**Paper 112 – 2010**

# Using SAS® Macro Functions to Manipulate Data

Ben Cochran, The Bedford Group, Raleigh, NC

## Abstract

The SAS DATA step has the reputation for being one of the best data manipulators in the IT world. While the author of this paper agrees with this statement, it is possible to go beyond the capabilities of the DATA step by using SAS Macro functions. It would be difficult to show the full power of these Macro Functions in an hour presentation, so, this paper will look at a few commonly used Macro Functions and compare and contrast them to DATA step functions. These functions can be used not only to manipulate data, but to manipulate entire programs as well.

## Introduction

Macro functions are like DATA step functions except they operate only text strings and macro variables. This paper covers several examples of using different categories of macro functions.

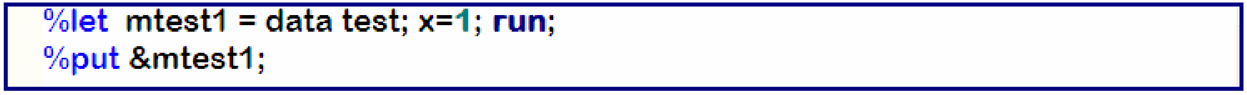
## Quoting Function Examples

Since the SAS Macro facility stores all values as a text string, there may be times when you want to:

- Store more than one statement a the value of a macro variable (see Try 1 below),
- Store quotation marks as the value of a macro variable (see Try 2 and Try 3 below)
- Maintain unresolved macro values in the output (see Try 4 below).

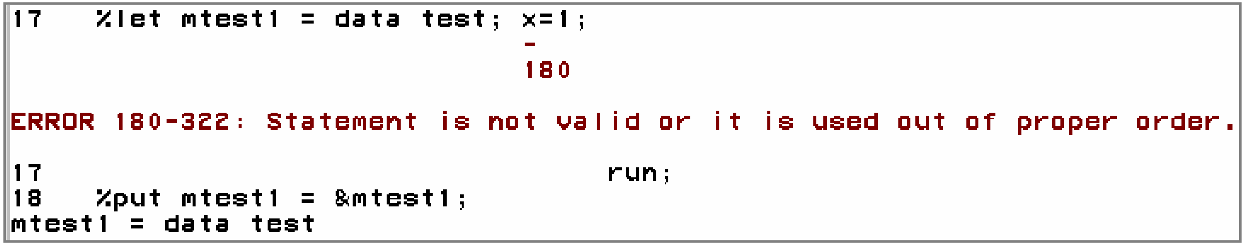
Suppose we use the %LET statement to store many statements in a macro variable. Then we will use a %PUT statement to see the results of the %LET.

**Try 1.**



```
%let mtest1 = data test; x=1; run;
%put &mtest1;
```

**Results of Try 1.**



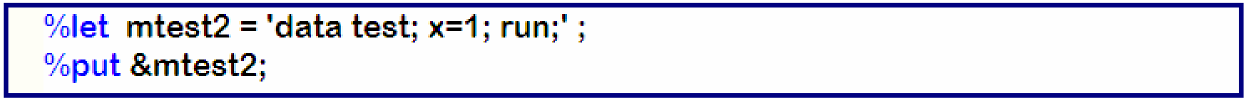
```
17   %let mtest1 = data test; x=1;
                              -
                              180

ERROR 180-322: Statement is not valid or it is used out of proper order.

17                                   run;
18   %put mtest1 = &mtest1;
mtest1 = data test
```

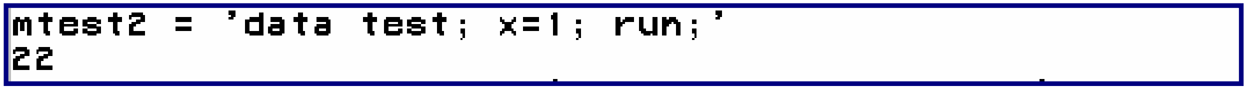
What caused the error? Notice the value of mtest1.

**Try 2.**



```
%let mtest2 = 'data test; x=1; run;' ;
%put &mtest2;
```

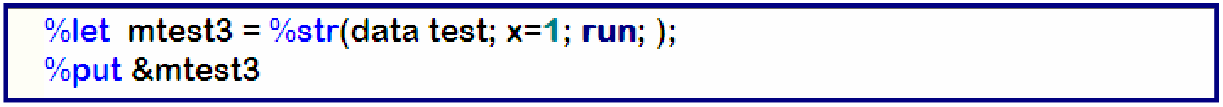
**Results of Try 2.**



```
mtest2 = 'data test; x=1; run;'
22
```

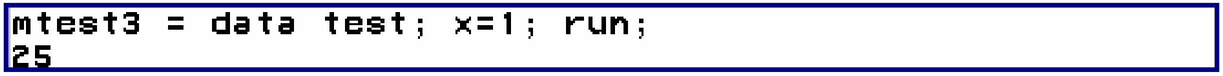
Notice the value of mtest2.   This works fine if we want quotation marks to be a part of the value.

