Paper 113-2007

List Processing Basics: Creating and Using Lists of Macro Variables

Ronald J. Fehd, Centers for Disease Control and Prevention, Atlanta, GA, USA
Art Carpenter, CA Occidental Consultants, Vista, CA, USA

ABSTRACT

List Processing or dynamic programming techniques free up your programming time by eliminating hard coded parameter values. List values can be obtained directly from information you already have, and there is more than one way to do it! Commonly the list is stored as series of macro variables or as a macro variable with a series of values or even as values in a data set.

This paper compares and contrasts the basic methods of treating variable values as parameters and placing them into repetitive code.

Audience: intermediate users.

Keywords: call execute, call symput, dynamic programming, into:, list processing, self-modifying

Contents

INTRODUCTION

One of the greatest strengths of the SAS® macro language is its ability to build and execute repetitive code. The level of repetition that encourages conversion to a macro depends on the tolerance of the programmer. For Ron and Art a program with a repetition of more than two items is a candidate for conversion.

As we write more sophisticated macros we want to have the macro determine as much of the information that it needs as possible. In a simple case consider the macro `%do` loop:

```
%macro DoLoop;
%do I = 1 %to 5;
    %*...;
```

This loop has a hard coded number of repetitions (5). We can generalize by passing the number of repetitions as a parameter:

```
%macro DoLoop(Count);
%do I = 1 %to &Count;
    %*...;
```

In a dynamic coding solution the macro will include code so that the number of repetitions can be determined by the macro itself.

```
%macro DoLoop;
%*... code determines Count;
%do I = 1 %to &Count;
    %*...;
```

The final solution is one kind of dynamic process, and in the following sections of this paper we will present several different approaches and techniques that are commonly used to solve dynamic coding problems.

In a dynamic programming situation it is very often desirable to process one or more operations for a series of items. The items themselves may be data sets, variables in a data set, (columns in a table), or values of a column within a table.

The hallmark of dynamic programming techniques is that the program itself will determine what the list is, build it,

and then use it. In a dynamic program all this is done without programmer intervention.

Consider the following simple `Proc Report` step which routes the output to an `html` file.

See Ex01.

Report-Africa.sas

```
1  ods html file  = "&path.Africa.html"
2           style = journal;
3
4  Title "Sales in Africa";
5
6  PROC Report data = sashelp.Shoes
7                     nowindows;
8              where  region = 'Africa';
9              column product subsidiary, sales;
10             define product    / group;
11             define subsidiary / across;
12             define sales      / analysis '';
13 run;
14 ods html close;
```

If we want a similar report for each region we cannot simply use a `by` statement in the PROC step because we want the `Title` and the `html` filename to contain the name of the region. Hard coding the value of one region while testing is no problem, for two it is irritating, for three it is time to find a better solution. We need a dynamic solution that can determine the number of regions, the values of each, and then execute this step for each one.

In the following sections we examine this problem from several angles and we show several different techniques that can be used to build the ability to provide the needed repetition.

The examples show the concepts of testing the routine as it is being developed.

- prepare routine
  - identify program parameter(s)
  - convert program to routine
- make list of parameter values
- iterate with each value: call routine

PREPARING A ROUTINE

IDENTIFY PROGRAM PARAMETERS

Our report has three occurences of the string *Africa*. This is the first value of the variable Region in the data set sashelp.Shoes.

See Ex01.

```
1  ...     file = 'Africa.html'
2  Title 'Sales in Africa';
3  where Region = 'Africa';
```

CONVERT TO ROUTINE

The first step in converting our program to a routine is to keep the program running by making it into a parameterized include file. Add one macro variable for each parameter and replace the occurences of each parameter value with the macro variable.

Find and replace the three occurences of the string *Africa* with the macro variable reference *&Region.*. Note that the prefix to the macro variable reference is the special character ampersand (`&`) and the suffix is the character dot (`.`). In the filename-extension the first dot ends the macro variable reference and the second dot separates the filename and the extension.

```
1  %Let              Region=Africa;
2  ...     file = "&Region..html"
3  Title "Sales in &Region.";
4  where Region = "&Region.";
```

CONVERT TO MACRO

Finally change the allocation of the macro variable from `%Let` to `%macro` parameter.

```
%macro Report(   Region=Africa);
...      file = "&Region..html"
Title "Sales in &Region.";
where Region = "&Region.";
```

The next task is to identify the list of parameter values and decide how best to generate the macro calls shown here in the test section at the bottom of the program.

Note lines 5–9: Since one value contains a slash (/) we create a new variable `Filename` and change the slash to a hyphen to avoid an operating system clash of folder `Central America/Caribbean` not existing.

Report-Region-macro.sas

```
1  %Macro Report(InData  = sashelp.Shoes
2               ,Region  =
3               ,OutPath = /*here: same folder*/
4               );
5  %local Filename; %Let Filename = &Region.;
6  %** change slash to hypen in
7      Central America/Caribbean;
8  %If %index(&Region,/) %then %Let Filename =
9      %sysfunc(translate(&Region.,-,/));
10
11 ods html file  = "&OutPath.&Filename..html"
12          style = journal;
13
14 Title "Sales in &Region.";
15
16 PROC Report data = &InData.
17                  nowindows;
18             where  Region = "&Region.";
19             column Product Subsidiary, Sales;
20             define Product    / group;
21             define Subsidiary / across;
22             define Sales      / analysis '';
23 run;
24 ods  html close;
25 run; %mend Report;
```

Task: How to generate these macro calls?

Report-Region-macro.sas

```
28 %Report(Region = Africa);
29 %Report(Region = Asia  );
30 %Report(Region
31 = Central America/Caribbean);
```

See Ex00.

To speed up our testing, we write a very simple test macro which only lists its parameter values.

ReportTest.sas

```
1  %Macro ReportTest(Region = );
2  %put _local_;
3  run; %Mend;
```

MAKING LISTS

WHAT IS A LIST?

In conversation we use the word *list* to mean any collection of *items*, however they may be related. An example would be a todo or shopping list. The items in the list share a common property: tasks to be done, items to purchase. In a formal language, such as set theory, a list is defined as an unordered set. In the SAS language a list is a table. For our discussion here, the list is a collection of the unique values of the variable region of the table sashelp.shoes.

