Macro Design Considerations for a Word Wrapping Macro

Ian Whitlock, Westat, Rockville, MD

Abstract

This paper presents a short tutorial on a macro designed for flexibility. A word wrapping problem is used to make the author's points.

%WordWrap is used within a DATA step to generate partial DATA step code. The discussion centers around why the macro should operate at this level and how the user can get control of the output without a great deal of effort.

For example: Generate a report writing, the value of LONGSTRING from the work data set REPORT. The value must begin in column 20 for up to 40 characters. When LONGSTRING is more than 40 bytes it should be broken on a word boundary and continued to the next line in column 20. The user might write a short macro, UserMac, to distribute the processing within his DATA step.

```
%macro usermac ;
   i + 1 ;
   if i = 1 then link topline ;
   else link continue ;
%mend  usermac ;
```

followed by the step.

```
data _null_ ;
   length wrap $ 40 ;
   set report ;
   i = 0 ;
   %wordwrap (invar = longstring ,
              outvar = wrap,
              mac = usermac )
return ;


topline:
   file print ;
   put // "Report for " id
      @20 wrap

      @70 "more stuff"  ;
return ;


continue:
   file print ;
   put @20 wrap ;
return ;
run ;
```


Introduction

SAS® Version 8 simplifies the job of making a word wrapping macro since we can assume the material to be wrapped is all in one variable.

First we have to answer some questions:

1. How much should the macro control?

   Compete DATA step or part of one

2. Should it produce an array, a data set, or a file?

Trying to control the whole step is wrong. What if the consumer wants to wrap two or more variables? How can we know what the consumer of the macro wants? All the possibilities listed in question 2 are reasonable and under the appropriate circumstances correct. A macro should not try to make decisions that it is in no position to make.

The only reasonable decision is to let the consumer decide. Now what is the best form to give him the information? An array is an obvious possibility. Should it be defined by the consumer or by the macro? If the macro, how big should the array be? What if the consumer merely wants to write each value in a PUT statement? Now an array doesn't look so good.

What is left? We must obtain a portion of the string in a loop. How can we give the consumer control? Asking the consumer to control the loop is too much. If he has to do that, he might as well write all of the code. The answer is we have to ask the consumer what code should be executed inside the loop producing each wrapped portion of the character string. We have to provide a parameter for the consumer to name a macro, which holds the code, he wants executed.

What are the other parameters? We have to provide a parameter for the name of the variable to wrap, otherwise he always has to know and be careful to use our variable name and, moreover, make sure it doesn't conflict with any of his names. This is a rather unreasonable request for a macro that should help the consumer. How do we name the output variable? Again we should provide a parameter for the user to assign his name, but this time it makes sense to give him a default name that he can change if desired.

How long should the output variable be? Well if the consumer names this variable and writes code to use this variable, it makes sense to let him assign the length. But how can we get that length? Version 8 provides the function VLENGTH. What if the consumer doesn't want to be bothered? We can provide a length parameter. If he isn't using version 8, or doesn't want to make a length statement, then he must provide the needed length.

We conclude that we need parameters:

INVAR to name the input variable to be wrapped
OUTVAR to name the output variable
MAC to name the consumer macro to execute
LEN in case the length of &OUTVAR has not already been determined.

The Macro

With this preparation we are ready to write a useful and friendly macro. To prevent the collision of variable names we specify that the consuming code should not use any variables beginning with a double underscore.

The basic algorithm is to mark off the length of &OUTVAR and then back up to a space. What if the end falls at a word boundary? Then we don't need to back up. Hence we should start looking for the space after the end of the marked length. Two variables, __START and __END are used to keep track of where we are in the string to be wrapped. At the beginning,. __END is 1 and we loop until __END is greater than or equal to the last relevant position in &INVAR. The final trick is to skip over leading spaces in &OUTVAR.

```
%macro WordWrap
   ( invar =  , /* reqd inpt var */
     outvar=chunk, /* result var */
     mac = m1 , /* required mac  */
      /*name of macro to execute */
      /* when a chunk is ready   */
     len=/* null deflt to len of */
         /* &outvar (null req V8)*/
   ) ;
  /* ----------------------------
   Break at space bndry within reqd
   length unless word longer than
   reqd len, then word is broken
   into reqd len pieces.
   The consumer controls what is
   done with created var, &outvar,
   in a macro that he specifies.
   Assume no vars begin with double
   underscore in PDV

   Examples:

   1) %macro m ; output w ; %mend ;
      data w ;
         length chunk $ 40 ;
         set in ;
         %wordwrap ( invar = string
                   , mac = m )
      run ;

   2) %macro m ;
         i + 1 ;
         if i <= dim ( a ) then
            a(i) = chunk ;
      %mend ;

      data w ( keep = a1 - a10 ) ;
         array a (10) $ 80 ;
         i = 0 ;
         set w ;
         %wordwrap(invar=longstring
                   , mac = m )
      run ;

   Ian Whitlock whitloi1@westat.com
   5may2000
   ---------------------------------
*/

   %if %length(&len) = 0 %then
     %let len = vlength(&outvar) ;
   drop __len __start __end __i ;
   __len = length ( &invar ) ;
   __end = 1 ;
   do until ( __end = __len ) ;
     do __end = __end to __len ;
       if substr(&invar, __end, 1)
            ^= "" then leave ;
     end ;
     __start = __end - 1 ;
     __end = __start + &len + 1;
     if __end > __len then
       __i = __len ;
     else
     do __i=__end to __start+1 by-1;
       if substr(&invar,__i,1) = " "
          then leave ;
     end ;
     if __i < __start + 1 then
       __i = __end - 1;
     else
       __end = __i ;
     &outvar = substr(&invar,
              __start+1,__i-__start)
     ;
     %&mac
   end ;
%mend wordWrap ;
```

Conclusion

With a little care one can learn to write useful friendly macros helpful to the consumer, rather than providing obscurity in ampersands and %-signs.

The author may be contacted by mail at

Ian Whitlock
Westat
1650 Research Boulevard
Rockville, MD 20850

or by e-mail

whitloi1@westat.com