**Try 3.**



```
%let  mtest3 = %str(data test; x=1; run; );
%put &mtest3
```

**Results of Try 3.**



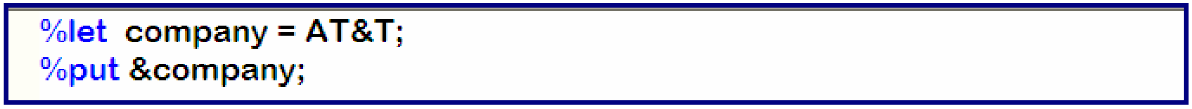
```
mtest3 = data test; x=1; run;
25
```

What is the difference between mtest2 and mtest3.

Using the **%STR** function allows you to let semicolons be part of the value of value of a macro variable.    In other words, it removes meaning from most special characters (like semicolons) at compile time.   It does not, however, remove the meaning from '&' and '%'  that are followed by non-blank 'tokens'.

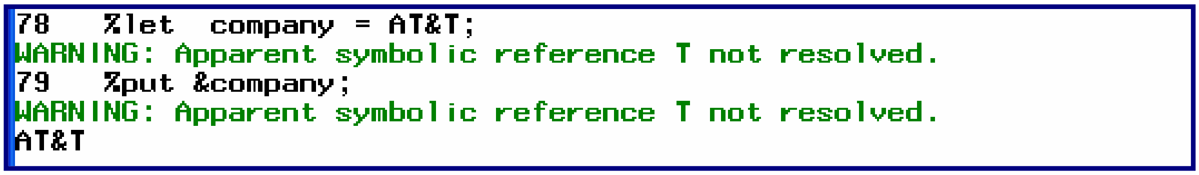
What if you needed to hide the effects of these macro triggers?  What if you needed to remove the meaning  from these '&' and '%' symbols followed by non-blank tokens.   Look's look at Try 4.  What if a macro trigger needs to be part of the value of a macro variable?

**Try 4.**



```
%let company = AT&T;
%put &company;
```

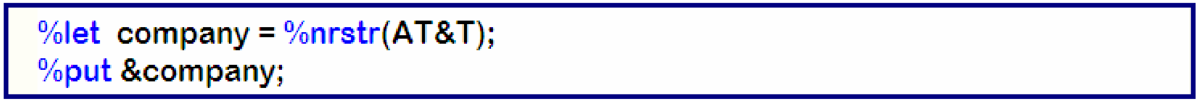
**Try 4 results.**



```
78   %let  company = AT&T;
WARNING: Apparent symbolic reference T not resolved.
79   %put &company;
WARNING: Apparent symbolic reference T not resolved.
AT&T
```

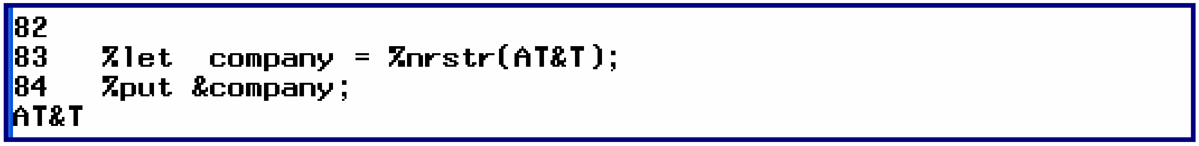
Using the **%NRSTR**  (No Resolve String)  function tells the macro processor to NOT resolve what appears to be a macro trigger.

**Try 5.**



```
%let company = %nrstr(AT&T);
%put &company;
```

**Try 5 results.**



```
82
83   %let  company = %nrstr(AT&T);
84   %put &company;
AT&T
```

## Character Handling Function Examples

These functions either change the value of text strings, or provide information about them.  Many of these text functions have a DATA step counterpart.  The only syntactical difference is they start with a '%'.  The program snippet below illustrates how **%SUBSTR,  %LENGTH, %LENGTH, %UPCASE,** and **%SCAN** work.

