Paper 052-2007

Names, Names, Names - Make Me a List

Ian Whitlock, Kennett Square, PA

Abstract

Any number of macros have been written to generate enumerated lists of names. For example,

```
lib.w1 lib.w2 lib.w3 ...
```

But they probably would not work with

```
lib.a lib.b lib.c
```

or

```
q1_a q1_b q2_a q2_b
```

Now consider a problem to convert a list of character variables to numeric with the same names while keeping the same names. For example,

```
data w ( drop = __: ) ;
   set w ( rename = ( x=__x y=__y z=__z ) ) ;
   x = input ( __x , best32. ) ;
   y = input ( __y , best32. ) ;
   z = input ( __z , best32. ) ;
run ;
```

The renames (in red) form a list and the assignment statements (in blue) form a list.

All these problems involve lists but one usually uses special purpose macros for each problem. General tools will be discussed to handle these problems in a uniform manner without modifying the underlying macros.

Introduction

A central problem to SAS® macro programming is the handling of lists. This paper introduces a set of macros that can be used to handle many standard problems involving lists. Each macro is self contained, yet the macros as a set have a consistent user interface making them easy to use. As a system the macros provide a strong service to the consumer, but can accept any one macro and get useful service from it alone. I think, that individually and as a set they illustrate good design principles and coding practices. Consequently they present a reasonably small self contained problem, independent of any data content, that the reader as a programmer should find easy to appreciate and yet complex enough to prove interesting. Hence this paper can be considered as a very short tutorial on SAS macro programming.

By the nature of the subject, macro quoting is needed; yet the reader need only know the basic quoting functions %STR and %SUPERQ. %STR hides code symbols like the comma and semicolon at macro compile time and after. %SUPERQ works at macro execution time to hide the code generated when a named macro variable is evaluated.

One immediate design question is whether to use positional parameters. I strongly prefer key word parameters as the only way to produce readable consumer code and the easy extensibility of macros required by ever changing maintenance requirements. Consequently all parameters in the system are key word parameters. However, the names have been kept short to mollify the lazy programmer. L is used for the list parameter when only one list is required, and the parameters L1 and L2 are used when two lists are required. The associated parameters to define separators are LSEP, SEP1, and SEP2 respectively. OSEP is used for the output list, i.e. the separator for the returned value (list) when you think of these macros as functions. Although the system was primarily developed to

work with space separated lists of SAS names, it can be used for other purposes as long as one does not expect it to work with a space separated list quoted items which internally contain spaces, e.g. "A B" "C D" will not be understood as a two element list, but
"A B","C D" will when the separator is a comma.

Another question worth asking is whether it is not more convenient to have the parameters name list variables rather than the lists themselves. There are definite advantages to specifying lists by name as well as by presenting the list. Hence all the macros allow both usages. I used LV, LV1, and LV2 for the parameters naming list variables. By default the LV variables are empty, i.e. the named variable is not used. Consequently the consumer can decide which method is more convenient for his code. Whenever, the LV variable is not empty it must be the name of the list variable (not its value). If you already have a list variable and the list is long then using the LV form will be more efficient. When you have lists short enough for the efficiency to not matter, then the L form may be easier to read.

How should the empty list be treated? Since an empty list is sort of the zero of the system all the macros do the reasonable thing with an empty list and produce an empty list. It is not considered an error or something worthy of a warning.

All of the macros consist only of macro instructions except for the "returned value". Hence these macros can usually be used in %LET and %PUT statements. However, when the output separator, &OSEP, requires macro quoting you must be careful to apply the appropriate macro quoting from the outside. For example, when the separator is a semicolon you might enclose the macro in a call to %BQUOTE to use it in a %LET or %PUT statement.

The basic binary list operators

There are three basic functions for making new lists out of old lists. The first, to concatenate two lists, is so simple that no macro is needed or helpful in accomplishing the task. The second, to pair corresponding elements of two lists by juxtaposition is handled by the macro ZIP. The third, to pair every element of the first list with every element of the second is handled by the macro XPROD.

The ZIP of the lists (A B C) and (X Y Z) is (AX BY CZ), i.e. the corresponding elements of two lists are zipped together. This list can be formed by

```
    %zip(l1=a b c, l2=x y z)
```

If one of the lists is shorter than the other then the list of paired elements is returned and a warning is posted to the log. Although ZIP is not as often required as XPROD, it is still useful enough to include in the system.

