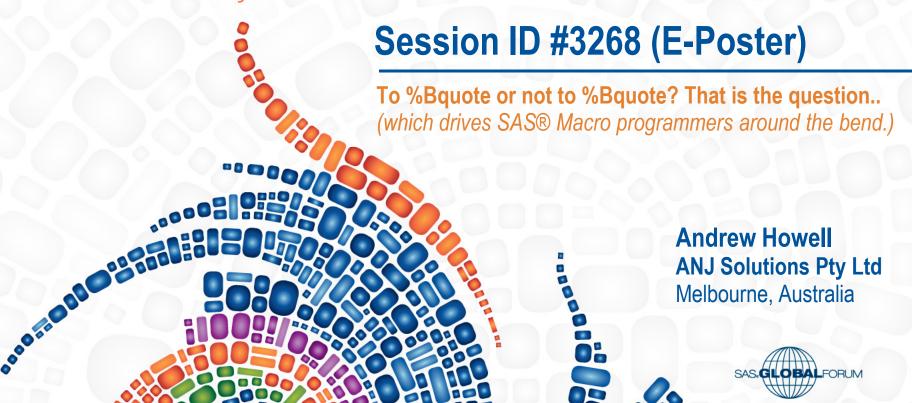
SAS°GLOBALFORUM 2015

The Journey Is Yours



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
 - * + * / < > = \neg ^ ~ ; , # blank AND OR NOT EQ NE LE LT GE GT IN % &
- To run SAS functions within macros
- To run SAS statements within macros







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() – unprotects
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They were superceded by their successor "%B" macro execution functions: %BQUOTE() and %NRBQUOTE().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
                                        data null;
                                              call symputx("VAR", '"');
                                        run;
%BQUOTE() is covered in later slides:
                                         %macro CharTest;
it protects the resolved value of VAR.
                                               %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;
 %STR() protects the text inside the
                                               %else %if %bquote(&var) eq %str(%") %then %put Double Quote;
brackets: unbalanced text requires an
                                               %else %put Char is neither a bracket nor a quote;
        additional % prefix.
                                        %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

```
%let CHOICE=Cats&Dogs;
& is not protected; it will be treated as a
                                             WARNING: Apparent symbolic reference DOGS not resolved.
  macro trigger and try to resolve the
                                             %put My choice is [&Choice];
  non-existent macro variable DOGS.
                                             WARNING: Apparent symbolic reference DOGS not resolved.
  Almost!! The macro processor still
                                             %let CHOICE=Cats&Dogs;
      attempts to resolve &Dogs.
                                             WARNING: Apparent symbolic reference DOGS not resolved.
  %NRSTR() needs to be used when
                                             %put My choice is [%nrstr(&Choice)];
assigning the macro, not when calling it.
                                             My choice is [&Choice]
                                             %let CHOICE=%nrstr(Cats&Dogs);
      %NRSTR() protects the &
                                             %put My choice is [&Choice];
which will be treated as a text and will not
                                             My choice is [Cats&Dogs]
  try to resolve a macro variable value
```





%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.

**Mend;

**Wend;

**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);
```

%macro YourName(name);



macro function.



%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.

%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 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.
%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 %bguote(&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

```
%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 %nrbquote(&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.
%Right(%nrstr(AT&T));
```







%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.

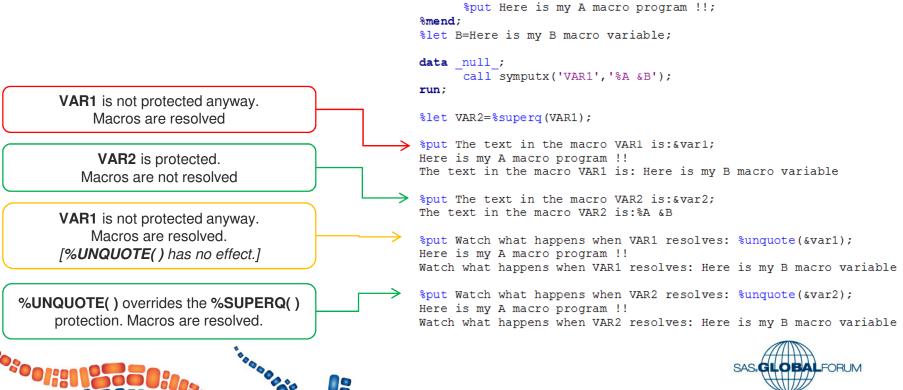
By referencing **VAR** within the **%SUPERQ()** function, all text stored within the **VAR** macro is protected, including (in this example) macro triggers.





%UNQUOTE()

Unprotects protected text



%macro A:

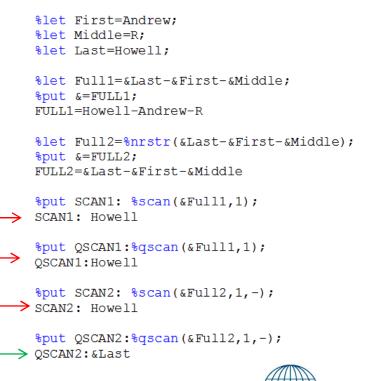
%Q..() macros – Execution macro functions

- 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 <u>FULL2's resolved value</u>.

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





%SYSFUNC()

Allows macro execution of SAS Data Step functions

Note: &SYSTIME is not "now". %put This SAS session began at: &SYSTIME; &SYSTIME is the time your SAS This SAS session began at: 21:18 session began. Incorrect: When executed by the %put The current time is: time(); macro processor, this simply returns The current time is: time() the literal text "time()" %SYSFUNC() causes the macro %put The current time is: %sysfunc(time()); processor to execute the TIME() The current time is: 79450.4696559906 function, returning the number of seconds between midnight and "now". \$\text{put The current time is: \sysfunc(time(),hhmm5.);} The current time is: 22:04 %SYSFUNC() also allows an optional format to be applied to the function's returned value.





%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

Twitter: @AndrewAtANJ

Skype: AndrewAtANJ

LinkedIn:

http://au.linkedin.com/in/howellandrew/

SAS Communities:

https://communities.sas.com/people/AndrewHowell







April 26-29 Dallas, TX