Definition of List

1. natural language: list contains items
2. formal language: an unordered set
3. SAS: data set or table

WHERE ARE LISTS?

SAS provides many unique data sets in either sashelp views or sql dictionary tables. DiIorio and Abolafia [5, sugi29.237] provide a comparision of sashelp views and sql dictionary tables and a dozen examples. Most summarization procedures — freq and means/summary — can produce an output data set which contains a unique list of variable values. Fehd [8, pnwsug2006.012] provides usage examples of the more common sql dictionary tables.

Sources of Lists

1. sashelp views
2. sql dictionary tables
3. summarization procedure output

MAKING UNIQUE FROM DATA

For this reporting example consider these methods to extract a unique list of parameter values from a data set. Each method produces a data set, `Regions`, with a list of unique values of the variable `Region`. We use this list to write the calls to the macro `Report`.

See Ex00: review or
Ex02: symputx.

- proc sort

make-unique-sort.sas

```
1  PROC Sort data = sashelp.Shoes
2           (keep = Region)
3            out  = Regions
4                   nodupkey;
5            by     Region;
```

- proc sql

make-unique-sql-create.sas

```
1  PROC SQL; create table    Regions as
2            select distinct Region
3            from   sashelp.Shoes;
4            quit;
```

USING LISTS

Having obtained a list of unique values in a table, we next need to convert the list stored in the table to a list available in the macro environment. Then we can generate the series of macro calls from this list. Consider the following methods to generate these calls to the macro Report:

- make table into macro array, using either:
  1. symputx — See Ex02.
  2. sql into — See Ex03.

  then call in macro `%do` loop: ..... See Ex04.
- make table into macro variable,
  pick item using `%scan` in `%do` loop — See Ex06.
- make table into macro variable containing calls,
  execute calls in macro variable ..... See Ex08.
- read table, write calls to file and include — See Ex09.
- read table, call execute generates calls ..... See Ex11.

MAKE AND ITERATE A MACRO ARRAY

Call SymputX: The `call symputx` data step routine is used to assign a value to a macro variable. By concatenating a number to the macro variable name we create what is effectively a macro array: a sequentially numbered set of macro variables. In this data step a series of macro variables of the form `Item1`, `Item2`, ..., `Item`*N*, is created, each holding the name of one region. When these macro variables are used inside a macro `%do` loop the macro variable form `&&Item&I` uses the `&I` counter as an index to the macro array; this allows us to step through the list of values.

Note: In this example we allocate the macro variable `NmbrItems` in line 7, and assign it the value of the upper bound of our macro array in line 15 which is used in the `%do` loop in line 19.

Note in lines 5, 17, and 28: the use of the macro variable `Testing` to print to the log.

See Ex02.

make-array-symputx.sas

```
1  %Include Project(make-unique-sort);
2  %Include Project(Report-Region-macro);
3  %Macro Demo(InData    = Regions
4              ,InVar     = Region
5              ,Testing   = 0);
6  %local I         ; %Let I         = 0;
7  %local NmbrItems; %Let NmbrItems = 0;
8
9  DATA _Null_;
10 do   I = 1 to             Nrows;
11      set  &InData. nobs = Nrows;
12      call symputx('Item' !! left(put(I,8.))
13                   ,&InVar.,       'l');
14      end;
15 call symputx('NmbrItems', Nrows, 'l');
16 run;
17 %if &Testing. %then %put _local_;
18
19 %do I = 1 %to &NmbrItems.;
20     %Put *Report(&InVar. = &&Item&I.);
21          %Report(&InVar. = &&Item&I.);
22     %end;
23 run; %Mend Demo;
24
25 options mprint;%*echo macro statements;
26 %Demo(InData    = Regions
27      ,InVar     = Region
28      ,Testing   = 1);
```

Line 17 lists the local macro variables.

make-array-symputx.log

```
77 DEMO NMBRITEMS 10
78 DEMO INDATA Regions
79 DEMO I 0
80 DEMO ITEM1 Africa
81 DEMO ITEM10 Western Europe
82 DEMO TESTING 1
83 DEMO ITEM3 Canada
```

The macro `%do` loop in lines 19–22 writes a note to the log and calls the macro Report with each value of Region.

make-array-symputx.log

```
92 *Report(Region = Africa)
93 MPRINT(REPORT):   ods html file = "Africa.html" style = journal;
94 NOTE: Writing HTML Body file: Africa.html
95 MPRINT(REPORT):   Title "Sales in Africa";
96 MPRINT(REPORT):   PROC Report data = sashelp.Shoes nowindows;
97 MPRINT(REPORT):   where Region = "Africa";
```

Here is the list of the html files produced by this method:

zdir-html.txt

```
6  03/02/2007  09:36a              17,407 Africa.html
7  03/02/2007  09:36a              15,603 Asia.html
8  03/02/2007  09:36a              16,327 Canada.html
9  03/02/2007  09:36a              15,992 Central America-Caribbean.html
10 03/02/2007  09:35a              15,973 Eastern Europe.html
11 03/02/2007  09:35a              15,615 Middle East.html
12 03/02/2007  09:36a              16,692 Pacific.html
13 03/02/2007  09:36a              17,057 South America.html
14 03/02/2007  09:36a              16,341 United States.html
15 03/02/2007  09:36a              17,404 Western Europe.html
```

. . . and the partial listing of the Africa report: See Ex01.

Report-Africa.lst

```
Sales in Africa

                                    Subsidiary
                  Addis Ababa       Algiers          Cairo  Johannesburg
  Product
  Boot                $29,761       $21,297         $4,846         $8,365
  Men's Casual        $67,242       $63,206       $360,209              .
  Men's Dress         $76,793      $123,743         $4,051              .
  Sandal              $62,819       $29,198        $10,532        $17,337
```

SQL into: A series of macro variables can also be created using Proc SQL. Unlike the data step which uses the `symputx` routine, SQL uses the `into` clause to create the series of macro variables. The macro variable names for the lower and upper bounds are preceded with a colon, line 9. The code shown here also creates a series of macro variables of the form of `Item1`, `Item2`, ..., `Item`*N*.

Note-1: The macro variable `SysMaxLong` — line 9 — ensures that the upper bound of the macro array, i.e., the *N-th* item, is created.

