Creating New Variable Names from Old Variable Names

Julia L. Bienias, U.S. Bureau of the Census, Washington, DC

ABSRACT

In this tip I describe how to automatically create a set of variables with new names and values, where the new names are derived from a set of variables already on the file. For example, one might want to take a set of variables and create a set of new variables that are the logarithms of the original values. The new variables might have names like "L_SALES" and "L_INCOME" to indicate they are logged values. One can write a simple macro that will do this for a particular specified variable name, but several steps are required to do it automatically for a set of variables whose names are not pre-specified in the macro call. The macro code presented here uses array statements and the "vname" function, which retrieves the name of a variable as a character variable. These variable names are placed in macro variables (via the CALL SYMPUT function) and can be retrieved in a subsequent data step and assigned values.

PRELIMINARY CODE & TEST DATA

```
data test;
  weight=5;
  sales=350;
  payroll=250;
  dividend=20;
  taxpaid=10;
  run;
```

* list the original ("old") variables;

```
%let num_vars = 4;
%macro myvars;
  sales payroll dividend taxpaid
  %mend;
```

* Execute a "dummy" DATA _NULL_ step, to place the macro variables NAME1, NAME2, etc., on the macro symbol table during regular compile time. Without this step, references to these variables do not resolve properly.;

```
data _null_;
   do i = 1 to &num_vars;
      call symput('name'||left(i),'junk');
      end;
   run;
```

* List macro variables that will contain the new variable names. This list will be used to reference the new variables.;

```
%macro newvars;
   %do i = 1 %to &num_vars;
      &&name&i
      %end;
%mend;
```

THE MAIN MACRO

* Create new variable names from old variable names by placing 2 characters in front of the old name. Place these new names in a series of macro variables called NAME1, NAME2, etc.;

```
%macro new(prefix);
data _null_;
   length oldname $ 8. newname $ 8.;
   array vars{&num_vars} %myvars;
   do i = 1 to &num_vars;
      call vname(vars{i},oldname);
      temp = min(length(oldname),6);
newname = "&prefix"||substr(oldname,1,temp);
      call symput('name'||left(i),newname);
      end;
   run;
%mend;
```

EXAMPLE

* execute the macro NEW, specifying a prefix of W_, for WEIGHTED;

```
%new(w_)
```

* Now, we can give assign values to the new variables, such as multiplying the original value by the WEIGHT variable.;

```
data test (drop=i);
   set test;
   array vars{&num_vars} %myvars;
   array new{&num_vars} %newvars;
   do i = 1 to &num_vars;
      new{i} = vars{i} * weight;
      end;
   run;
```

The following shows how the macros are resolved during a PROC PRINT.

```
proc print data=test; var weight
MLOGIC(MYVARS):  Beginning execution.
                      %myvars
MPRINT(MYVARS): SALES PAYROLL DIVIDEND
                TAXPAID
MLOGIC(MYVARS):  Ending execution.
                      %newvars; run;
MLOGIC(NEWVARS):  Beginning execution.
SYMBOLGEN:  Macro variable NUM_VARS resolves
            to 4
MLOGIC(NEWVARS):  %DO loop beginning; index
            variable I; start value is 1;
            stop value is 4; by value is 1.
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 1
```

```
SYMBOLGEN:  Macro variable NAME1 resolves to
            w_SALES
MLOGIC(NEWVARS):  %DO loop index variable I
            is now 2; loop will iterate
            again.
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 2
SYMBOLGEN:  Macro variable NAME2 resolves to
            w_PAYROL
MLOGIC(NEWVARS):  %DO loop index variable I
            is now 3; loop will iterate
            again.
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 3
SYMBOLGEN:  Macro variable NAME3 resolves to
            w_DIVIDE
MLOGIC(NEWVARS):  %DO loop index variable I
            is now 4; loop will iterate
            again.
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 4
SYMBOLGEN:  Macro variable NAME4 resolves to
            w_TAXPAI
MLOGIC(NEWVARS):  %DO loop index variable I
            is now 5; loop will not iterate
            again.
MPRINT(NEWVARS): W_SALES W_PAYROL W_DIVIDE
                 W_TAXPAI
MLOGIC(NEWVARS):  Ending execution.
```

Output:

```
WEIGHT SALES PAYROLL DIVIDEND TAXPAID

  5     350    250       20        10

    W_SALES W_PAYROL W_DIVIDE W_TAXPAI

      1750    1250       100       50
```

VARIATIONS

In my example, I have inserted a prefix in front of the variable name, "moving the name over" and truncating it (if necessary) at the end. One could instead add characters at the end as a suffix (e.g., "_W"), again truncating the original name as necessary. Alternatively, one might wish to replace the front characters with different ones (rather than "moving" the original name over). Below are examples of how to do these two variations.

* add 2 characters as a suffix (truncating existing name if necessary);

```
temp = min(length(oldname),6);
newname = substr(oldname,1,temp)||"&prefix";
call symput('name'||left(i),newname);
```

```
WEIGHT SALES PAYROLL DIVIDEND TAXPAID

  5     350    250       20        10

    SALES_W PAYROL_W DIVIDE_W TAXPAI_W

      1750    1250       100       50
```

* replace first 2 characters with particular prefix;
* first, extract 3rd and following characters;

```
newname = "&prefix"||substr(oldname,3);
call symput('name'||left(i),newname);
```

Example using original variables from a prior month of data ("PM"), updating them to reflect the current month ("CM"):

```
WEIGHT PMSALES PMPAYRL PMDIVDEN PMTAXPD
   5     350     250       20        10

       CMSALES CMPAYRL CMDIVDEN CMTAXPD
         1750    1250      100       50
```

In addition, one could expand the code to include the assignment of values to the new variables, adding an additional macro parameter that acts as a switch to activate one of several different transformations. For example, one might have %macro new(prefix,choice), where "choice" could be 1 for weighting the data, 2 for log-transforming the data, or 3 for taking the cube root of the data. Then, in a data step embedded in the macro, there would be additional code to perform these transformations depending on the value of "choice," e.g.,

```
new{i} = log(vars{i});
new{i} = sign(vars{i}) *
         (abs(vars{i}))**(1/3);
```

ACKNOWLEDGMENTS

I would like to thank Lydia Gorina for providing me with the original non-automated code and Thomas Bell for his comments on an earlier draft.