Using SAS® Macro Functions
to
Manipulate Data

By
Ben Cochran
The Bedford Group
bencochran@nc.rr.com
919.741.0370

Table of Contents

1. Handling Numeric Values

Since the macro facility stores all values as **text strings,** there may be times when you want to **:**

- **add** two or more text strings together to get a number.

Suppose you want to get the FIRST word from a string. Examine **TRY1.**

```
Log - (Untitled)
192  %macro try1;
193                  %let sentence = This is a test;
194                  %let i = 1;
195                  %let word=%scan(&sentence, &i);
196                  %put Word &i.. is &word;
197  %mend;
198
199  %try1;
Word 1. is This
```

The macro variable **I** resolves to **1** and the first word from the text string resolves to **This.**

more →

Handling Numeric Values

Suppose you want to get the **FOURTH** word from a string. Examine TRY2.

```
200
201  %macro try2;
202                  %let sentence = This is a test;
203                  %let i = 1;
204                  %let j = &i + 3;
205                  %let word=%scan(&sentence, &j);
206                  %put Word &j.. is &word;
207  %mend;
208
209  %try2;
Word 1 + 3. is test
```

The macro variable **J** resolves to the expression **1 + 3 .**
Since expressions can be used as arguments in Functions, the **%SCAN** returns the fourth word **test .**

However, what if we wanted the last PUT statement to yield**:**

Word 4. is test

more →

Handling Numeric Values

Use the **%EVAL** function to resolve the expression **1 + 3** so that it equals **4.**

```
220
221  %macro try3;
222                  %let sentence = This is a test;
223                  %let i = 1;
224                  %let j = %eval(&i + 3);
225                  %let word=%scan(&sentence, &j);
226                  %put Word &j.. is &word;
227  %mend;
228
229  %try3;
Word 4. is test
```

The macro variable **J** now resolves to the number **4 .**

The **%SCAN** function still returns the fourth word **test .**

more →

2. Quoting in the Macro Facility

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

- store more than one statement as the value of a macro variable **- Try 1** & **Try 2 ,**
- store quotation marks as the value of a macro variable - **Try 2** and **Try 3 ,**
- maintain unresolved values in the output - **Try 4 .**

Suppose you want to use the %LET statement to store many statements in a macro variable **.** Use a %PUT statement to see the results of the %LET**.**

Try 1.

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

Results . . .

```
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.**

more →

Quoting in the Macro Facility

Try 2. - one solution . . .

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

Results . . .

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

Notice the value of mtest2.

Try 3. - another solution . . .

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

Results . . .

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

What is the difference between mtest2 and mtest3 **?**

try 4 →

Quoting in the Macro Facility

Try 4. - What if a macro trigger needs to be part a value of a macro variable?

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

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
```

Try 5. - use the **%NRSTR** if you do **N**OT want the value **R**esolved **.**

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

Results . . .

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

more quoting →

Quoting in the Macro Facility

Try 6. - Use the **%QUOTE** function to **mask** special characters and mnemonic operators in a resolved value at macro execution.

```
%macro check1(state);
   %if &state=nc %then
        %put The State %upcase(&state) is in the East;
        %else %put The State %upcase(&state) is in the West;
%mend check1;
%check1(or);

%macro check2(state);
   %if %quote(&state)=nc %then
          %put The State %upcase(&state) is in the East;
          %else %put The State %upcase(&state) is in the West;
%mend check2;
%check2(or);
```

SAS Log →

Quoting in the Macro Facility

Try 6. - Log. Passing a value that is the same as an **operator** can cause problems, but using the %QUOTE function can eliminate them.

```
153  %macro check1(state);
154     %if &state=nc %then
155          %put The State %upcase(&state) is in the East;
156          %else %put The State %upcase(&state) is in the West;
157  %mend check1;
158  %check1(or);
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric
       operand is required. The condition was: &state=nc
ERROR: The macro CHECK1 will stop executing.
159
160  %macro check2(state);
161     %if %quote(&state)=nc %then
162          %put The State %upcase(&state) is in the East;
163             %else %put The State %upcase(&state) is in the West;
164  %mend check2;
165  %check2(or);
The State OR is in the West
```

hows →

Workshop - 1

1. Submit the following statements to show how the **%EVAL** function works. What is the value of X, Y, or Z in each case?