Note-2: The advantage of SQL is that the automatic macro variable `SqlObs`, line 15, contains the value of the upper bound of the macro array.

Note-3: It is not necessary to sort InData when using PROC SQL as the data set with the unique list of values is prepared with the distinct function, line 8.

See Ex03 and Ex04.

Most PROC SQL examples show the creation of a macro array with this statement: `into :Item1 - :Item999`

make-array-sql-into.sas

```
1  %Include Project(report-Region-macro);
2  %Macro Demo(InData  =
3              ,InVar   =
4              ,Testing = 0);
5  %local I;  %Let   I = 1;
6
7  PROC SQL  %If not &Testing %then noprint;
8          ; select distinct &InVar.
9            into   :Item1 - :Item&SysMaxLong.
10           from   &InData.;
11           quit;
12 run;
13 %If &Testing. %then %Put _local_;
14
15 %do I = 1 %to &SqlObs.;
16     %Put *Report(Region = &&Item&I.);
17          %Report(Region = &&Item&I.);
18     %end;
19 run; %Mend Demo;
20 %Demo(InData  = sashelp.Shoes
21      ,InVar   = Region
22      ,Testing = 1);
23
24 %Demo(InData  = sashelp.Shoes
25      ,InVar   = Region
26      ,Testing = 0);
```

SysMaxLong.log

```
22 1          %Put SysMaxLong<&SysMaxLong.>;
23 SysMaxLong<2147483647>
```

Using `SysMaxLong` ensures that all the macro variables of your list are created, i.e., for the 1,000-th Item — `Item1000` — on up to the maximum number of rows in the table.

MAKE AND SCAN A MACRO VARIABLE

Rather than create a series of macro variables each with one value, as was done in the previous two examples, we can create a single macro variable that contains the series of values.

This version of the `%Demo` macro has been further generalized by turning the name of the macro to be called into a macro parameter, `MacroName`, as well. This feature allows us to test quicker, because we do not write all ten html files.

In the macro `%do` loop in lines 19–24, we use a `%Put` in line 21 to show that individual values are used as a parameter to a macro call. Here SQL is used to place the values of Region into the macro variable `List`, line 12. The `separated by` clause, line 13, instructs SQL to create a concatenated list of values that are separated by the `Delimiter` value in quotes.

Instead of using the `&&Item&I` form as was done before, we now use the `%scan` function to parse the individual values from the macro variable `List`, line 20.

In this example, line 19, the number of items in the list is stored in the automatic macro variable `SqlObs`.

Note in lines 6, 17, and 22, the use of the macro parameter `Testing` which either writes the values in macro variable `List` to the log or executes the macro calls.

See Ex05.

write-list-into-mvar-scan.sas

```
1  %Include Project(ReportTest);
2  %Macro Demo(InData    =
3              ,InVar     =
4              ,Delimiter = +
5              ,MacroName =
6              ,Testing   = 0);
7  %local I;    %Let    I = 1;
8  %local List;%Let List = *empty;
9
10 PROC SQL  %If not &Testing %then noprint;
11         ; select distinct &InVar.
12             into   :List
13                    separated by "&Delimiter."
14             from   &InData.;
15             quit;
16 run;
17 %if &Testing. %then %put List<&List.>;
18
19 %do I = 1 %to &SqlObs.;
20     %Let Item = %scan(&List.,&I.,&Delimiter.);
21     %Put *&MacroName.(&InVar. = &Item.);
22     %if  not &Testing. %then
23          %&MacroName.(&InVar. = &Item.);
24     %end;
25 run; %Mend Demo;
```

Note that we now use the macro `ReportTest` so as not to write the html files.

write-list-into-mvar-scan.log

```
48 30          %Demo(InData    = sashelp.Shoes
49 31               ,InVar     = Region
50 32               ,MacroName = ReportTest
51 33               ,Testing   = 1);
```

The values in the macro variable `List`, written while testing:

write-list-into-mvar-scan.log

```
60 List<Africa+Asia+Canada+Central America/Caribbean+Eastern Europe+Middle
61 East+Pacific+South America+United States+Western Europe>
62 *ReportTest(Region = Africa)
63 *ReportTest(Region = Asia)
```

We use the option `mprint` to continue our testing with the macro `ReportTest`.

write-list-into-mvar-scan.log

```
72 34          options mprint;%* view macro statements;
73 35          %Demo(InData    = sashelp.Shoes
74 36               ,InVar     = Region
75 37               ,MacroName = ReportTest
76 38               ,Testing   = 0);
77 MPRINT(DEMO):   PROC SQL noprint ;
78 MPRINT(DEMO):   select distinct Region into :List separated by "+" from
79 sashelp.Shoes;
80 MPRINT(DEMO):   quit;
```

. . . and see the macro calls are correct.

write-list-into-mvar-scan.log

```
89 *ReportTest(Region = Africa)
90 REPORTTEST REGION Africa
91 MPRINT(REPORTTEST):   run;
92 *ReportTest(Region = Asia)
93 REPORTTEST REGION Asia
94 MPRINT(REPORTTEST):   run;
```

WRITE CALLS TO A MACRO VARIABLE

In the previous example the macro variable `List` was used to hold only the list of distinct regions. Since we want to use each region in a macro call we could construct the list to contain the macro calls themselves.

In this example the concatenation function `cat` is used to build the macro call.

Note the function `cat` does not trim trailing spaces; thus we use `trim(Item)` in line 11. Compare the function cat, used here, to the function `cats` in later examples.

