Paper 029-30

Text Utility Macros for Manipulating Lists of Variable Names

Robert J. Morris, RTI International, Research Triangle Park, NC

ABSTRACT

When using data sets that have many variables, tasks such as mass variable renaming or generating table statements in PROC FREQ can involve lots of copy-and-paste and hand manipulation. This paper presents several text utility macros that can make these and other tasks easier to do the first time and easier to maintain later. The available macros include: counting the number of variables in a list, adding a prefix or suffix to each variable in a list, creating a list of variables that have a numeric counter suffix, and interleaving two lists of variables. One of the best features of the macros is that they can be combined and nested to allow even greater utility and flexibility. The paper discusses how to use each of the macros and provides examples of their use.

INTRODUCTION

Lists of variable names are used throughout SAS® programs: in KEEP and RENAME statements, in ARRAY definitions, in TABLE statements, and in MODEL statements, just to name a few. Sometimes the same list of variable names appears in many places in a program. Sometimes the repeated lists are not the same but are very similar. For example, one list may be just like another except that each variable name has an “r_” prefix added to it. Other times, variable lists need to be renamed to comply with a piece of code's requirements or a project's naming convention.

Writing and maintaining a program becomes more difficult as these situations occur more often and as the number of variables involved increases. Adding and removing variables becomes error-prone because the variable lists must be changed in numerous places. Programs become less readable because of the long lists of variable names scattered throughout them. These are exactly the types of problems that led to the development of the text utility macros described here.

The text utility macros will be introduced through a series of examples. After the examples, a brief reference section will describe each of the macros and their arguments. The paper will conclude by providing the macro source code.

EXAMPLES

Throughout these examples, we will assume that the following macro variable has been defined to hold your original list of variable names:

```
%let orig_vars = a01a a01b d01 d02 t1_r t2_r;
```

To help keep the examples below less cluttered, this list includes only 6 variable names. The advantages of using the text utility macros become more apparent with longer lists of variables. When reading through the examples, try to imagine the code when there are 60 or 100 variables instead of just 6. In the programs for which these macros were developed, lists were used containing up to 400 variables.

EXAMPLE 1 – RECODING VARIABLES

Suppose you want to create a set of variables that are set to 1 whenever the `&orig_vars` variables are positive and set to 0 whenever the `&orig_vars` variables are nonpositive. You want your recoded variables to have the same names as the original variables, but with a “r_” prefix. One way to do this would be:

```
data newdata;
    set mydata;
    array orig[*] &orig_vars;
    array recode[*] r_a01a r_a01b r_d01 r_d02 r_t1_r r_t2_r;
    do i=1 to dim(orig);
        if (orig[i] > 0) then recode[i] = 1; else recode[i] = 0;
    end;
    run;
```

However, you do not need to hardcode the new variable names if you use one of the text utility macros:

```
data newdata;
    set mydata;
    array orig[*] &orig_vars;
    array recode[*] %add_string(&orig_vars, r_, location=prefix);
    do i=1 to dim(orig);
        if (orig[i] > 0) then recode[i] = 1; else recode[i] = 0;
    end;
```

```
    run;
```

Notice that nothing changed except the variable list in the second array statement. You have replaced the hardcoded list with a call to the `%add_string` macro. The `%add_string` macro adds the text string in its second argument to each item in the list provided as its first argument. Whether the text is added as a prefix or suffix is controlled by the optional third argument.

Also notice that in this example, you did not really save any keystrokes by using the `%add_string` macro. However, imagine if the list had 60 variables instead of just 6. The amount of typing in the first snippet would increase nearly tenfold while the second snippet would need no modification at all. Now imagine you decided to remove a variable from the `&orig_vars` list. In the first snippet, you would need to delete that variable from the second array statement, but the second snippet would again need no modification.

EXAMPLE 2 – PROC MEANS OUTPUT DATA SET

This is another example of the `%add_string` macro. Here, you use the MEANS procedure to find the mean and the max of each of your variables. You want the mean variables in the output data set to have a "_mean" suffix, and you want the max variables to have a "_max" suffix. Without using `%add_string`, you might use this code:

```
proc means data=mydata noprint;
    output out=newdata
           mean(&orig_vars) = a01a_mean a01b_mean d01_mean
                              d02_mean  t1_r_mean t2_r_mean
           max(&orig_vars)  = a01a_max  a01b_max  d01_max
                              d02_max   t1_r_max  t2_r_max;
    run;
```

But using `%add_string`, the code would look like:

```
proc means data=mydata noprint;
    output out=newdata
           mean(&orig_vars) = %add_string(&orig_vars, _mean)
           max(&orig_vars)  = %add_string(&orig_vars, _max);
    run;
```

As in Example 1, adding or removing a variable requires no modification in the second snippet. In the first snippet, it would require modifying not one but two variable lists.

