carpenter's Complete Guide to the SAS® Macro Language, Second Edition.**
- SESUG 2008 paper CS-049  
**Macro Quoting**  
Toby Dunn, AMEDDC&S, Fort Sam Houston
- NESUG 1999 paper BT185  
**Secrets of Macro Quoting Functions – How and Why**  
Susan O'Connor, SAS Institute Inc., Cary, NC
- MWSUG 2010 paper  
**SAS® Macros: Top Ten Questions (and Answers!)**  
Kevin Russell –Technical Support Analyst, SAS Institute Inc.

I welcome your feedback and any questions.

## CONTACT INFORMATION

**Andrew Howell, ANJ Solutions Pty Ltd**

PO Box 765, Macleod VIC 3085 AUSTRALIA

Phone: + 61 407 898 513

Email: [info@anjsolutions.com.au](mailto:info@anjsolutions.com.au)

Twitter: [@AndrewAtANJ](https://twitter.com/AndrewAtANJ)

Skype: **AndrewAtANJ**

LinkedIn:  
<http://au.linkedin.com/in/howellandrew/>

SAS Communities:  
<https://communities.sas.com/people/AndrewHowell>





April 26-29  
Dallas, TX