1a. `%put x = 11 + 4;` _____

b. `%put x = %eval(11 + 4) ;` _____

c. `%put x = %eval( 4 lt 11);` _____

d. `%put x = %eval( 4 gt 11);` _____

e.
```
%let x = 2;
%let y = %eval( &x + 2);
%put y = &y;
```
_____

f. `%let z = %eval(&y / &x);` _____

g.
```
%let z = %eval(&y * &x);
%put z = &z;
```
_____

Workshop

2. Submit the following statements to show how the quotes work in macro processing. Pay close attention to the quotation marks. How do the PROC PRINT steps differ?

```
%let mac_var_f = F;
%let mac_var_m = "M";

title "Report for Gender: &mac_var_f ";
proc print data=sashelp.class;
      where sex = "&mac_var_f ";
run;

title "Report for Gender:" &mac_var_m;
proc print data=sashelp.class;
      where sex = &mac_var_m;
run;
```

For greater flexibility it is a good idea NOT to store quotes as part of a macro variables value.

3. Character Handling Functions

Submit the following statements to show how these commonly used macro functions work.…

```
%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 ;
```


Character Handling Functions

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
```

The **%SYSFUNC** Function

Before looking at the next set of examples, we need an introduction to the %SYSFUNC function.

This feature 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* >)

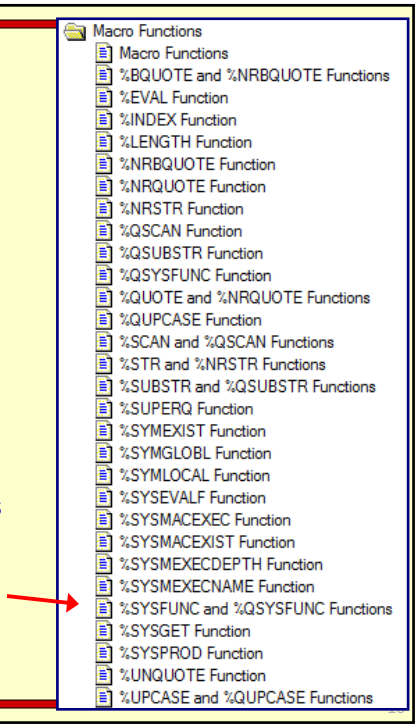
The %SYSFUNC function **:**

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

The **%SYSFUNC** Function

Here is a list of **all** the Macro functions in **SAS9.3.** However, using the **SYSFUNC** function allows you to harness the power of most/all of the functions available in the DATA Step.

Can you think of some DATA step functions that are not displayed on this page?



The %SYSFUNC Function

Use the **%SYSFUNC** function to display a SAS date in the SAS Log.

```
181  proc sql noprint;
182     select max(sales_date),  count(*)
183          into : cutoff, : r_count
184     from sas_3.sales_header;
185  quit;
NOTE: PROCEDURE SQL used:
      real time             0.60 seconds
      cpu time              0.59 seconds


186  %put Note: cutoff=&cutoff   is %sysfunc(int(&cutoff), date9.).;
Note: cutoff=   14608   is 30DEC1999.
187  %put Note: r_count=%sysfunc(int(&r_count), comma12.).;
Note: r_count=   1,279,147.
```

The latest date (30DEC1999) is stored in a Macro variable named **&cutoff**.
The number of records (1,279,147) is stored in a macro variable named **&r_count**.

4. Conditional Processing

Write a macro program with logic to check for the value of the data set name.

```
%macro qwkpeek (dsn, obs=5);

    %let dsn=%upcase(&dsn);
    %if %length(&dsn)=0 %then
    %do;
        %put
        Warning: No dataset name was given. Request Cancelled.;
        %goto quitnow;
    %end;
    proc print data=&dsn(obs = &obs);
       title "A Quick Peek at: &dsn";
    run;

    %quitnow:

%mend qwkpeek;

%qwkpeek;
```

Examine the logic of the macro program **.**

Note: the macro call **does not** pass a data set name. What value will &dsn have **?**

log →

Conditional Processing

Viewing the SAS Log verifies the program worked correctly .

```
245
246    %macro qwkpeek (dsn, obs=5);
247
248        %let dsn=%upcase(&dsn);
249        %if %length(&dsn)=0 %then
250        %do;
251            %put
252            Warning: No dataset name was given. Request Cancelled.;
253            %goto quitnow;
254        %end;
255        proc print data=&dsn(obs = &obs);
256           title "A Quick Peek at: &dsn";
257        run;
258
259        %quitnow:
260
261    %mend qwkpeek;
262
263    %qwkpeek;
Warning: No dataset name was given. Request Cancelled.
```

Consequently, nothing is written to the output window .

What if a value for &dsn is passed, but the data set does not exist ?

%sysfunc →

Conditional Processing

Modify the program to add logic to check for the existence of a data set.

Examine the added logic of the macro program . Notice the macro call.