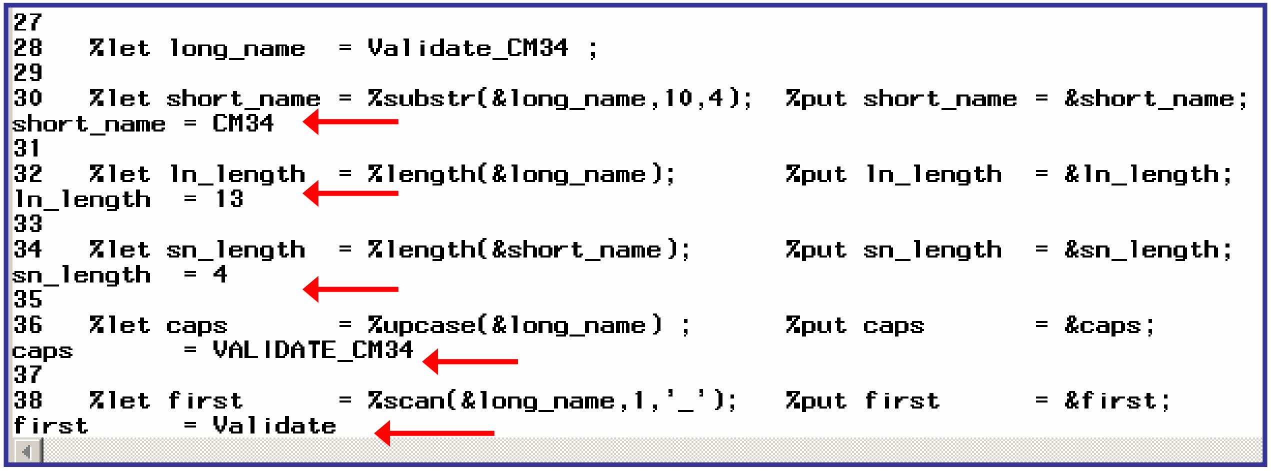
.

Suppose these macro statements were submitted for execution:

```
%let long_name        = Validate_CM34 ;
%let short_name       = %substr(&long_name,  10, 4);  %put short_name = &short_name;
%let ln_length        = %length(&long_name) ;         %put  ln_length    = &ln_length;
%let sn_length        = %length(&short_name) ;        %put   sn_length  = &sn_length;
%let caps             = %upcase(&long_name) ;         %put  caps          = &caps ;
%let first            = %scan(&long_name, 1, '_' ) ;  %put  first         = &first ;
```

Program Segment 1.

Examine the results in the log.



```
27
28    %let long_name   = Validate_CM34 ;
29
30    %let short_name = %substr(&long_name,10,4);  %put short_name  = &short_name;
short_name = CM34
31
32    %let ln_length   = %length(&long_name);        %put ln_length   = &ln_length;
ln_length   = 13
33
34    %let sn_length   = %length(&short_name);       %put sn_length   = &sn_length;
sn_length   = 4
35
36    %let caps        = %upcase(&long_name) ;       %put caps        = &caps;
caps         = VALIDATE_CM34
37
38    %let first       = %scan(&long_name,1,'_');    %put first       = &first;
first        = Validate
```

SAS Log 1.

## Character Handling Function Application

The Manager of the Dept. of Human Services needs to create a SAS Data set for very state represented in the CRIME data set. This needs to be a dynamic program because **not all states** may be included in the CRIME data set.

The tasks that need to be accomplished are:

1. Create a list that has one unique value for every state represented in the CRIME data set.

2. Since not all state names are valid SAS names (some have embedded blanks), create a second list that converts blanks into '_' to create valid SAS names.

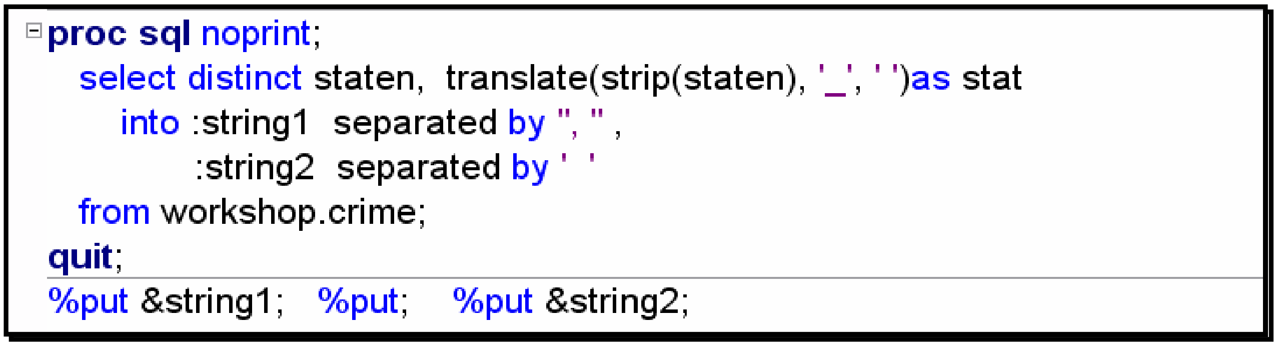
3. Create Macro variables named STRING1 and STRING2 that hold the values of these 2 lists.

4. Create a macro program that allows you to scan the lists and create a series of conditional statements for a subsequent DATA step. The series of IF-THEN statements need to look something like this:

   ex. If STATEN = 'New York' then output New_York ;

5. Write a DATA step that creates a Data set for each state, and correctly outputs the observations to the correct DATA set based on the values of the variable STATEN.