XPROD, on the other hand, is very important in building new lists from old ones. The name, XPROD, stands for Cartesian (or cross) product. In particular it can create the first two lists in the abstract:

```
    lib.w1 lib.w2 lib.w3

    lib.a lib.b lib.c
```

Note that the list lengths do not have to match and that the most common cases are where the first or second list has only one element.

```
    %xprod(l1=lib.w, l2=1 2 3)

    %xprod(l1=lib., l2=a b c)
```

When both lists contain multiple elements the resulting list is grouped by the values in the first list. Hence,

```
    %xprod(l1=a_ b_ c_, l2=x y)
```

produces

```
    a_x a_y b_x b_y c_x c_y
```

Usually one treats these two problems as separate leading to two different kinds of list making macros, since 1 2 3 is easy to generate in an iterative loop while the list a b c is not. To unify these tasks under one macro I added a macro, RANGE, to make it easy to produce lists like, 1 2 3.

Now consider the list

```
    q1_a q1_b q2_a q2_b
```

from the abstract. It can be produced from the component lists (Q), (1 2) and (_A _B) using two calls to XPROD.

```
    %xprod(l1=q, l2=%xprod(l1=%range(to=2), l2=_a _b))
```

or perhaps in the easier to read form

```
    %let list = %xprod( l1=%range(to=2), l2=_a _b) ;
    %xprod(l1=q, lv2=list)
```

Using %REPLACE to solve simple repetition problems

Traditionally the conversion problem mentioned in the abstract is solved with two macros say, %RENAME and %CHAR2NUM, to create the two lists shown in red and blue in

```
    data w ( drop = __: ) ;
       set w ( rename = ( x=__x y=__y z=__z ) ) ;
       x = input ( __x , best32. ) ;
       y = input ( __y , best32. ) ;
       z = input ( __z , best32. ) ;
    run ;
```

For example the solution to the problem in the abstract is given by

```
    data w ( drop = __: ) ;
       set w ( rename = (%rename(x y z)) ) ;
       %char2num(x y z)
    run ;
```

Note key word parameters were not used in %RENAME or %CHAR2NUM because they are almost "throw away" macros. Here is the code.

```
    %macro rename ( list, pref=__ ) ;
       %* make a rename list from &LIST *;
       %replace ( l=&list, code = # = &pref# )
    %mend  rename ;

    %macro char2num ( list , pref = __ ) ;
       %* make list of char to num assignments *;
       %replace ( l=&list
                , code= %str(# = input(&pref#,best32.);)
                )
    %mend  char2num ;
```

To see understand the %REPLACE call in RENAME you need to know that each item in the list L=X Y Z, will replace the symbolic variable # in CODE = # =__#. Consequently,

```
    x=__x y=__y z=__z
```

will be generated and returned to %RENAME as generated code. Similarly, %CHAR2NUM generates a sequence (list) of assignment statements. Note the use of %STR in the value of CODE to hide the semicolon.

Here RENAME and CHAR2NUM are introduced only to make the code more readable by showing the intent. REPLACE is the real work horse. Note that a call to REPLACE can often replace a simple %DO-loop in the consuming macro code.

Once you have learned how to use %REPLACE, it helps you to solve many little coding repetition problems where only relatively simple substitution is involved. The symbolic variable named by the parameter KEY is given the default value # because as a symbol it stands out and is not often used in code. However, you can use more complex expressions #1#, #2#, or -X-, <XYZ> etc. The ability to have multiple symbolic variables allows nested calls to REPLACE. The chose key values must not interfere with the intended code.

Handling quote marks and separators

It is annoying to write lists where each element must appear in quote marks. The macro QT allows one to add these quote marks to a simple list.

```
    %qt(l=a b c)
```

produces the list

```
    "a" "b" "c"
```

The parameter QT controls which quote mark is used and the parameter OSEP controls what the separator will be. It is very convenient when generating code for a comma separated language such as PROC SQL.

If one can add the quote marks to a list of elements, then sometimes it must be convenient to remove them. This is the job of the macro UQT.

Throughout this system space separation has been emphasized and planned for because the elements of a list are most commonly made up of SAS names. Moreover the separator for each list is provided by a parameter so changing separators is not usually required when using the list processing macros presented. However, there are other occasions where it might be convenient to change the separator. Hence the macro, CHANGESEP, has been added to the basic macros.

Efficiency

I have used a style of coding that collects the information to be generated by the macro in a macro variable which is then evaluated at the end of the macro, since I find this style clearest. However, in reviewing the paper, Chang Chung has shown that it is significantly more efficient to use the style shown at the end for the second version of RANGE. It is applicable to all of the macros presented. Here are sample timing results when Chang's suggested is followed for RANGE while the TO parameter is changed and other parameters have default values. As you can see the loop length has to be over 50 before it begins to matter.

TO=	50	100	200	400	800
Chang time	0.01 sec.	0.01 sec.	0.02 sec.	0.04 sec.	0.08 sec.
Original time	0.01 sec.	0.02 sec.	0.05 sec.	0.17 sec.	0.72 sec.