```
%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;

%IsItThere(sashelp.retailx, 5);
```

log →

Conditional Processing

Viewing the log verifies the program worked correctly.

Log - (Untitled)

```
22   %macro IsItThere (dsn, n);
23
24         %let dsn=%upcase(&dsn);
25         %if %length(&dsn)=0  %then
26               %do;
27                     %put  Warning: No dataset name was given. ;
28                     %goto fastexit;
29               %end;
30        %if %sysfunc(exist(&dsn)) < 1 %then
31               %do;
32                     %put  Warning:  The dataset &dsn does not exist.  ;
33                     %goto fastexit;
34               %end;
35         proc print data=&dsn(obs=&n);
36                title " A Quick Look at: &dsn";
37         run;
38
39         %fastexit:
40   %mend IsItThere;
41
42   %IsItThere(sashelp.retailx, 5);
Warning:  The dataset SASHELP.RETAILX does not exist.
43
```

more →

Conditional Processing

Suppose you need to know the **number of observations** and **variables** in data set. Use the %SYSFUNC function to do this.

There have been many solutions generated in the past to obtain this information from SAS data sets. Typically, previous solutions have used a combination of **:**

- DATA _NULL_ steps **,**
- SET statements with the NOBS= option **,**
- arrays **,**

… to obtain this information**.**

You can also use the **:**

- OPEN function **,**
- ATTRN function **,**
- along with the %SYSFUNC function **.**

pgm →

Conditional Processing

```
%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);
```

Notice :
- OPEN, ATTRN, and CLOSE functions, SYSMSG function,
- along with the %SYSFUNC function,
- the data set used in the macro call.

log →

Conditional Processing

Viewing the log verifies the program worked correctly.

```
179  %macro Dimensions (dsn);
180
181       %let dsn=%upcase(&dsn);
182       %let dsid = %sysfunc(open(&dsn));                              *<-- Opens the data
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
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
```

Next, pass to this macro a data set that does not exist .

more →

Conditional Processing

Notice the macro call: **%Dimensions(sashelp.glass);**

Viewing the log verifies the program worked correctly.

```
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.
```

5. →

Workshop - 2

3. Write a macro program that prints the first 5 rows of a SAS dataset whose name will be passed to the macro as a parameter. Include code to see if the dataset exists before you print the report. Also include customized Log messages. When invoking the macro, pass SASHELP.SHOES as the parameter.

   Accomplish this task by modifying the program on the right. Change the 'x's to valid values.

```
%macro  print_it ( dsn );

   %if  %sysfunc(exist(&dsn)) < 1 %then
        %do;
            %put  Warning:  xxxxxxxx.  ;
            %goto  exitnow;
        %end;

        proc print data=&xxxxx (obs = x);
           title "First x obs of &dsn";
        run;

        %exitnow:

%mend printit;

%print_it(xxxxxx);
```

→

5.0 DATA Step / SQL Interfaces

5.1 Creating Macro Variables that contain Lists

5.2 Populating Spreadsheets

5.3 Exercises

5.1 Creating Macro Variables that Contain Lists

A retail store chain needs to create a separate SAS dataset for each store found in their catalog sales dataset. The first challenge is that not all stores are represented in the catalog sales dataset. The second challenge is that the stores are in the dataset by their numeric code and because we want to create a separate SAS dataset for each store, we will need to create not only a list of stores, but also a list of valid SAS names for each store. Notice the **store** variable.

VIEWTABLE: Sas_3.Catalog_sales

	Store	City Name	State	invoice	Account Number	sales_year	sales_mon	sales_day	Sale Amount
1	050	Charlotte	NC	4309501010064	A4300064100	2005	1	1	177.64
2	050	Charlotte	NC	4309501010064	A4300064100	2005	1	1	123.80
3	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	779.55
4	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	30.57
5	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	124.96
6	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	229.63
7	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	377.50
8	050	Charlotte	NC	4309501015567	A4305567700	2005	1	1	71.83
9	050	Charlotte	NC	430950101D100	A430D100500	2005	1	1	87.85
10	050	Charlotte	NC	730950101G879	A730G879300	2005	1	1	2.35
11	050	Charlotte	NC	9709501014589	A0004589430	2005	1	1	0.92

Creating Macro Variables That Contain Lists

Step 1. Create two lists of Store values.

One list will be the store numbers themselves, and the other list will be the store numbers preceded by the letter **'S'** so that they are valid SAS names**.**