See Ex07.

write-calls-into-mvar.sas

```
1  %Include Project(ReportTest);
2  %Macro Demo(InData    =
3              ,InVar     =
4              ,MacroName =
5              ,Testing   = 0);
6
7  PROC SQL  %If not &Testing. %then noprint;
8          ; select distinct &InVar. as Item,
9                    cat( '%'
10                      , "&MacroName.(&InVar.="
11                      , trim(Item), ')'
12                      ) as CallMacro
13           into   :Item,
14                  :List  separated by ' '
15           from   &InData.;
16           quit;
17 run;
18 %If  not &Testing. %then
19      &List.; %*execute statements in mvar List;
20 run; %Mend Demo;
```

When testing,

write-calls-into-mvar.log

```
43 25         %Demo(InData    = sashelp.Shoes
44 26              ,InVar     = Region
45 27              ,MacroName = ReportTest
46 28              ,Testing   = 1);
47 MPRINT(DEMO):   PROC SQL ;
48 MPRINT(DEMO):   select distinct Region as Item, cat( '%' ,
49 "ReportTest(Region=" , trim(Item), ')' ) as CallMacro into :Item, :List
50 separated by ' ' from sashelp.Shoes;
51 MPRINT(DEMO):   quit;
52 NOTE: The PROCEDURE SQL printed page 1.
```

. . . the calls of the macro are written to the list:

write-calls-into-mvar.lst

```
6  Item                          CallMacro
7  ------------------------------------------------------------------------
8  Africa                        %ReportTest(Region=Africa)
9  Asia                          %ReportTest(Region=Asia)
10 Canada                        %ReportTest(Region=Canada)
```

When not testing

write-calls-into-mvar.log

```
62 29         %Demo(InData    = sashelp.Shoes
63 30              ,InVar     = Region
64 31              ,MacroName = ReportTest
65 32              ,Testing   = 0);
66 MPRINT(DEMO):   PROC SQL noprint ;
67 MPRINT(DEMO):   select distinct Region as Item, cat( '%' ,
68 "ReportTest(Region=" , trim(Item), ')' ) as CallMacro into :Item, :List
69 separated by ' ' from sashelp.Shoes;
70 MPRINT(DEMO):   quit;
```

. . . the calls are executed:

write-calls-into-mvar.log

```
78 MPRINT(DEMO):   run;
79 REPORTTEST REGION Africa
80 MPRINT(REPORTTEST):   run;
81 REPORTTEST REGION Asia
82 MPRINT(REPORTTEST):   run;
```

WRITE TO FILE THEN INCLUDE

Write to file, then Include, was the first method available to programmers to replicate statements. It is reviewed here in order for you to recognize the algorithm and note that it may be replaced with any of the above macro variable processing methods.

The authors share the opinion that this technique is used primary by those programmers who do not fully understand the approaches described in the preceeding examples.

Note: compare the `Stmnt` assignment using function `cats` in line 16 with `cat` in write-calls-call-execute.

See Ex09.

write-calls-to-file-include.sas

```
1  %Include Project(ReportTest);
2  %Include Project(make-unique-sort);
3  %Macro Demo(InData    =
4              ,InVar     =
5              ,MacroName =
6              ,Testing   = 0);
7
8  filename TempFile 'zCallMacro.txt';
9
10 DATA   _Null_;
11 attrib Stmnt length = $100;
12 file   TempFile;
13
14 do until(EndoFile);
15    set  &InData. end = EndoFile;
16    Stmnt = cats('%', "&MacroName.(&InVar.="
17                  ,                  &InVar.
18                  ,                ')' );
19    %If &Testing. %then %do;
20        putlog Stmnt=;
21        %end;
22    put Stmnt;
23    end;
24 stop; run;
25
26 %If not &Testing. %then %do;
27     %Include TempFile;
28     %end;
29 filename TempFile clear;
30 run;      %Mend Demo;
```

When testing, the calls of the macro are written to the log:

write-calls-to-file-include.log

```
63 40          %Demo(InData    = Regions
64 41               ,InVar     = Region
65 42               ,MacroName = ReportTest
66 43               ,Testing   = 1);
67
68 NOTE: The file TEMPFILE is:
69       File Name=C:\LaTeX\HOW-list-proc\sas\zCallMacro.txt,
70       RECFM=V,LRECL=256
71
72 Stmnt=%ReportTest(Region=Africa)
73 Stmnt=%ReportTest(Region=Asia)
74 Stmnt=%ReportTest(Region=Canada)
```

. . . and the TempFile:

zCallMacro.txt

```
1  %ReportTest(Region=Africa)
2  %ReportTest(Region=Asia)
3  %ReportTest(Region=Canada)
```

When not testing the TempFile is `%Include`d. Note use of option `source2` which allows us to see the statements in the included file in the log; this is similar to the option `mprint`.

write-calls-to-file-include.log

```
44         options source2;%*echo Include to log;
45         %Demo(InData    = Regions
46               ,InVar     = Region
47               ,MacroName = ReportTest
48               ,Testing   = 0);
49
NOTE: The file TEMPFILE is:
      File Name=C:\LaTeX\HOW-list-proc\sas\zCallMacro.txt,
      RECFM=V,LRECL=256
103
NOTE: 10 records were written to the file TEMPFILE.
```

write-calls-to-file-include.log

```
NOTE: %INCLUDE (level 1) file TEMPFILE is file
      C:\LaTeX\HOW-list-proc\sas\zCallMacro.txt.
49         +%ReportTest(Region=Africa)
REPORTTEST REGION Africa
50         +%ReportTest(Region=Asia)
REPORTTEST REGION Asia
```

CALL EXECUTE

The Call Execute data step routine is the successor to Write To File Include.

A series of macro calls can be managed in the data step. The `call execute` routine allows us to build statements that are stacked for later execution. If these statements contain macro calls, the macro calls are executed as soon as the data step terminates.

In this example the data step reads the unique values of region, and uses the `cats` function to build the macro call with the current value of region. Again there will be one macro call for each region, but we did not have to create any of the intermediate macro variables!

See Ex10.

write-calls-call-exec-demo.sas

```
%Include Project(make-unique-sort);
%Include Project(ReportTest);
%Macro Demo(InData    =
           ,InVar     =
           ,MacroName =
           ,Testing   = 0);
DATA _Null_;
do   until(EndoFile);
     set &InData.   end = EndoFile;
     call execute(
              cats('%', "&MacroName.(&InVar.="
                  ,                  &InVar.
                  ,')' %*end macro call;
                  )    %*end cats;
                  );   %*end execute;
     end;
stop; run; %*calls executed here;
run;  %Mend Demo;
%Demo(InData    = Regions
     ,InVar     = Region
     ,MacroName = ReportTest
     ,Testing   = 1);
```

Caveat: Call Execute does not execute macros correctly when the macro contains complexity: i.e., when the macro contains any of these statements: `%if`, `%do`, or `symput`, as do our macros Report and ReportTest. An explanation of the processing issue is beyond the scope of this paper. The technical description is that macros called by call execute are expanded when they are pushed onto the SAS language processor stack. In order for the macro to work correctly it must be expanded when the macro call is popped from the stack. We accomplish this delay with the `nrstr` function, — NoRescan STRing — shown in the next example.