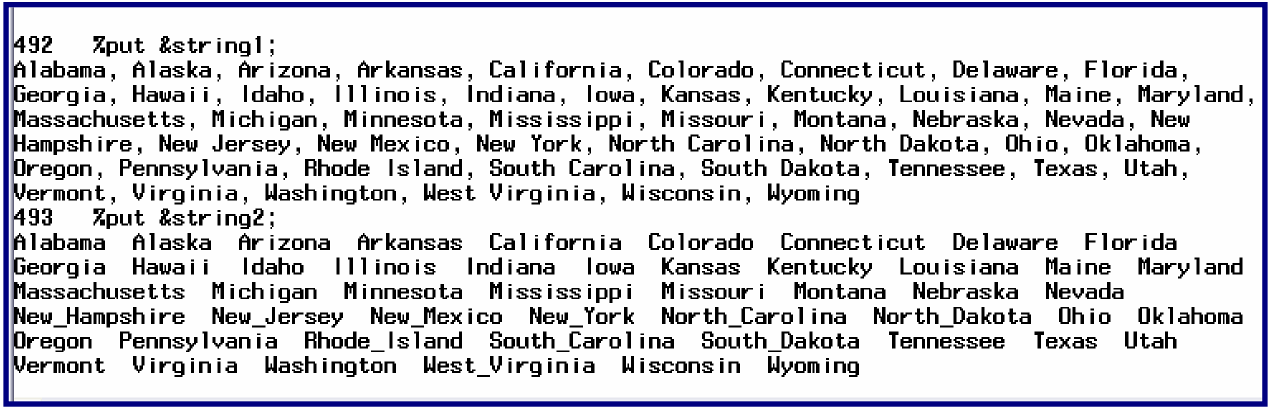
Steps 1, 2 and 3 are accomplished in the following PROC SQL step.



```
proc sql noprint;
   select distinct staten,  translate(strip(staten), '_', ' ')as stat
      into :string1  separated by ", " ,
           :string2  separated by ' '
   from workshop.crime;
quit;
%put &string1;   %put;   %put &string2;
```

Program Segment  2.

Examine the results in the log.



```
492   %put &string1;
Alabama, Alaska, Arizona, Arkansas, California, Colorado, Connecticut, Delaware, Florida,
Georgia, Hawaii, Idaho, Illinois, Indiana, Iowa, Kansas, Kentucky, Louisiana, Maine, Maryland,
Massachusetts, Michigan, Minnesota, Mississippi, Missouri, Montana, Nebraska, Nevada, New
Hampshire, New Jersey, New Mexico, New York, North Carolina, North Dakota, Ohio, Oklahoma,
Oregon, Pennsylvania, Rhode Island, South Carolina, South Dakota, Tennessee, Texas, Utah,
Vermont, Virginia, Washington, West Virginia, Wisconsin, Wyoming
493   %put &string2;
Alabama  Alaska  Arizona  Arkansas  California  Colorado  Connecticut  Delaware  Florida
Georgia  Hawaii  Idaho  Illinois  Indiana  Iowa  Kansas  Kentucky  Louisiana  Maine  Maryland
Massachusetts  Michigan  Minnesota  Mississippi  Missouri  Montana  Nebraska  Nevada
New_Hampshire  New_Jersey  New_Mexico  New_York  North_Carolina  North_Dakota  Ohio  Oklahoma
Oregon  Pennsylvania  Rhode_Island  South_Carolina  South_Dakota  Tennessee  Texas  Utah
Vermont  Virginia  Washington  West_Virginia  Wisconsin  Wyoming
```

SAS Log 2.

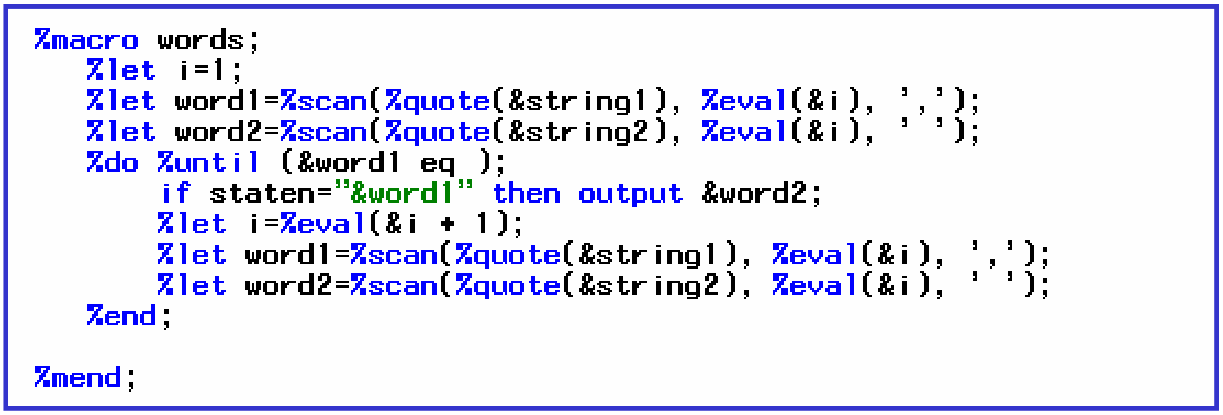
Notice the following:

1. &String1 values are separated by a comma and state names contain embedded blanks.
2. &String2 values are separated by blanks and state names have '_' instead of the blanks in &String1. This makes the state names valid SAS names.

