SESUG Paper 145-2020

User-Written Functions to Check for SAS® Macro Quoting Triggers

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

Programmers may import a string into a SAS® macro variable, only to discover the hard way - via a failed run - that the string contained special characters or mnemonics that require use of quoting functions. Here we present PROC FCMP user-written functions to detect some macro triggers; these new functions use K-character functions for internationalization. The first function is the simplest and detects the presence of a % or & in the string. The second detects the presence of any special single character that may trigger macro quoting. The third detects many instances of mnemonics - potential logical and comparison operators: AND, OR, EQ, GE, etc. in the string (some constraints apply). The 4th function checks for unmatched left or right parentheses in the string; these can cause compile-time errors. Finally, we briefly mention some of the methods for handling strings with macro quoting triggers.

Keywords: macro language, quoting, masking, special characters, PROC FCMP, user-written functions.
SAS products: Base SAS.
User level: intermediate+.

OVERVIEW

A SAS program may import character strings – from an external database, a text file, a SAS dataset, user input from a prompt or other means – and inject that string into a SAS macro variable for later use. If the derived macro variable contains special characters or certain mnemonics, then the macro variable may throw an error when used later in the program unless the macro variable is “quoted” – the term used in SAS documentation to describe a masking process that will allow the variable to be used without error, at least in some circumstances.

Macro quoting is a complex issue; some quoting functions operate at compile time, others at run time. The SAS documentation on macro quoting is at the URL:

https://documentation.sas.com/?docsetId=mcrolref&docsetTarget=p07u5itr1teq0dn1bx0lli1ri5dy.htm&docsetVersion=9.4&locale=en

The special characters that are potentially problematic are: blank &%"()+-*/<>=¬°~;,#|^'
and the mnemonics are: LT GE AND GT OR IN NOT EQ NE LE.

Unmatched parentheses in a string –) or (- can cause problems in compilation, and so can unmatched single or double quotes – “ or ‘.

Billings (2019) describes a work-around that mitigates some macro quoting issues, by working with the string in hexadecimal format and converting to the target character encoding using %QSYSFUNC paired with INPUTC. However the workaround has constraints and it is not a cure-all for macro quoting issues.

Recognizing that it may be helpful to know if a string may need macro quoting or may contain unmatched parentheses, we present here 4 related user-written PROC FCMP functions. All of the functions below accept a character string as input parameter and return a numeric (indicator) result.

Function #1: Check for % or & in the string

% and & are the most likely characters to require quoting, and if unquoted, to throw an error. This is the simplest of the 4 functions:

```
proc fcmp outlib=fcndev.macsupp.utility;
      function ckmqap(str $);
            k = kindexc(str,'&%');
            return (k);
      endsub;
```

The function returns a numeric 0 if there are no % or & characters in the string; otherwise it returns a positive integer. KINDEXC is used here instead of INDEXC, for internationalization. If this is the only function that is of interest to you here, PROC FCMP can be skipped – instead, just use the KINDEXC logic above.

Function #2: Check for any special (macro trigger) character except blank

We do not check for blanks because they are (often) not a significant issue, and because SAS has the ANYSPACE DATA step function. Once again we use KINDEXC, and here the ability of KINDEC to handle multiple sets of target characters makes it easier to use than INDEXC would be in SBCS datasets.

```
      function ckmqsc(str $);
            k = kindexc(str,'&%"()+-*/<>=¬°~;,#|^',"'");
            return (k);
      endsub;
```

The characters that are searched for are: `&%"()+-*/<>=¬°~;,#|^'` The function returns a numeric 0 if none of the listed characters are in the string, otherwise it returns a positive integer.

Function #3: Check for mnemonics that may trigger quoting

This is rather more complex; it checks a string for possible instances of any of the following mnemonics: LT GE AND GT OR IN NOT EQ NE LE

Again, K-functions are used for internationalization. The function result is numeric; it returns a 0 if none of the mnemonics above are present in a string, otherwise a 1 is returned. Let XX denote any of the above mnemonics and b a blank; a 1 is returned by the function if the target string, after trimming any left and right blanks:

- is an exact match to a mnemonic
- begins with XXb
- contains bXXb (this test includes ending with bXX).

The logic above is intended to try to differentiate possible mnemonics from the same text string that may occur in a word. This function should be considered experimental.

```
      function ckmqop(str $);
```

```
            length str2 $32767;
            str2 = kupcase(ktrim(kleft(str)));
            r = 0;

            if (not missing(str2)) and
                  (klength(str2) >1) then
                  do;
                        if (klength(str2) = 2) and (str2 IN
("OR","IN","EQ","NE","LE","LT","GE","GT")) then
                              r=1;
                        else if (klength(str2) = 3) and (str2 IN
("AND","NOT")) then
                              r=1;
                        else if (klength(str2) > 3) then
                              do;
                                    if (Kindex(str2," AND ")>0) OR
(Ksubstr(str2,1,4) ="AND ") then
                                          r=1;

                                    if (Kindex(str2," NOT ")>0) OR
(Ksubstr(str2,1,4) ="NOT ") then
                                          r=1;

                                    if (Kindex(str2," OR ")>0) OR
(Ksubstr(str2,1,3) ="OR ") then
                                          r=1;

                                    if (Kindex(str2," IN ")>0) OR
(Ksubstr(str2,1,3) ="IN ") then
                                          r=1;

                                    if (Kindex(str2," EQ ")>0) OR
(Ksubstr(str2,1,3) ="EQ ") then
                                          r=1;

                                    if (Kindex(str2," NE ")>0) OR
(Ksubstr(str2,1,3) ="NE ") then
                                          r=1;

                                    if (Kindex(str2," LT ")>0) OR
(Ksubstr(str2,1,3) ="LT ") then
                                          r=1;

                                    if (Kindex(str2," LE ")>0) OR
(Ksubstr(str2,1,3) ="LE ") then
                                          r=1;

                                    if (Kindex(str2," GT ")>0) OR
(Ksubstr(str2,1,3) ="GT ") then
                                          r=1;

                                    if (Kindex(str2," GE ")>0) OR
(Ksubstr(str2,1,3) ="GE ") then
                                          r=1;
                              end;

                        return (r);
            end;
    endsub;
```