```
proc sql noprint;
   select distinct store,  'S'!!trim(store) as dsn
   into : storelist separated by ' ',
        : dsnlist   separated by ' '
   from sas_3.catalog_sales(obs=max);
quit;
%put &storelist;
%put &dsnlist;
```

Both lists have values that are separated by blanks. Examine the log to verify the values of **&storelist** and **&dsnlist.**

```
16   %put &storelist;
050  100  1000  150  200  250  300  350  400  450  500  550  600  650  700  750  800
17   %put &dsnlist;
S050  S100  S1000  S150  S200  S250  S300  S350  S400  S450  S500  S550  S600  S650
S800  S850  S900  S950
```


Parse The Lists

Step 2. Create a macro program to parse each list.

The next step is to produce a dynamic IF THEN statement that will look like the one below when it resolves and processes the first store value in the list**.**

if store = **"050"** then output **S050;**

```
%macro parse;
   %let i = 1;
   %do %until (&Store  eq  );
      %let store=%scan(%quote(&storelist), %eval(&i), ' ');
      %let dsn  =%scan(%quote(&dsnlist),   %eval(&i), ' ');
      if store = "&store" then output &dsn;
      %let i = %eval(&i + 1);
   %end;
%mend;
```

Notice the **%LET** statements. They are creating a value for **&STORE** and a value for **&DSN.** The DO Loop will execute once for each store value. The IF THEN statement will put the correct store value in the correct dataset.

Parse The Lists

Step 3. Write a DATA step to parse the data:

First, notice **&dsnlist.** It's a list of all the Store_ID numbers preceded with the letter 'S' that are found in the data. Because it is on the DATA statement, it will create a SAS dataset for every store.

Also notice that datasets are generated from first quarter data (where sales_mon between 1 and 3) and unwanted variables are being dropped**.**

```
data &dsnlist;
    set sas_3.catalog_sales(obs=max drop=x y z store_id);
    where sales_mon between 1 and 3;
    %parse;
run;
```

The **%PARSE** calls the macro program (created in Step 2) that generates an IF THEN statement that puts the correct store value in the correct dataset.

Parse The Lists

Step 3. continued:

The SAS Log shows that the program created several different datasets, one for each store.

Log - (Untitled)

```
NOTE: There were 288595 observations read from the data set SAS_3.CATALOG_SALES.
      WHERE (sales_mon>=1 and sales_mon<=3);
NOTE: The data set WORK.S050 has 25162 observations and 15 variables.
NOTE: The data set WORK.S100 has 24441 observations and 15 variables.
NOTE: The data set WORK.S1000 has 18356 observations and 15 variables.
NOTE: The data set WORK.S150 has 20956 observations and 15 variables.
NOTE: The data set WORK.S200 has 26576 observations and 15 variables.
NOTE: The data set WORK.S250 has 12363 observations and 15 variables.
NOTE: The data set WORK.S300 has 8847 observations and 15 variables.
NOTE: The data set WORK.S350 has 21838 observations and 15 variables.
NOTE: The data set WORK.S400 has 16285 observations and 15 variables.
NOTE: The data set WORK.S450 has 11271 observations and 15 variables.
NOTE: The data set WORK.S500 has 7630 observations and 15 variables.
NOTE: The data set WORK.S550 has 10028 observations and 15 variables.
NOTE: The data set WORK.S600 has 9908 observations and 15 variables.
NOTE: The data set WORK.S650 has 6952 observations and 15 variables.
NOTE: The data set WORK.S700 has 17488 observations and 15 variables.
NOTE: The data set WORK.S750 has 13525 observations and 15 variables.
NOTE: The data set WORK.S800 has 7604 observations and 15 variables.
NOTE: The data set WORK.S850 has 9105 observations and 15 variables.
NOTE: The data set WORK.S900 has 9265 observations and 15 variables.
NOTE: The data set WORK.S950 has 10995 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           1.96 seconds
      cpu time            1.27 seconds
```

5.2 Populating Spreadsheets (Optional)

The next few steps involves writing data from SAS datasets to spreadsheets.

Step 4. Write a PROC EXPORT step.

This step tests the PROC EXPORT step to make sure that it works by taking one SAS dataset and writing it to a spreadsheet.