We have now completed tasks 1 – 3.

**Task 4:** Create a **Macro Program** that allows you to scan the lists and create a series of conditional statements so that each observation will be output to the correct Data set. (This macro program will be embedded into a future DATA step) .

ex.   If staten = 'New Mexico' then output New_Mexico ;



```
%macro words;
   %let i=1;
   %let word1=%scan(%quote(&string1), %eval(&i), ',');
   %let word2=%scan(%quote(&string2), %eval(&i), ' ');
   %do %until (&word1 eq );
       if staten="&word1" then output &word2;
       %let i=%eval(&i + 1);
       %let word1=%scan(%quote(&string1), %eval(&i), ',');
       %let word2=%scan(%quote(&string2), %eval(&i), ' ');
   %end;

%mend;
```

Program Segment 3.

Note : **1.** The use of the **%SCAN** functions. This creates WORD1 and WORD2. WORD1 and WORD2 will contain

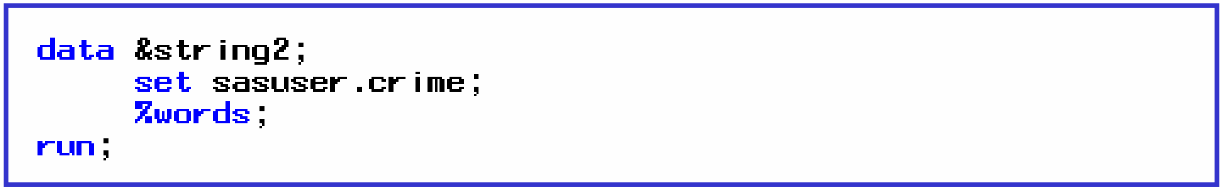
different spellings for the same state (One will have an embedded blank .

**2.** The IF / THEN statement. For example, when 'I' is equal to 32 this statement resolves to:

if staten = 'New York' then output New_York;

**3.** The **%EVAL** function evaluates integer arithmetic or logical expressions. **%EVAL** operates by converting its argument from a character value to a numeric or logical expression. Then, it performs the evaluation. Finally, **%EVAL** converts the result back to a character value and returns that value to the program.

**Task 5:** Write the DATA Step that uses the IF / THEN statement created by the macro **WORDS.**



```
data &string2;
     set sasuser.crime;
     %words;
run;
```

Program Segment 4.

Note: **1.** Tthe MACRO call **(%WORDS**) in the DATA Step**.**

**2.** The DATA Statement… it creates a SAS Data set for each state found originally in the CRIME data set.

## The %Sysfunc function

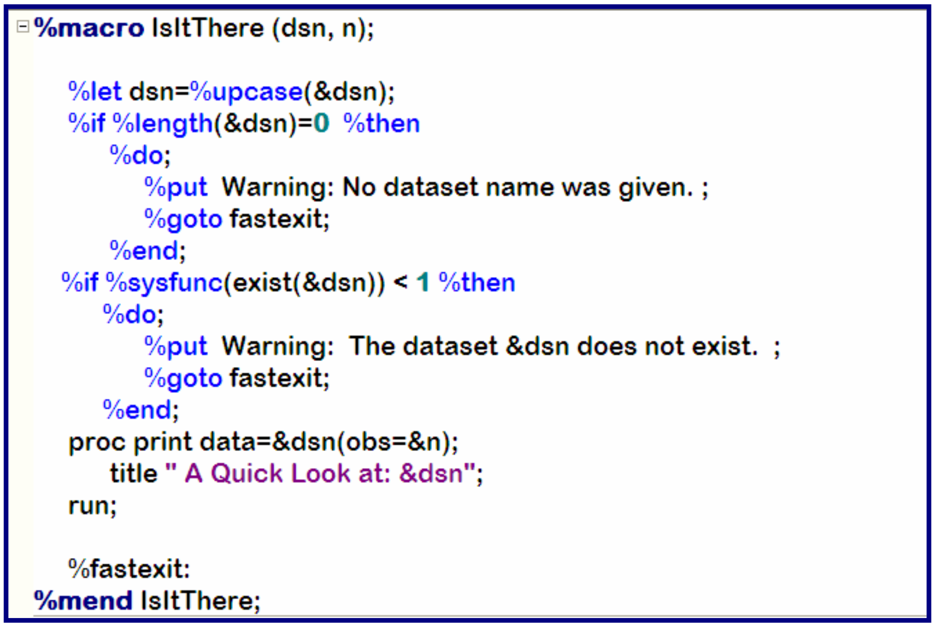
Before we go any further, we need an introduction to the %SYSFUNC function. This function brings some functionality to macros that was previously available only in the DATA step and the SCL area of the SAS system. The typical syntax is **:**