Whitlock [13, sugi22.070] illustrates the problems associated with executing macro statements before SAS statements.

This portion of the log shows that the macro statements are executed *before* the SAS statements; this is an indication that the macro is not executing the way that we would like.

write-calls-call-exec-demo.log

```
28          %Demo(InData    = Regions
29               ,InVar     = Region
30               ,MacroName = ReportTest
31               ,Testing   = 1);

REPORTTEST REGION Africa
REPORTTEST REGION Asia
```

. . . here are the SAS statements:

write-calls-call-exec-demo.log

```
NOTE: CALL EXECUTE generated line.
1          + run;
2          + run;
```

Our last example combines the algorithms from above examples so that the macro calls may be reviewed while testing.

Note.1 line 13: Compare the `Stmnt` assignment using function `cats` with function `cat` in write-calls-to-file-include.

write-calls-call-execute.sas

```
1  %Macro CallExec(InData    =
2                 ,InVar     =
3                 ,MacroName =
4                 ,Testing   = 0);
5  %If &Testing. %then %put _local_;
6
7  DATA   _Null_;
8  attrib Stmnt   length = $100;
9
10 do until(EndoFile);
11    set &InData. end = EndoFile;
12    Stmnt = cats( '%', "&MacroName."
13                ,      "(Var=&InVar."
14                ,  ',Value=',&InVar.
15                ,      ')'
16                );
17    %if   &Testing. %then %do;
18          put Stmnt=;
19          %end;
20    %else %do;
21          call execute(cats('%nrstr(', Stmnt, ')'
22                       )    );
23          %end;
24    end; %*do until(EndoFile);
25 stop; run; %*macro calls execute here;
26 run;  %Mend;
```

Note.2 lines 13–15: This production macro writes a different macro call than our previous examples:

previous: `%Report(Region=Africa)`
here: `%Report(Var=Region ,Value=Africa)`

Note.3 line 22: Using the `nrstr` quoting function. In this `call execute` statement, the `nrstr` delays the expansion of the macro call until it is removed (popped) from the stack.

See Ex11.

Note that we use a different `ReportTest` macro for this production version, so that this program may be used to test and iterate other macros besides our Report-sashelp.Shoes.Region macro.

ReportTestVarValue.sas

```
1  %Macro ReportTest(Var   =
2                   ,Value =);
3  %put _local_;
4  %Put (where = (&Var. eq "&Value."));
5  run; %Mend;
```

See the write-calls-call-exec-Class-Sex example following write-calls-call-exec-Shoes-Region.

Here is the test program for sashelp.Shoes.Region.

write-calls-call-exec-Shoes-Region.sas

```
1  *options  source2;%* echo include statements;
2  %Include Project(make-unique-sort);
3  %Include Project(ReportTestVarValue);
4  %Include Project(write-calls-call-execute);
5  options  mprint;%* echo macro statements;
6  %CallExec(InData    = Regions
7           ,InVar     = Region
8           ,MacroName = ReportTest
9           ,Testing   = 1);
10 %CallExec(InData    = Regions
11          ,InVar     = Region
12          ,MacroName = ReportTest);
```

This portion of the log shows that the statements written to the log while testing.

write-calls-call-exec-Shoes-Region.log

```
57 Stmnt=%ReportTest(Var=Region,Value=Africa)
58 Stmnt=%ReportTest(Var=Region,Value=Asia)
```

This portion of the log shows that the macro statements are executed correctly.

write-calls-call-exec-Shoes-Region.log

```
98  NOTE: CALL EXECUTE generated line.
99  1          +  %ReportTest(Var=Region,Value=Africa)
100 REPORTTEST VALUE Africa
101 REPORTTEST VAR Region
102 (where = (Region eq "Africa"))
103 MPRINT(REPORTTEST):   run;
104 2          +  %ReportTest(Var=Region,Value=Asia)
105 REPORTTEST VALUE Asia
106 REPORTTEST VAR Region
107 (where = (Region eq "Asia"))
108 MPRINT(REPORTTEST):   run;
```

This test program for sashelp.Class.Sex shows that the macros `CallExec` and `ReportTest` can be used to test other list processing macros.

write-calls-call-exec-Class-Sex.sas

```
1  *options  source2;%* echo include statements;
2  %Include Project(ReportTestVarValue);
3  %Include Project(write-calls-call-execute);
4  options  mprint;%* echo macro statements;
5  PROC Sort data = sashelp.Class
6           (keep = Sex)
7            out  = Sexs
8                   nodupkey;
9            by     Sex;
10 run;
11 %CallExec(InData    = Sexs
12          ,InVar     = Sex
13          ,MacroName = ReportTest
14          ,Testing   = 1);
```

write-calls-call-exec-Class-Sex.log

```
62 Stmnt=%ReportTest(Var=Sex,Value=F)
63 Stmnt=%ReportTest(Var=Sex,Value=M)
```

SUMMARY There are a number of ways to both build and use a list of values in the macro language. To the programmer the huge advantage is that we can automate the process. SAS can build the list; SAS can determine how many items are in the list; SAS can step through the list. The processing of the list is independent of the programmer. It is a dynamic process!

While the code examples in this paper are fairly straightforward real life tends not to be so easy. Study this paper and the references in the 'Suggested Reading' section. Be patient with yourself. Pick the method that makes the most sense to you and generate a simple example. Practice. Don't expect a full understanding to be instant. Our brains tend not to be that cooperative, but with practice you too will be able to generate dynamic applications.

