The Design and Coding of Simple Utility Macros

Frank DiIorio
CodeCrafters, Inc., Chapel Hill NC

A SAS programmer with even a modest amount of experience will eventually have one or both of the following internal dialogues. "Why didn't they provide a [bizzare and somewhat convoluted operation] function in Version 9? Maybe I should have sent it in as a SASware ballot item …" Another perrenial favorite: "Why on earth does %put _all_ display variables in such a peculiar and unreadable format?" Both are good questions, and in both cases it's probably better to create your own solution than search for an answer.

This article describes two utility macros that address some simple needs. The first, `QuoteList`, takes a macro variable with one or more unquoted tokens and returns a variable with the tokens quoted and optionally upper-cased. The second macro, `AllMacVars`, prints the beginning of each global macro variable, listing them in alphabetical order. The intent of the article is twofold. First, the code is intended to be useful in and of itself. Second, as the code is explained, we demonstrate some underlying good programming practices.

QuoteList

Here's another internal dialog: "The calling program handed me a macro variable with a list of values that I'll have to use later in a DATA step array [notice that truly obsessive programmers capitalize even when no one's looking]. I could borrow some quoting code from another program, but maybe it's time to write something generalized."

Right. The impetus for general-purpose programs is, sometimes, a predefined need. More often, though, it is from the realization that the activity has been done before. Maybe it was not *exactly* the same, but it is close enough that a well-chosen set of parameters can be used to generalize the program. One possible result of our immediate need to insert quotes is the macro shown below:

```
%macro QuoteList(macin=, macout=, upcase=no, print=no); {1}
%let continue = yes; {2}
%if &macin. = %then %do; {2}
    %put QuoteList-> MacIn was not specified. Execution terminating. ;
    %let continue = no;
    %end;
%if &macout. = %then %do; {2}
    %put QuoteList-> MacOut was not specified. Execution terminating. ;
    %let continue = no;
    %end;
%if &continue. = yes %then %do; {2}
    %global &MacOut.;
    %if %upcase(&print.) = YES %then %put QuoteList-> Input variable &MacIn. [&&&macin.] ; {3}
    %let &MacOut. = %sysfunc(compbl(&&&MacIn.)); {4}
    %let &MacOut. = "%sysfunc(tranwrd(&&&MacOut., %str( ), " "))"; {5}
    %if %upcase(&UpCase.) = YES %then %let &MacOut. = %upcase(&&&MacOut.);
    %if %upcase(&print.) = YES %then %put QuoteList-> Output variable &macout. [&&&macout.] ; {3}
    %end;
%mend;
```

Let's look at what `QuoteList` does, following the highlighted items in `{braces}`

{1} Always use keyword parameters to avoid confusion. Likewise, when you develop other utilities requiring similar sorts of parameters – uppercasing, printing, etc. – try to keep the parameter names and values identical; `print=no` in this utility and `prt=n` in another only serve to confuse the user, and the user will often be you!

Parameters:
`macin` → name of input (unquoted) macro variable. Required.

`macout` → name of output (quoted) macro variable. May be identical to `macin`. Required.
`upcase` → '`yes`' (without quotes) to uppercase `macout`. Default is to leave case as-is.
`print` → '`yes`' (without quotes) to write input and output values to the SAS Log. Default is no printing.

{2} Accumulate error messages, setting a flag – macro variable `continue` – to indicate whether we can continue. This method lets us alert the user to multiple specification problems at once (i.e., both MacIn and MacOut were missing).

{3} Print on the basis of the upper-cased parameter value. Show where the message came from by prefixing all messages with `QuoteList->`. This is especially helpful in lengthy Logs and programs with many macro calls.

{4} Prepare the macro variable by removing extraneous blanks.

{5} Enclose the entire input macro variable in quotes, then use the `TRANWRD` function to convert each blank ( ) to what amounts to a quote ending one word and starting another (" ").