**%SYSFUNC (** ***function***( ***argument(s)*** **) <** ***format*** > )

The %SYSFUNC function **:**

- allows the user to execute functions that were previously unavailable in the macro facility **,**
- is especially useful in finding information about data sets **.**

Write a macro program that contains logic to see if a dataset exists**.**



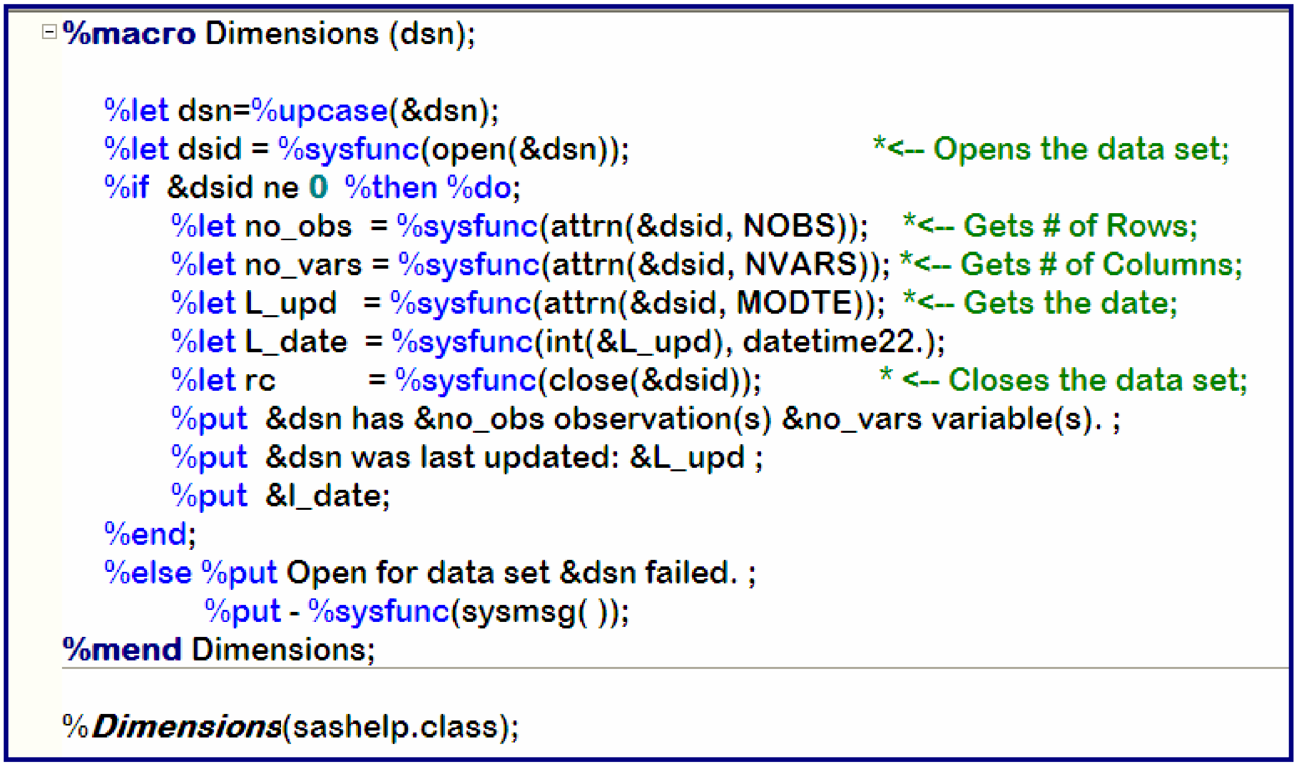
```
%macro IsItThere (dsn, n);

    %let dsn=%upcase(&dsn);
    %if %length(&dsn)=0  %then
        %do;
            %put  Warning: No dataset name was given. ;
            %goto fastexit;
        %end;
   %if %sysfunc(exist(&dsn)) < 1 %then
       %do;
            %put  Warning:  The dataset &dsn does not exist.  ;
            %goto fastexit;
       %end;
    proc print data=&dsn(obs=&n);
        title " A Quick Look at: &dsn";
    run;

    %fastexit:
%mend IsItThere;
```

Note :   **1.** The use of the **%UPCASE, %LENGTH** and **%SYSFUNC** functions in this macro program.
**2.** There is a colon ':' after **%fastexit** because it is a label in this program.