SUGGESTED READING

applications	DiIorio and Abolafia [5, sugi29.237] provide a dozen examples, Fehd [9, nesug2006.014] shows a data review report which writes constant text of macro calls into a macro variable, Fehd [8, pnwsug2006.012] provides sql examples using the more common dictionary tables
books	Burlew [1, saspress.56516] ; Carpenter [2, saspress.59224]
call execute	Michel [11, sugi30.027] shows applications using procs compare, copy and print; Virgile [12, sugi22.086] demonstrates conditional execution of procedures and compares macro do loops with call execute; Whitlock [13, sugi22.070] highlights the problems of executing code containing both macro and sas statements
macro arrays	Carpenter [3, sugi30.028] compares symput with sql to create macro arrays, demonstrates use of scl functions; Clay [4, sugi31.040] presents two macro functions built with macro arrays; Fehd [6, sugi22.080] uses proc contents output data set and symput to create macro arrays; Fehd [7, sugi29.070] uses proc sql to create macro arrays
sashelp views	DiIorio and Abolafia [5, sugi29.237] provide a thorough comparision of sashelp views and sql dictionary tables
SQL	DiIorio and Abolafia [5, sugi29.237] provide examples using sql; Fehd [7, sugi29.070] uses proc sql to create macro arrays ; Fehd [9, nesug2006.014] uses proc sql to write constant text of macro calls into a macro variable; Fehd [8, pnwsug2006.012] provides examples using most common sql dictionary tables; Feng [10, sugi31.044] discusses general usage of proc sql; Whitlock [14, sugi26.060] compares creating and iterating macro arrays with writing constant text of macro calls into a macro variable

BIBLIOGRAPHY

[1] Michele Burlew. *SAS Macro Programming Made Easy, Second Edition*. Cary, NC: SAS Institute Inc., 1998. ISBN 978-1-59047-882-0. URL `http://support.sas.com/publishing/bbu/companion_site/56516.html`.

[2] Arthur L. Carpenter. *Carpenter's Complete Guide to the SAS Macro Language, Second Edition*. Cary, NC: SAS Institute Inc., 2004. URL `http://support.sas.com/publishing/bbu/companion_site/59224.html`. 13 chap., 475 pp., appendices: 5, glossary: 3 pp., bibliography: 19 pp., index: 13 pp.

[3] Arthur L. Carpenter. Storing and using a list of values in a macro variable. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/028-30.pdf`. Coders' Corner, 6 pp.; making macro arrays using symputx or sql, iterating macro arrays, using scl functions, bibliography.

[4] Ted Clay. Tight looping with macro arrays. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL `http://www2.sas.com/proceedings/sugi31/040-31.pdf`. Coders' Corner, 8 pp.; two macro functions: array and do_over.

[5] Frank DiIorio and Jeff Abolafia. Dictionary tables and views: Essential tools for serious applications. In *Proceedings of the 29th SAS User Group International Conference*, 2002. URL `http://www2.sas.com/proceedings/sugi29/237-29.pdf`. Tutorials, 19 pp.; comparison of data structure of dictionary tables and sashelp views, review of differences between v6, v8, and v9, 12 examples.

[6] Ronald Fehd. Array: Construction and usage of arrays of macro variables. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL http://www2.sas.com/proceedings/sugi22/CODERS/PAPER80.PDF. Coders' Corner, 4 pp.; using proc contents output and symput, bibliography.

[7] Ronald Fehd. Array: Construction and usage of arrays of macro variables. In *Proceedings of the 29th SAS User Group International Conference*, 2002. URL http://www2.sas.com/proceedings/sugi29/070-29.pdf. Coders' Corner, 6 pp.; using proc sql, bibliography.

[8] Ronald Fehd. How to use proc sql select into for list processing. In *Proceedings of the Pacific NorthWest SAS User Group Conference*, 2006. URL http://www.pnwsug.com/Conference_2006/Procedings/PNWSUGotherfiles/PN12FehdSQL.pdf. Hands On Workshop, 15 pp.; sql select syntax, comparison of procedures vs. sql, writing constant text into macro variable, using dictionary tables; bibliography.

[9] Ronald Fehd. Journeymen's tools: Data review macro FreqAll – using proc sql list processing with dictionary.columns to eliminate macro do loops. In *Proceedings of the NorthEast SAS User Group Conference*, 2006. URL http://www2.sas.com/proceedings/sgf2007/028-2007.pdf. Coders' Corner, 10 pp.; designing macros for reporting, creating and using macro arrays, writing text of macro calls into macro variable, executing macro calls in macro variable, bibliography.

[10] Ying Feng. The sql procedure: When and how to use it. In *Proceedings of the 31st SAS User Group International Conference*, 2006. URL http://www2.sas.com/proceedings/sugi31/044-31.pdf. Coders' Corner, 7 pp.; using sql to make and iterate a macro array, bibliography.

[11] Denis Michel. Call execute: A powerful data management tool. In *Proceedings of the 30th SAS User Group International Conference*, 2005. URL http://www2.sas.com/proceedings/sugi30/027-30.pdf. Applications Development, 13 pp.; reading sql dictionary tables, executing constant text, bibliography.

[12] Bob Virgile. Magic with call execute. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL http://www2.sas.com/proceedings/sugi22/CODERS/PAPER86.PDF. Coders' Corner, 2 pp.; conditional execution of procedures, comparison of macro do loop and call execute.

[13] H. Ian Whitlock. Call execute: How and why. In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL http://www2.sas.com/proceedings/sugi22/CODERS/PAPER70.PDF. Coders' Corner, 5 pp.; executing statements and macros, comparing compile and execution timing of sas and macro, bibliography.

[14] Ian Whitlock. Proc sql: Is it a required tool for good sas programming? In *Proceedings of the 22nd SAS User Group International Conference*, 1997. URL http://www2.sas.com/proceedings/sugi26//p060-26.pdf. Beginning Tutorials, 6 pp.; using sql to create macro arrays, selecting distinct text into macro variable.