Conclusion

This paper has introduced the usage for six list processing macros. The reader is left to study the details of the macros in the appendix. I have made no attempt to be complete, i.e. to give all possible types of list processing macros. Perhaps the first thing you should note is that %RANGE is the only macro to produce a list without having at least one list as input. So where do the lists come from? You might begin by asking what list making macros should be added to the system to manufacture lists.

PROC SQL is an excellent list maker, but beyond the scope of this paper.

I am interested in any problems or criticism of the macros presented in the appendix that you might encounter when studying or learning to use them in your SAS programming. You might think a list length function is missing function. However, I wanted to make lists the primary objects of interest and show that knowing their lengths has little to do with their manipulation.

Contact information

Ian Whitlock

29 Lonsdale Lane
Kennett Square, PA 19348
Ian.Whitlock@comcast.net

Appendix

```
/* ListMacs -

   Objective: basic self contained consistent list processing macros

   Functions:

      1) zip two lists together by joining correponding elements
         a b and c d ==> ac bd
      2) take cross product of two lists, i.e. join every element
         of the second list with each element of the first list
         a b and c d ==> ac ad bc bd
      3) create arithmetic sequence of integers

      4) replace symbolic variable in block of code with
         each element of a list
         a b and code = #=__# ==> a=__a b=__b
         where # is the symbolic variable
      5) add quotes to each element in a list(omitted due to space requirement)
      6) remove quotes from each element of a list (omitted)
      7) change the separator for a list (omitted)

   Corresponding macro headers:

      1) %zip (l1= , lv1= , sep1=,  l2= , l2v= , sep2= , osep= )
      2) %xprod ( l1= , lv1= , l2= , lv2=
                , sep1=%str( ), sep2=%str( ), osep=%str( ) )
      3) %range ( to=, from=1 , step=1 , sep=%str( ) )
      4) %replace ( l= , lv= , code= , key= , lsep=%str( ), osep=%str( ) )
      5) %qt   ( l=, lv= , lsep= %str( ), qt = %str(%"), osep=%str( ) )
      6) %uqt  ( l=, lv= , lsep= %str( ), qt = %str(%"), osep=%str( ) )
      7) %changesep ( l=, lv= , lsep= %str( ), osep=%str(,) )

   Notes:

      All macros return a list and all the macros require one or two lists
       as input.
      The L# parameter gives a list.
      The LV# gives a possible override naming the list variable.
      The SEP parameters give the coresponding separator.
      By default all separators are one or more spaces, with the exception
       of CHANGESEP where the default output separtor is a comma.
      If any separator other than space is used the consumer is responsible
       for insuring single separator between items and no unwanted spaces.
*/

%macro zip
   ( l1=         /* first list */
   , lv1=        /* external variable override for first list   */
   , sep1=%str( )    /* separator between the joined elements   */
   , l2=         /* second list                                 */
   , lv2=        /* external variable override for second list */
   , sep2=%str( )   /* separator between the joined elements   */
   , osep=%str( )   /* separator between new elements          */
   ) ;

   /*  %zip ( l1= a b , l2= c d ) produces ac bd
       so does
           %let list1 = a b ;
           %let list2 = c d ;
           %zip (lv1=list1, lv2=list2)

       If lists do not have same length shorter length used and
       warning to the log.  Empty lists result in empty list and
       no message.

       If the LV options are used then L1, L2, and ZIP_: should be
       avoided for variable names.
   */

   %local zip_i zip_1 zip_2 zip_list ;
   %if %length(&lv1) = 0 %then
      %let lv1 = l1 ;
   %if %length(&lv2) = 0 %then
      %let lv2 = l2 ;

   %do zip_i = 1 %to &sysmaxlong ;
      %let zip_1 = %qscan(%superq(&lv1) , &zip_i, &sep1 ) ;
      %let zip_2 = %qscan(%superq(&lv2) , &zip_i, &sep2 ) ;
      %if %length(&zip_1) = 0 or %length(&zip_2) = 0 %then
          %goto check ;
      %if &zip_i = 1 %then
         %let zip_list = &zip_1&zip_2 ;
      %else
         %let zip_list = &zip_list&osep&zip_1&zip_2 ;
   %end ;

   %check:
      %if %length(&zip_1) > 0 or %length(&zip_2) > 0 %then
          %put WARNING: Macro ZIP - list lengths do not match - shorter used. ;

   %unquote(&zip_list)

%mend  zip ;