Let's look at an example of `QuoteList`'s use. Assume macro variable `idlist` has the value `ms183 ms198 va122` We can use `QuoteList` as follows:

```
%QuoteList(macin=idlist, macout=Qidlist, upcase=yes)
data subset;
     set master(where=(ptid in (&Qidlist.));
```

The DATA step code executed by SAS looks like:

```
data subset;
     set master(where=(ptid in ("MS183" "MS198" "VA122"));
```

An obvious extension to such a macro is to return the number of tokens that were quoted. This would allow the programmer to prevent situations where `macin` is null and the chaos that would result when errant / unusable values of `macout` were referenced. The number of tokens requirement could be satisfied by some extra code in the macro. Even better, we could call a utility macro that counts tokens, thus allowing us to share the counting capability among multiple programs.

AllMacVars

This macro utilizes SAS's metadata (aka "dictionary tables") to produce a clean, simple to read listing of global macro variables. For an in-depth discussion of dictionary tables, see the SouthEast SAS User Group (SESUG) 2003 Proceedings paper "Dictionary Tables: Essential Tools for Serious Applications" (it is also in the SUGI 29 Proceedings). The paper is also available at http://www.CodeCraftersInc.com.

```
%macro AllMacVars(dest=log, StartWith=); {1}
%local _mprint;
%let _mprint = %sysfunc(getoption(mprint)); {2}
options nomprint nonotes; {2}
proc sql noprint;
     create table _macvars_ as
     select *
     from dictionary.macros
     where offset=0 and scope='GLOBAL' {3}
     %if &StartWith. ^= %then & name like ("%upcase(&StartWith.)%") ;
     order by name
     ;
quit;
%if &SQLobs. = 0 %then %do; {4}
   %if &StartWith. ^= %then %put AllMacVars-> No global macro variables began with [&StartWith.];
        %else               %put AllMacVars-> No global macro variables were defined. ;
    %goto bottom;
    %end;
```

```
data _null_;
     set _macvars_ end=eof;
     file &dest. notitles;
     if _n_ = 1 then
        put / 'Macro Variable' @34 'First 65 Characters' /
              32*'=' +1 65*'=' ;
     put name $33. value $char65.;
     if eof then put   32*'=' +1 65*'='
                     / "# of variables = " _n_
                     / 98*'='
                     ;
run;
proc delete data=_macvars_; {5}
run;
%bottom: options &_mprint. notes ; {2}
%mend;
```

Let's look at what `AllMacVars` does, following the highlighted items in `{braces}`

{1} Refer to the `QuoteList` discussion, above, for comments about parameter naming and values.

Parameters:

`dest` → Location of output. Default is the SAS Log. The value could be print or even an external file, enclosed in quotes.

`StartWith` → Restrict the listing to those variables whose name begins with this value. Enter in mixed case; the value will be compared in upper case. Default is all global macro variables.

{2} To reduce clutter in the SAS Log, save the value of the `MPRINT` option, then turn off `NOTES` and `MPRINT`. Reset them at the bottom of the macro, just before terminating.

{3} Restrict `OFFSET` to 0 to avoid duplication of names. Variables with values longer than 200 characters are split into multiple observations, with `OFFSET` values of 200, 400, etc.

{4} If no metadata met the subsetting criteria, print a message and exit.

{5} Clean up after ourselves before exiting. Remember the `DELETE` procedure? Simple to use, with less overhead than `PROC DATASETS`.

Let's look at the macro in action. Suppose we defined two global macro variables, `global1` and `testmacvar`. A call to `AllMacVars` would produce the following output:

```
Macro Variable                  First 65 Characters
=============================== ================================================================
GLOBAL1                         G1
TESTMACVAR                      tmv
=============================== ================================================================
# of variables = 2
================================================================================================
```

The display is easy to read, clearly labeled. The alphabetical ordering of the variable names doesn't present a huge advantage in our simple test case, but it doesn't take much imagination to see how this presentation would be helpful when dozens of variables are involved..

Closing Comments

There are, I'm sure, many different ways to attack the problems that gave rise to the macros described in this article. This is the non-standardized charm of SAS programming – there are always many different and valid ways to skin even the simplest cat. The macros shown here are simply presented to show Real World examples of sound utility design. Look at them as much for *how* they work as *what* they do.