```
proc export data = work.S050
        outfile = "c:\S050.xls"
        dbms = xls replace;
run;
```

The log shows that the step was successful.

```
62
63        proc export data = work.S050
64                    outfile = "c:\S050.xls"
65                    dbms = xls  replace;
66        run;

NOTE: The export data set has 25162 observations and 15 variables.
NOTE: "c:\S050.xls" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time             1.23 seconds
      cpu time              0.06 seconds
```

33

Populating Spreadsheets (Optional)

Step 5. Convert the PROC EXPORT code into a Macro program.

Create the macro and call it passing ‘s100’ as the value for the DSN parameter. The SAS log shows the macro program was successful.

```
%macro export(dsn) ;
    proc export data = work.&dsn
        outfile = "c:\&dsn..xls"
        dbms = xls replace;
    run;
%mend;

%export(dsn = s100);
```

```
41
42        %macro export(dsn) ;
43                proc export data = work.&dsn
44                        outfile = "c:\&dsn..xls"
45                        dbms = xls  replace;
46                run;
47        %mend;
48
49        %export(dsn = s100);

NOTE: The export data set has 24441 observations and 15 variables.
NOTE: "c:\s100.xls" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time             0.32 seconds
      cpu time              0.06 seconds
```

34

Populating Spreadsheets (Optional)

Step 6. Create a DATA set that contains a list of Stores.

This DATA step reads the **&dsnlist** macro variable and writes out an observation for each value in the list.

```
data dsn(drop=i);
   length dsn $ 10;
   i = 1;
   do until (dsn eq ' ');
      dsn=scan("&dsnlist", i, ' ');
      if dsn ne ' ' then output;
      i + 1;
    end;
run;
```

VIEWTABLE: Work.Dsn

	dsn
1	S050
2	S100
3	S1000
4	S150
5	S200
6	S250
7	S300
8	S350
9	S400
10	S450
11	S500
12	S550
13	S600
14	S650
15	S700
16	S750
17	S800
18	S850
19	S900
20	S950

The **CALL Execute** Routine

Step 7. Write a DATA step that uses a CALL EXECUTE routine to process the EXPORT macro.

This DATA step reads the **dsn** dataset and uses the values of **dsn** as a value to pass into the EXPORT macro.

```
data _null_;
  set dsn(obs=max);
  call execute('%export(dsn = ' !! dsn !! ' ) ' );
run;
```

Log - (Untitled)

```
68   data  _null_;
69           set dsn(obs=max);
70           call execute('%export(dsn = ' !! dsn !! ' ) '  );
71       run;

NOTE: There were 20 observations read from the data set WORK.DSN.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


NOTE: CALL EXECUTE generated line.
1   + proc export data = work.S050                          outfile = "c:\S050.xls"
    dbms = xls  replace;
1   +
              run;

NOTE: The export data set has 25162 observations and 15 variables.
NOTE: "c:\S050.xls" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.49 seconds
      cpu time            0.07 seconds
```

Macro Options

Re-run the DATA step to illustrate the effects of the **MLOGIC** and **MPRINT** options**.**

Log - (Untitled)

```
72   options mlogic  mprint;
73        data  _null_;
74            set dsn(obs=1);
75            call execute('%export(dsn = ' !! dsn !! ' ) '  );
76        run;

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds

MLOGIC(EXPORT):  Beginning execution.
MLOGIC(EXPORT):  Parameter DSN has value S050
MPRINT(EXPORT):   proc export data = work.S050 outfile = "c:\S050.xls" dbms = xls replace;
MPRINT(EXPORT):   run;
MLOGIC(EXPORT):  Ending execution.
NOTE: There were 1 observations read from the data set WORK.DSN.

NOTE: CALL EXECUTE generated line.
1    + proc export data = work.S050                        outfile = "c:\S050.xls"
     dbms = xls  replace;
1    +
               run;

NOTE: The export data set has 25162 observations and 15 variables.
NOTE: "c:\S050.xls" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.41 seconds
      cpu time            0.07 seconds
```

The End

About the Speaker

Speaker	Ben Cochran
Location of company	The Bedford Group 3224 Bedford Ave. Raleigh, NC 27607
Telephone	(919) 741.0370
E-Mail	bencochran@nc.rr.com