/* ============================================================== */

%macro xprod
   ( l1=        /* first list */
   , lv1=       /* external variable override for first list     */
   , sep1=%str( )   /* separator between elements of first list  */
   , l2=        /* second list                                   */
   , lv2=       /* external variable override for second list    */
   , sep2=%str( )   /* separator between elements of second list */
   , osep=%str( )   /* separator between elements of new list    */
   ) ;
   /*  %xprod ( l1= a b , l2= c d ) produces ac ad bc bd
       so does
           %let list1 = a b ;
           %let list2 = c d ;
           %xprod (lv1=list1, lv2=list2)

       LV1 and LV2 provide override to specify external variable
       name instead of lists.

       If one or more of the lists are empty then the empty
       list is returned.

       If the LV options are used then L1, L2, and XP_: should be
       avoided for variable names.
   */

   %local xp_i xp_j xp_1 xp_2 xp_list ;
   %if %length(&lv1) = 0 %then
      %let lv1 = l1 ;
   %if %length(&lv2) = 0 %then
      %let lv2 = l2 ;

   %do xp_i = 1 %to &sysmaxlong ;
      %let xp_1 = %qscan(%superq(&lv1), &xp_i, &sep1) ;
      %if %length(&xp_1) = 0 %then %goto endloop1 ;
      %do xp_j = 1 %to &sysmaxlong ;
         %let xp_2 = %qscan(%superq(&lv2), &xp_j, &sep2) ;
         %if %length(&xp_2) = 0 %then %goto endloop2 ;
         %if &xp_i = 1 and &xp_j = 1 %then
            %let xp_list = &xp_1&xp_2 ;
         %else
            %let xp_list = &xp_list&osep&xp_1&xp_2 ;
      %end ;
      %endloop2:
   %end ;
   %endloop1:

   %unquote(&xp_list)

%mend  xprod ;

/* =============================================================== */

%macro range
   ( to=1         /* end integer value         */
   , from=1       /* starting integer value    */
   , step=1       /* increment integer         */
   , osep=%str( )  /* sparator between integers */
   ) ;

   /*
      return sequence of integers starting at &FROM going to &TO
      in steps of &step

      %range(to=5) produces 1 2 3 4 5
   */

   %local rg_i rg_list ;

   %do rg_i = &from %to &to %by &step ;
      %if &rg_i = &from %then
         %let rg_list = &rg_i ;
      %else
         %let rg_list = &rg_list&osep&rg_i ;
   %end ;

   %unquote(&rg_list)

%mend  range ;

/* ================================================================ */

%macro replace
   ( l=            /* value list */
   , lv=           /* external variable override for value list  */
   , lsep=%str( )  /* separator between values                   */
   , code=         /* block of code containing symbolic variable */
   , key=#         /* symbolic variable to replace  (#abc# etc.) */
   , osep=%str( )  /* separator between new elements             */
                   /* may be %str(;) when code is statement      */
                   /* if so remember to add closing semicolon    */
   ) ;
   /* for elt in the list replace key in code

      LV provides override to specify external variable
      name instead of list.

      If the LV option is used then L and RG_: should be
      avoided for variable names.
   */
   %local rg_i rg_w rg_list ;
   %if %length(&lv) = 0 %then
      %let lv = l ;

   %if %length(%superq(&lv)) = 0 /*or %index(%superq(code),&key) = 0*/ %then
   %do ;
       %let rg_list = %superq(code) ;
       %goto mexit ;
   %end ;

   %do rg_i = 1 %to &sysmaxlong ;
      %let rg_w = %qscan(%superq(&lv),&rg_i,&lsep) ;
      %if %length(&rg_w) = 0 %then %goto mexit ;
      %if &rg_i = 1 %then
         %let rg_list = %sysfunc(tranwrd(%superq(code),&key,&rg_w)) ;
      %else
         %let rg_list =
            &rg_list&osep%sysfunc(tranwrd(%superq(code),&key,&rg_w)) ;
   %end ;
%mexit:
   %unquote(&rg_list)

%mend  replace ;


/* ============================================================== */

%macro range   /* second more efficient version due to Chang Chung */
   ( to=1         /* end integer value          */
   , from=1       /* starting integer value     */
   , step=1       /* increment integer          */
   , osep=%str( )  /* sparator between integers */
   ) ;

   /*
      return sequence of integers starting at &FROM going to &TO
      in steps of &step

      %range(to=5) produces 1 2 3 4 5
   */

   %local rg_i ;

   %do rg_i = &from %to &to %by &step ;
      %if &rg_i = &from %then
         %do;&rg_i%end ;
      %else
         %do;&osep&rg_i%end ;
   %end ;
%mend  range ;
```

Contact information

Ian Whitlock

29 Lonsdale Lane
Kennett Square, PA 19348
Ian.Whitlock@comcast.net