EXAMPLE 3 – RENAMING VARIABLES TO HAVE A COMMON PREFIX

This example is similar to Examples 1 and 2 in that you want to create a new variable list that is the original list with a prefix or suffix added to each variable. The difference here, though, is that you want to use the RENAME statement to rename the original variables to the new ones. You could do this with:

```
data newdata;
    set mydata;
    rename a01a=r_a01a a01b=r_a01b d01=r_d01 d02=r_d02 t1_r=r_t1_r t2_r=r_t2_r;
    run;
```

This looks like a task that should be handled by a text utility macro. But you cannot use `%add_string` here because the original variable list and the new variable list are mixed together. This is different from the earlier examples since the original and new variable lists are separated in those. Instead, you can use a different macro called `%rename_string`. This macro is designed specifically for the rename statement, and can be used as follows:

```
data newdata;
    set mydata;
    rename %rename_string(&orig_vars, r_, location=prefix);
    run;
```

Like `%add_string`, `%rename_string` adds a prefix or suffix to each variable in its first argument, but, in addition, it also includes the original variables and equals signs in a format suitable for the RENAME statement.

EXAMPLE 4 – PROC FREQ TABLE STATEMENT

Now suppose you want to validate the recoded variables created in Example 1 by examining frequency tables of each original variable by its corresponding recoded variable. This can be done with:

```
proc freq data=newdata;
    tables a01a*r_a01a a01b*r_a01b d01*r_d01 d02*r_d02 t1_r*r_t1_r t2_r*r_t2_r;
    run;
```

Using the text utility macros %parallel_join and %add_string, though, this can be done with (lines wrapped for visual convenience):

```
proc freq data=newdata;
    tables %parallel_join(
               &orig_vars,
               %add_string(&orig_vars, r_, location=prefix),
               *
               );
    run;
```

The %parallel_join macro takes two variable name lists as arguments and a text string by which it joins each pair of variable names. Here, the first variable list is your original list of variables. The second variable list is generated through the %add_string macro, and is your list of variables with an "r_" prefix. The text string provided in the third argument is the asterisk character, which is used to indicate interactions in the tables statement.

EXAMPLE 5 – RENAMING VARIABLES TO HAVE A COUNTER SUFFIX

This final example is the most complicated because it combines three different utility macros: %parallel_join and the newly introduced %num_tokens and %suffix_counter. Suppose someone has given you a piece of code that performs some fancy algorithm you want to use. The only problem is that it expects the variable names in your data set to be named qvar1-qvarN, but your variables are named a01a a01b d01 d02 t1_r t2_r. Because you do not want to modify the code you have been given, you need to rename your variables to fit the code's requirements. One way to do this is to hardcode the variable renames:

```
data newdata;
    set mydata;
    rename a01a=qvar1 a01b=qvar2 d01=qvar3 d02=qvar4 t1_r=qvar5 t2_r=qvar6;
    run;
```

However, you can avoid this hardcoding by using the text utility macros (lines wrapped for visual convenience):

```
data newdata;
    set mydata;
    rename %parallel_join(
               &orig_vars,
               %suffix_counter(qvar, %num_tokens(&orig_vars)),
               =
               );
    run;
```

You again use the %parallel_join macro to join two lists of variable names. The first list is your original list of variables. The second list, which is generated by the %suffix_counter macro, is qvar1 qvar2 qvar3 qvar4 qvar5 qvar6. The variables in the two lists are joined by an equals sign as required by the rename statement. The %suffix_counter macro creates a list of variable names by appending an incremental numeric counter to a base name. Here, the base name is qvar, which is given as the first argument to %suffix_counter. The second argument specifies where the counter ends. In this case, you want to have the same number of qvar variables as there are variables in &orig_vars. So you use the %num_tokens macro to count the number of variables in &orig_vars, and pass this in as the second argument to %suffix_counter.

MACRO REFERENCE

This section lists all the text utility macros, along with a brief description of their purpose and arguments. Please see the code listing section below for the macros' source code.

%num_tokens

Purpose:
Count the number of "tokens" (variables) in a list.
Required arguments:
words – the variable list
Optional arguments:
delim – the character(s) separating each variable in the &words list [default: space]
Example:

- %num_tokens(a b c d e) produces the text 5
- %num_tokens(a-b-c-d-e, delim=-) produces the text 5

%add_string

Purpose:
Add a text string to each variable in a list as either a prefix or suffix.