Function #4: Check for unmatched (or) characters

Unmatched (or) characters in a string need to be preprocessed for use in %STR, %NRSTR:

https://documentation.sas.com/?docsetId=mcrolref&docsetTarget=p0pnc7p9n4h6g5n16g6js048nhfl.htm&docsetVersion=9.4&locale=en

This function checks for unmatched (or) characters in a string; this is more difficult than one might expect. For example, a very simplistic view would be to count the number of left and right parentheses in a string; if the counts don't match then unmatched parentheses are present. The function does this test but such a test would fail to flag an obvious pathological case like: ')))(((' which has an equal number of both left and right parentheses, but they are obviously unmatched. Clearly, additional tests are needed to detect unmatched parentheses.

The function steps through a target string and creates a derived subset string that contains ONLY the (and) characters from the target string, i.e., a reduced string. If the result of this is a null/missing string, then there are no (or) characters in the target string and the function returns a numeric 0 as result. Note that this derived subset string is SBCS, single byte characters.

If there are (and/or) characters in the string, then the subset string is processed. In a loop, the string is searched for occurrences of '()'; when this is found it is replaced by XX which is then removed by compression. The process continues until all pairs have been removed; at that time if there are any characters left then there are unmatched (and/or) parentheses in the string. A numeric 1 is returned if unmatched (or) are present in the string. This function can detect unmatched parentheses but it does nor pre-pend them with % as needed for use in %STR, %NRSTR.   If this situation is present, handling the string in hexadecimal notation is another option to consider as discussed in Billings (2019).

```
    function ckunbp(str $);
          length mystr sbcs $32767;
          length R1 RF RN L1 LF LN 8;
          call missing(sbcs,mystr,R1,RF,L1,LF);
          mystr = kcompress(str," ");
          RN=0;
          LN=0;
          kl = klength(mystr);

          if (not missing(mystr)) then
                do;
                      j=0;

                      do i=1 to kl;
                            if (ksubstr(mystr,i,1) = "(") then
                                  do;
                                        j=j+1;
                                        substr(sbcs,j,1) = "(";

                                        if missing(L1) then
                                              L1=j;
                                        LF = j;
                                        LN = LN+1;
                                  end;
```

```
                              if (ksubstr(mystr,i,1) = ")") then
                                    do;
                                          j=j+1;
                                          substr(sbcs,j,1) = ")";

                                          if missing(R1) then
                                                R1=j;
                                          RF = j;
                                          RN = RN+1;
                                    end;
                        end;
                  end;

            sbcs = strip(sbcs);
            kl2 = length(sbcs);

            if (missing(mystr)) or (missing(sbcs)) or (RN+LN = 0) then
                  do;
                        r=0;
                        return(r);
                  end;
            else
                  do;
                        r=0;

                        if (RN ne LN) or (LF > RF) or (R1 < L1) or
(kindex(sbcs,'()') = 0) then
                              r=1;
                        else
                              do;
                                    do while (kindex(sbcs,'()') ne 0);
                                          m = kindex(sbcs,'()');
                                          substr(sbcs,m,2) = "xx";
                                          sbcs = compress(sbcs,"x");
                                    end;

                                    if (not missing(sbcs)) then
                                          r=1;
                              end;

                        return(r);
                  end;
      endsub;
run;
```

Supplement

The macro functions %SUPERQ and %NRBQUOTE are very useful in quoting input strings for macro variables. The functions above are potentially useful in deciding if an input string may need quoting or other special handling.

No function is provided to check for/identify instances of unmatched ‘ or “ in a string. It is not clear to this writer re: how SAS parses anomalous, “extreme” examples of strings with multiple, embedded single and double quotes that intersect.

In general, PROC FCMP functions can be called using %SYSFUNC, %QSYSFUNC. The functions above have been tested (only) with strings in DATA steps. As these functions are intended to detect situations that trigger quoting, %SYSFUNC should not be used with these functions. They have not been tested with %QSYSFUNC but –at least in theory – should work with %QSYSFUNC.

Summary

PROC FCMP user-written functions are provided above to support macro quoting applications. The following functions are supplied:

- Detect presence of a % or & in a string
- Detect the presence of any special character that may trigger quoting in a string, excluding blanks
- Detect the presence in a string of a possible mnemonic that may trigger macro quoting
- Detect the presence in a string of unmatched (or) characters.

Note: As of the date of publication, the author is an employee of MUFG Union Bank, N.A. The content of this paper does not reflect the views or opinions or work product of MUFG Union Bank.

APPENDIX 1:
BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

REFERENCES

Billings T. (2019). Hexadecimal Encoding Can Mitigate Some SAS Macro Quoting Issues. 2019 *Southeast SAS Users Group (SESUG) Conference Proceedings.* URL: https://www.lexjansen.com/sesug/2019/SESUG2019_Paper-156_Final_PDF.pdf

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): https://goo.gl/uCUHoa

Note: Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
Email: tebillings@gmail.com