Suppose you need to know the number of **observations** and **variables** in data set. While you are at it, you might want to find out when the data set was last updated. You can use the **%SYSFUNC** function along with the **OPEN, CLOSE** and **ATTRN** functions to do this.

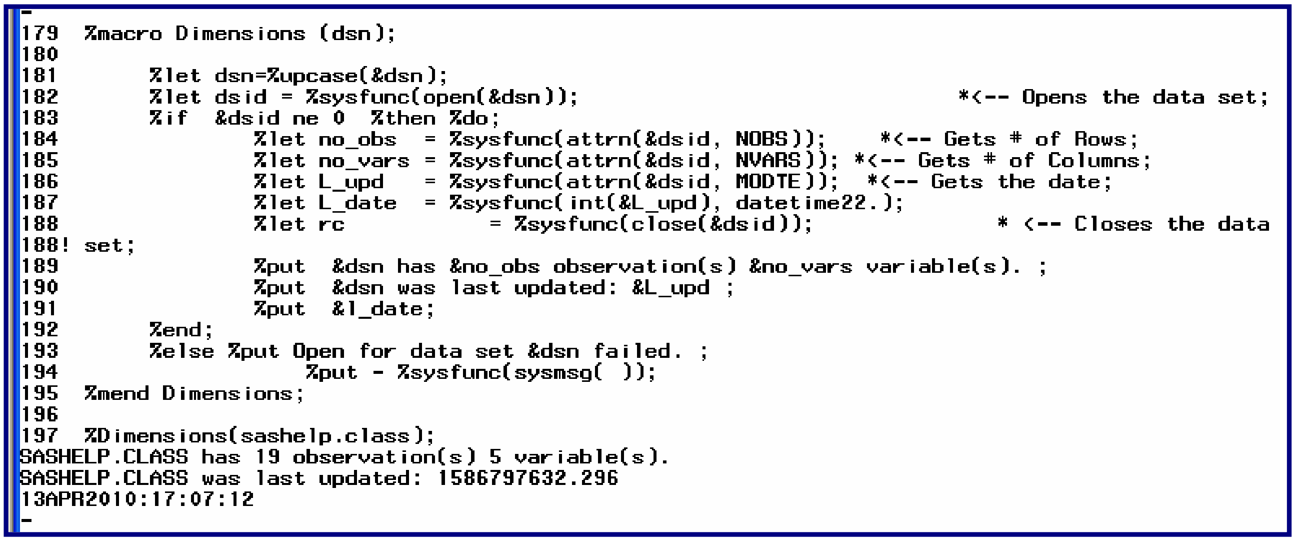


```
%macro Dimensions (dsn);

    %let dsn=%upcase(&dsn);
    %let dsid = %sysfunc(open(&dsn));                       *<-- Opens the data set;
    %if  &dsid ne 0  %then %do;
         %let no_obs  = %sysfunc(attrn(&dsid, NOBS));   *<-- Gets # of Rows;
         %let no_vars = %sysfunc(attrn(&dsid, NVARS)); *<-- Gets # of Columns;
         %let L_upd   = %sysfunc(attrn(&dsid, MODTE));  *<-- Gets the date;
         %let L_date  = %sysfunc(int(&L_upd), datetime22.);
         %let rc          = %sysfunc(close(&dsid));              * <-- Closes the data set;
         %put  &dsn has &no_obs observation(s) &no_vars variable(s). ;
         %put  &dsn was last updated: &L_upd ;
         %put  &l_date;
    %end;
    %else %put Open for data set &dsn failed. ;
              %put - %sysfunc(sysmsg( ));
%mend Dimensions;

%Dimensions(sashelp.class);
```