Author: Ronald Fehd **mailto:RJF2@cdc.gov**
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

Author: Art Carpenter **mailto:art@caloxy.com**
CA Occidental Consultants **www.caloxy.com**
PO Box 430
Vista, CA 92085-0430

To get the code examples in this paper send an e-mail to the author mailto:RJF2@cdc.gov with the subject:

```
request HOW List Processing
```


Document Production: This paper was typeset in LaTeX. For further information about using LaTeX to write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                    :
The subject is or contains: LaTeX
The author's address        : RJF2
Since                       : 01 June 2003
```

APPENDIX

PROJECT FILES FOR BATCH PROCESSING

These files are used in the project folder in order to run each of the programs and produce .log and .lst files for inclusion in this paper.

sas.bat

```
rem    name: SAS.bat for either V8 or V9
if not exist "C:\Program Files\SAS\SAS 9.1\SAS.exe" goto v8
            "C:\Program Files\SAS\SAS 9.1\SAS.exe" %*
goto EOF
:v8
    "C:\Program Files\SAS Institute\SAS\V8\SAS.exe" %*
:EOF
```

sasv8.cfg

```
1  /*name: SASv8.cfg
2  06Jun08 RJF2 ........ */
3  -config 'C:\Program Files\SAS Institute\SAS\V8\SASv8.cfg'
4  -SET                ProjRoot     'C:\LaTeX\HOW-list-proc'
5  -SASinitialFolder '!ProjRoot\sas'
6
7  -noovp      /* no overprint errors nor warnings */
8  -nosplash   /* no display SAS banner at startup */
```

NOTE: if you use either of these configuration files you *must* change the value of `ProjRoot`: `sasv8.cfg`: line 4 `sasv9.cfg`: line 4 to the name of the *parent* folder of your sas programs. If `sas` is not the name of your program folder then change the value of `SASinitialFolder` as well. Here is an example for programs in `C:/temp/programs`:

```
-set ProjRoot 'C:\temp'
-sasinitialfolder '!ProjRoot\programs'
```

sasv9.cfg

```
1  /*name: SASv9.cfg
2  06Jun08 RJF2
3  ******* ............................... */
4  -SET                ProjRoot     'C:\LaTeX\HOW-list-proc'
5  -SASinitialFolder '!ProjRoot\sas'
6
7  -noovp      /* no overprint errors nor warnings */
8  -nosplash   /* no display SAS banner at startup */
```

autoexec.sas

```
1  filename Project '.';     %* here: same folder      ;
2  options  nocenter         %* flush left output      ;
3           nodate           %* no date-stamp          ;
4           nonumber         %* no page numbers        ;
5           details          %* Proc Contents          ;
6           formchar =       %* no special chars       ;
7                            '--------------------'
8           formdlim = ' '   %* no CR/LF for page break;
9           fullstimer       %* expand time block      ;
10          linesize = 72    %* for LaTeX printing     ;
11          pagesize = max;  %* no page breaks in *.lst;
12 Title   'SGF: Fehd+Carpenter: HOW on list processing';
13
14 %* Arts path; %Let Path
15 =C:\Primary\Presentations\papers\Papers70-79\72-ManagingLists\Workshop\;
16
17 %* Rons path; %Let Path = ;%*here;
```

EXAMPLE FILES FOR WORKSHOP

Ex00.sas

```
1  *Ex00  fix path for SGF;
2  %*Let Path = ???;
3  *review:;
4  %*Include Project(make-unique-sort);
5  %*Include Project(Report-Region-macro);
6  %*Include Project(ReportTest);
```

1. see: make-unique-sort
2. see: Report-Region-macro
3. see: ReportTest

Ex01.sas

```
1  * Ex01.sas;
2  ods html file  = 'Africa.html'
3           style = journal;
4
5  Title 'Sales in Africa';
6
7  PROC Report data = sashelp.Shoes
8                     nowindows;
9              where  Region = 'Africa';
10             column Product Subsidiary,
11                                  Sales;
12             define Product    / group;
13             define Subsidiary / across;
14             define Sales      / analysis
15                                   '';
16 run;
17 ods html close;
```

1. see: Report-Africa

Ex02.sas

```
1  * Ex02.sas: vertical to vertical: ;
2  *           macro array;
3  *Q: name of variable with number of items?;
4
5  options  nosource2;%*echo Include to log;
6  %Include Project(make-unique-sort);
7
8  DATA  _Null_;
9  set   Regions;
10 Count +1;
11 call  symputx('Region' || left(put(
12                         Count , 3.))
13              , Region,          'l' );
14 call  symputx('Count ', Count , 'l' );
15 run;
16 %Put _global_;%* Show macro variables;
17
18 %Put Region1<&Region1.>;
19 *task: show 3rd region;
```

1. see: make-unique-sort

Ex03.sas

```
1  * Ex03.sas: vertical to vertical: ;
2  *           macro array;
3
4  *Q: name of variable with number of items?;
5
6  PROC SQL print;
7           select distinct    Region
8           into   :Region1 - :Region999
9           from   sashelp.Shoes;
10          quit;
11
12 %Put _global_;%* Show macro variables;
13
14 %Let I = 3;
15 %Put Region&I.<&&Region&I..>;
16 *task: show last region;
```

Ex04.sas

```
* Ex04.sas: vertical to vertical: ;
*           macro array, do loop;

options  nosource2;%*echo Include to log;
%Include Project(Report-Region-macro);
%Include Project(ReportTest);

options mprint;
%macro  Demo;
%local  I;

PROC SQL noprint;
         select distinct   Region
         into  :Region1 - :Region999
         from   sashelp.Shoes;
         quit;

%do I = 1 %to &SqlObs.;
    %put *Report(Region = &&Region&I.);
    %end;
%mend Demo;
%Demo;

*task    : execute macro Report;
*solution: see make-array-sql-into;
```

1. see: Report-Region-macro
2. see: ReportTest

see: make-array-sql-into

1. see: Report-Region-macro
2. see: ReportTest

Ex05.sas

```
* Ex05.sas: vertical to horizontal: ;
*           write list to macro var;

PROC SQL noprint;
         select distinct Region
         into           :Regions
                separated by '+'
         from   sashelp.Shoes;
         quit;

%put Regions |&Regions.|;

*task    : use macro function scan;
*          show 5th region;
*put region5(scan(Regions,...));
*solution: see Ex06;
```

see: write-list-into-mvar-scan

Ex06.sas

```
* Ex06.sas: vertical to horizontal: ;
*           write list to macro var, scan;

options  nosource2;%*echo Include to log;
%Include Project(Report-Region-macro);
%Include Project(ReportTest);

options mprint;
%macro  Demo;
%local  I Item;

PROC SQL noprint;
         select distinct Region
         into   :Regions separated by '+'
         from   sashelp.Shoes;
         quit;