Required arguments:
words – the variable list
str – the text string to add to each variable in the &words list
Optional arguments:
location – whether to add the text string as a prefix or suffix [prefix|suffix, default: suffix]
delim – the character(s) separating each variable in the &words list [default: space]
Examples:

- `%add_string(a b c, _max)` produces the text `a_max b_max c_max`
- `%add_string(a b c, max_, location=prefix)` produces the text `max_a max_b max_c`
- `%add_string(%str(a,b,c), _max, delim=%str(,))` produces the text `a_max,b_max,c_max`

`%rename_string`
Purpose:
Create a list suitable for the rename statement where the variables in a list are renamed so that they have a common text string as a prefix or suffix.
Required arguments:
words – the variable list containing the original names
str – the text string to add to each renamed variable
Optional arguments:
location – whether to add the text string as a prefix or suffix [prefix|suffix, default: suffix]
delim – the character(s) separating each variable in the &words list [default: space]
Examples:

- `%rename_string(a b c, _1)` produces the text `a=a_1 b=b_1 c=c_1`
- `%rename_string(a b c, r_, location=prefix)` produces the text `a=r_a b=r_b c=r_c`
- `%rename_string(a|b|c, _1, delim=|)` produces the text `a=a_1 b=b_1 c=c_1`

`%suffix_counter`
Purpose:
Create a list of variable names formed by adding a numeric counter suffix to a base name.
Required arguments:
base – the text that should be the base of the variable names
end – the last number in the counter
Optional arguments:
start – the first number in the counter [default: 1]
zpad – the number of digits to which the counter should be padded. Use zpad=0 for no padding. [default: 0]
Examples:

- `%suffix_counter(v, 4)` produces the text `v1 v2 v3 v4`
- `%suffix_counter(v, 14, start=10)` produces the text `v10 v11 v12 v13 v14`
- `%suffix_counter(v, 4, zpad=2)` produces the text `v01 v02 v03 v04`

`%parallel_join`
Purpose:
Join two variable lists by connecting each variable in the first list to its corresponding variable in the second list by a text string.
Required arguments:
words1 – the first variable list
words2 – the second variable list
joinstr – the text string used to join the variable names in &words1 with the variable names in &words2
Optional arguments:
delim1 – the character(s) separating each variable in the &words1 list [default: space]
delim2 – the character(s) separating each variable in the &words2 list [default: space]
Examples:

- `%parallel_join(a b c, d e f, *)` produces the text `a*d b*e c*f`
- `%parallel_join(a#b#c, d.e.f, *, delim1=#, delim2=.)` produces the text `a*d b*e c*f`