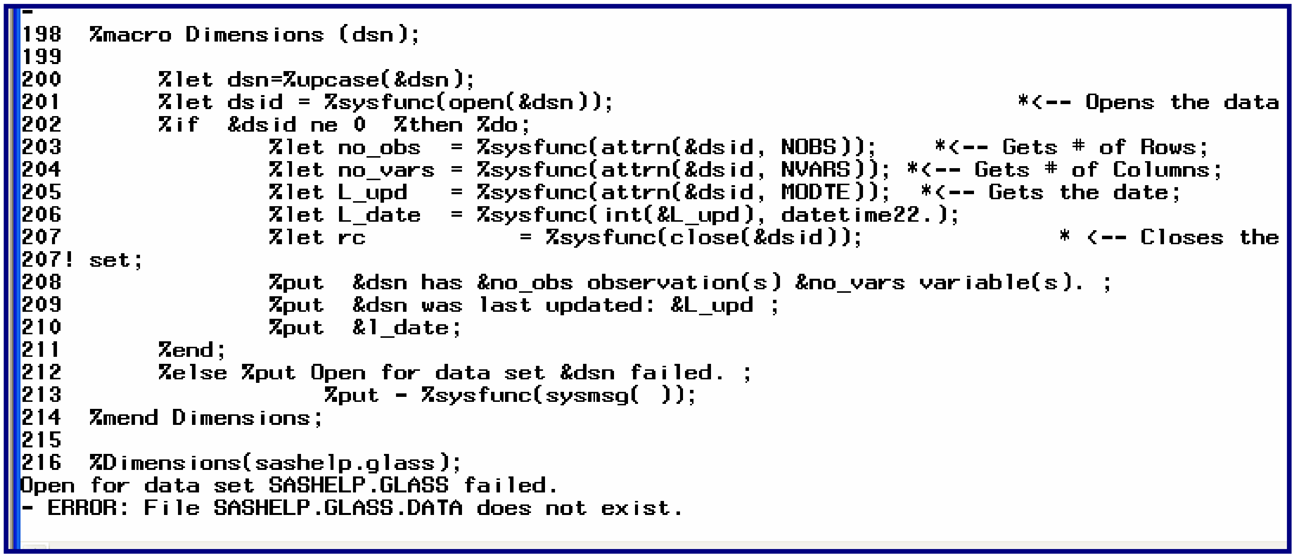
All the information is written to the log…



```
-
179  %macro Dimensions (dsn);
180
181       %let dsn=%upcase(&dsn);
182       %let dsid = %sysfunc(open(&dsn));                             *<-- Opens the data set;
183       %if  &dsid ne 0  %then %do;
184               %let no_obs  = %sysfunc(attrn(&dsid, NOBS));    *<-- Gets # of Rows;
185               %let no_vars = %sysfunc(attrn(&dsid, NVARS)); *<-- Gets # of Columns;
186               %let L_upd   = %sysfunc(attrn(&dsid, MODTE));  *<-- Gets the date;
187               %let L_date  = %sysfunc( int(&L_upd), datetime22.);
188               %let rc           = %sysfunc(close(&dsid));             * <-- Closes the data
188! set;
189               %put  &dsn has &no_obs observation(s) &no_vars variable(s). ;
190               %put  &dsn was last updated: &L_upd ;
191               %put  &l_date;
192       %end;
193       %else %put Open for data set &dsn failed. ;
194                   %put - %sysfunc(sysmsg( ));
195  %mend Dimensions;
196
197  %Dimensions(sashelp.class);
SASHELP.CLASS has 19 observation(s) 5 variable(s).
SASHELP.CLASS was last updated: 1586797632.296
13APR2010:17:07:12
-
```

Notice the results of the %PUT statements at the bottom of the log. Also notice the MODTE option returns the SAS Datetime value.

Now, let’s rerun the macro and pass the name of a nonexistent dataset (SASHELP.GLASS).



```
-
198  %macro Dimensions (dsn);
199
200       %let dsn=%upcase(&dsn);
201       %let dsid = %sysfunc(open(&dsn));                             *<-- Opens the data
202       %if  &dsid ne 0  %then %do;
203               %let no_obs  = %sysfunc(attrn(&dsid, NOBS));    *<-- Gets # of Rows;
204               %let no_vars = %sysfunc(attrn(&dsid, NVARS)); *<-- Gets # of Columns;
205               %let L_upd   = %sysfunc(attrn(&dsid, MODTE));  *<-- Gets the date;
206               %let L_date  = %sysfunc( int(&L_upd), datetime22.);
207               %let rc           = %sysfunc(close(&dsid));             * <-- Closes the
207! set;
208               %put  &dsn has &no_obs observation(s) &no_vars variable(s). ;
209               %put  &dsn was last updated: &L_upd ;
210               %put  &l_date;
211       %end;
212       %else %put Open for data set &dsn failed. ;
213                   %put - %sysfunc(sysmsg( ));
214  %mend Dimensions;
215
216  %Dimensions(sashelp.glass);
Open for data set SASHELP.GLASS failed.
- ERROR: File SASHELP.GLASS.DATA does not exist.
```

Notice the result of the SYSMSG function nested within the %SYSFUNC function.

## Conclusion

Macro functions give the DATA step tremendous power in manipulating data as well as controlling the program flow.

It is difficult to do the topic justice in a short ( one hour ) presentation and a fairly short proceedings paper. Hopefully,  you appetite for more SAS Macro knowledge whetted. There are numerous books on this topic in the books by user section of most SAS User gatherings. The Education division of SAS Institute has some very good courses on this topic as well.

## Acknowledgments

I would like to acknowledge and greatly thank the Technical Support Department at SAS Institute for their helpful knowledge and expertise that they so freely gave. I would also like to thank Art Carpenter for his SAS Macro knowledge. Whenever I have a macro question he always gives me a great answer.

## Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Ben Cochran
The Bedford Group
Raleigh, NC 27607
Phone: 919.741.0370
Email: bedfordgroup@nc.rr.com