%put Regions: |&Regions.|;
%do I = 1 %to &SqlObs.;
    %let Item = %scan(&Regions., &I., +);
    %put *ReportTest(Region = &Item.);
    %end;
run;%mend Demo;
%Demo

%*task    : execute macro ReportTest;
%*solution: see write-list-into-mvar-scan;
```

Ex07.sas

```
* Ex07.sas: vertical to horizontal: ;
*           write macro calls   ;
*                 into macro var;

options  nosource2;%*echo Include to log;
%Include Project(Report-Region-macro);
%Include Project(ReportTest);

PROC SQL noprint;
         create table    Regions as
         select distinct Region
         from sashelp.Shoes;
         select cats( '*'
                    , 'Report(Region='
                    ,          Region
                    ,         ')'
                    )
         into :List separated by ' '
         from Regions;
         quit;

%Put List<&List.>;

*task    : execute macro ReportTest;
*solution: see Ex08;
```

Ex08.sas

```
* Ex08.sas: vertical to horizontal;
*           write macro calls   ;
*                 into macro var, execute;

options  nosource2;
%Include Project(Report-Region-macro);
%Include Project(ReportTest);

PROC SQL noprint;
         create table    Regions as
         select distinct Region
         from sashelp.Shoes;
         select cats( '%'
                    , 'ReportTest(Region='
                    ,              Region
                    ,             ')'
                    )
         into :List separated by ' '
         from Regions;
         quit;

&List.;%*execute calls;
```

Ex09.sas

```
* Ex09.sas: vertical:
*           add prefix and suffix;
*           write to file, include;

options  nosource2;%*echo Include to log;
%Include Project(make-unique-sort);
%Include Project(Report-Region-macro);
%Include Project(ReportTest);
options  mprint;

filename TempFile 'zCallMacroX.txt';

DATA   _Null_;
attrib Stmnt length = $100;
file   TempFile;

do until(EndoFile);
   set  Regions end = EndoFile;
   Stmnt = cats( '*'
               , 'ReportTest(Region='
               ,             Region
               ,           ')' );
   putlog Stmnt=;
   put    Stmnt;%*to file;
   end;%* do until;
stop; run;

%Include TempFile/source2;
filename TempFile clear;
run;

*task    : execute macro ReportTest;
*solution: see write-calls-to-file-include;
```

see: write-calls-to-file-include

Ex10.sas

```
* Ex10.sas: vertical: ;
*           add prefix and suffix;
*           call execute without nrstr;

options  nosource2;%*echo Include to log;
%Include Project(make-unique-sort);
%Include Project(Report-Region-macro);
%Include Project(ReportTest);
%* NOTE: not works correctly!;
options  mprint;

DATA _Null_;
set  Regions;
call execute('%ReportTest(Region='
                        || Region || ')');
*call execute('%Report   (Region='
                        || Region || ')');
run;%*calls executed here;
run;
```

See explanation at Caveat.

Ex11.sas

```
* Ex11.sas: vertical: ;
*           add prefix and suffix;
*           call execute with nrstr;

options  nosource2;%*echo Include to log;
%Include Project(make-unique-sort);
%Include Project(Report-Region-macro);
%Include Project(ReportTest);
*Include Project(ReportTestVarValue);
options  mprint;

DATA _Null_;
set  Regions;
call execute(
         cats(
       '%nrstr('
              ,'%ReportTest(Region='
              ,             Region
              ,           ')' %*end macro;
            ,')'              %*end nrstr;
             )                %*end cats;
            );                %*end execute;
run;%*calls executed here;
run;

*task: execute ReportTest
*      with the Var=
*               Value= parameters;
*solution: see write-calls-call-execute;
*solution: see Ex11-solution;
```

see: write-calls-call-execute

SOLUTIONS TO EXAMPLES

S02.sas

```
%Put Region1<&Region1.>;

*task: show 3rd region;
%Put Region3<&Region3.>;
```

S02.log

```
26         *task: show 3rd region;
27         %Put Region3<&Region3.>;
Region3<Canada>
```

S03.sas

```
%Let I = 3;
%Put Region&I.<&&Region&I..>;

*task: show last region;
%*solution 1:;
%Let I = &SqlObs.;
%Put Region&I.<&&Region&I..>;

%*solution 2:;
%Put Region&SqlObs.<&&Region&SqlObs..>;
```

S03.log

```
18         %*solution 1:;
19         %Let I = &SqlObs.;
20         %Put Region&I.<&&Region&I..>;
Region10<Western Europe>
```

S04.sas

```
%do I = 1 %to &SqlObs.;
    %*task: execute macro Report:;
    %put *Report(Region = &&Region&I.);
         %Report(Region = &&Region&I.);
    %end;
```

S04.log

```
NOTE: Writing HTML Body file: Africa.html
MPRINT(REPORT):   Title "Sales in Africa";
```

S05.sas

```
*task    : use macro function scan;
*          show 5th region;
%put reg5(%scan(&Regions,5,+));
```

S05.log

```
15         %put reg5(%scan(&Regions,5,+));
reg5(Eastern Europe)
```

S06.sas

```
%do I = 1 %to &SqlObs.;
    %let Item = %scan(&Regions., &I., +);
    %*task: execute macro ReportTest;
    %put *ReportTest(Region = &Item.);
         %ReportTest(Region = &Item.);
    %end;
```

S06.log

```
*ReportTest(Region = Africa)
REPORTTEST REGION Africa
MPRINT(REPORTTEST):   run;
MPRINT(DEMO):  ;
```

S09.sas

```
   %*task    : execute macro ReportTest;
   %*Stmnt = cats( '*';
   Stmnt = cats( '%'
               , 'ReportTest(Region='
```

S09.log

```
76        +%ReportTest(Region=Africa)
REPORTTEST REGION Africa
MPRINT(REPORTTEST):   run;
77        +%ReportTest(Region=Asia)
```

S11.sas

```
%*        ,'%ReportTest(Region=';
call execute(
         cats(
       '%nrstr('
          ,'%ReportTest(Var=Region,Value='
              ,             Region
```

S11.log

```
1          +  %ReportTest(Var=Region,Value=Africa)
REPORTTEST VALUE Africa
REPORTTEST VAR Region
(where = (Region eq "Africa"))
MPRINT(REPORTTEST):   run;
```