CODE LISTING

This section presents the source code for each of the text utility macros. Documentation is provided in the macro reference section above.

```
%macro num_tokens(words, delim=%str( ));
    %local counter;

    %* Loop through the words list, incrementing a counter for each word found. ;
    %let counter = 1;
    %do %while (%length(%scan(&words, &counter, &delim)) > 0);
         %let counter = %eval(&counter + 1);
```

```
    %end;

    %* Our loop above pushes the counter past the number of words by 1. ;
    %let counter = %eval(&counter - 1);

    %* Output the count of the number of words. ;
    &counter

%mend num_tokens;


%macro add_string(words, str, delim=%str( ), location=suffix);
    %local outstr i word num_words;

    %* Verify macro arguments. ;
    %if (%length(&words) eq 0) %then %do;
        %put ***ERROR(add_string): Required argument 'words' is missing.;
        %goto exit;
    %end;
    %if (%length(&str) eq 0) %then %do;
        %put ***ERROR(add_string): Required argument 'str' is missing.;
        %goto exit;
    %end;
    %if (%upcase(&location) ne SUFFIX and %upcase(&location) ne PREFIX) %then %do;
        %put ***ERROR(add_string): Optional argument 'location' must be;
        %put ***                   set to SUFFIX or PREFIX.;
        %goto exit;
    %end;

    %* Build the outstr by looping through the words list and adding the
     * requested string onto each word. ;
    %let outstr = ;
    %let num_words = %num_tokens(&words, delim=&delim);
    %do i=1 %to &num_words;
        %let word = %scan(&words, &i, &delim);
        %if (&i eq 1) %then %do;
            %if (%upcase(&location) eq PREFIX) %then %do;
                %let outstr = &str&word;
            %end;
            %else %do;
                %let outstr = &word&str;
            %end;
        %end;
        %else %do;
            %if (%upcase(&location) eq PREFIX) %then %do;
                %let outstr = &outstr&delim&str&word;
            %end;
            %else %do;
                %let outstr = &outstr&delim&word&str;
            %end;
        %end;
    %end;

    %* Output the new list of words. ;
    &outstr

    %exit:
%mend add_string;


%macro rename_string(words, str, delim=%str( ), location=suffix);

    %* Verify macro arguments. ;
    %if (%length(&words) eq 0) %then %do;
        %put ***ERROR(rename_string): Required argument 'words' is missing.;
        %goto exit;
    %end;
```

```
    %if (%length(&str) eq 0) %then %do;
        %put ***ERROR(rename_string): Required argument 'str' is missing.;
        %goto exit;
    %end;
    %if (%upcase(&location) ne SUFFIX and %upcase(&location) ne PREFIX) %then %do;
        %put ***ERROR(rename_string): Optional argument 'location' must be;
        %put ***                      set to SUFFIX or PREFIX.;
        %goto exit;
    %end;

    %* Since rename_string is just a special case of parallel_join,
     * simply pass the appropriate arguments on to that macro. ;
    %parallel_join(
        &words,
        %add_string(&words, &str, delim=&delim, location=&location),
        =,
        delim1 = &delim,
        delim2 = &delim
        )

    %exit:
%mend rename_string;


%macro suffix_counter(base, end, start=1, zpad=0);
    %local outstr i counter;

    %* Verify macro arguments. ;
    %if (%length(&base) eq 0) %then %do;
        %put ***ERROR(suffix_counter): Required argument 'base' is missing.;
        %goto exit;
    %end;
    %if (%length(&end) eq 0) %then %do;
        %put ***ERROR(suffix_counter): Required argument 'end' is missing.;
        %goto exit;
    %end;
    %if (&end < &start) %then %do;
        %put ***ERROR(suffix_counter): The 'end' argument must not be less;
        %put ***                       than the 'start' argument.;
        %goto exit;
    %end;

    %* Construct the outstr by looping from &start to &end, adding the counter
     * value to &base in each iteration. To handle the zero-padding, use the
     * putn function to format the counter variable with the Z. format. ;
    %let outstr=;
    %do i=&start %to &end;
        %if (&zpad > 0) %then %do;
            %let counter = %sysfunc(putn(&i, z&zpad..));
        %end;
        %else %do;
            %let counter = &i;
        %end;
        %let outstr=&outstr &base&counter;
    %end;

    %* Output the new list. ;
    &outstr

    %exit:
%mend suffix_counter;


%macro parallel_join(words1, words2, joinstr, delim1=%str( ), delim2=%str( ));
    %local i num_words1 num_words2 word outstr;

    %* Verify macro arguments. ;
```

```
    %if (%length(&words1) eq 0) %then %do;
        %put ***ERROR(parallel_join): Required argument 'words1' is missing.;
        %goto exit;
    %end;
    %if (%length(&words2) eq 0) %then %do;
        %put ***ERROR(parallel_join): Required argument 'words2' is missing.;
        %goto exit;
    %end;
    %if (%length(&joinstr) eq 0) %then %do;
        %put ***ERROR(parallel_join): Required argument 'joinstr' is missing.;
        %goto exit;
    %end;

    %* Find the number of words in each list. ;
    %let num_words1 = %num_tokens(&words1, delim=&delim1);
    %let num_words2 = %num_tokens(&words2, delim=&delim2);

    %* Check the number of words. ;
    %if (&num_words1 ne &num_words2) %then %do;
        %put ***ERROR(parallel_join): The number of words in 'words1' and;
        %put ***                      'words2' must be equal.;
        %goto exit;
    %end;

    %* Build the outstr by looping through the corresponding words and joining
     * them by the joinstr. ;
    %let outstr=;
    %do i = 1 %to &num_words1;
        %let word = %scan(&words1, &i, &delim1);
        %let outstr = &outstr &word&joinstr%scan(&words2, &i, &delim2);
    %end;

    %* Output the list of joined words. ;
    &outstr

    %exit:
%mend parallel_join;
```

CONCLUSION

This paper presents several text utility macros that can make your programming life a little easier when manipulating long lists of variable names. Examples of tasks that can benefit from using the macros include recoding variables, naming the variables on a PROC MEANS output data set, renaming variables to have a common prefix, generating table statements in PROC FREQ, and creating variable lists with a counter suffix. These examples are provided just to give you an idea of what can be done. Many other tasks are also possible, especially by exploiting the ability to combine and nest the macros.

REFERENCES

SAS Institute Inc., *SAS Macro Language: Reference, Version 8*, Cary, NC: SAS Institute, Inc., 1999. 310 pages.

ACKNOWLEDGMENTS

I would like to acknowledge the contribution of Gabriel Cano, who authored the original `%num_tokens` macro.

CONTACT INFORMATION

The author welcomes any comments, questions, or suggestions for improvements. Contact the author at:

Robert J. (Joey) Morris
RTI International
3040 Cornwallis Rd.
Research Triangle Park, NC 27709
Email: rjmorris@rti.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.