

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

Preparing a Routine	2
Making Lists	3
Using Lists	4
Make and Iterate a Macro Array	5
Make and Scan a Macro Variable	6
Write Calls to a Macro Variable	8
Write to File then Include	9
Call Execute	10
Summary	13
Suggested Reading	13
Bibliography	13
Appendix	15
Project Files for Batch Processing	15
Example Files for Workshop	16
Solutions to Examples	20

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:

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

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;
%do I = 1 %to 5;
    %*...;
%macro DoLoop(Count);
%do I = 1 %to &Count;
    %*...;
%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, pg. 16.

```
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 occurrences of the string *Africa*. This is the first value of the variable *Region* in the data set *sashelp.Shoes*.

See Ex01, pg. 16.

```
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 occurrences of each parameter value with the macro variable.

Find and replace the three occurrences 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.

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.

Task: How to generate these macro calls?

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

See Ex00, pg. 16.

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

```
1 %macro Report( Region=Africa);
2 ... file = "&Region..html"
3 Title "Sales in &Region.";
4 where Region = "&Region.";
```

```
1 Report-Region-macro.sas
2 %Macro Report(InData = sashelp.Shoes
3 ,Region =
4 ,OutPath = /*here: same folder*/
5 );
6 %local Filename; %Let Filename = &Region.;
7 %** change slash to hyphen in
8 Central America/Caribbean;
9 %If %index(&Region,/) %then %Let Filename =
10 %sysfunc(translate(&Region.,-,/));
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
24 run;
25 ods html close;
26 run; %mend Report;
```

```
1 ReportTest.sas
2 %Macro ReportTest(Region = );
3 %put _local_;
4 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.

WHERE ARE LISTS?

SAS provides many unique data sets in either sashelp views or sql dictionary tables. Dilorio and Abolafia [5, sugi29.237] provide a comparison 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.

Definition of List

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

Sources of Lists

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

MAKING UNIQUE LISTS 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 pg. 16 or Ex02: symputx, pg. 16.

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`:

- `proc sort`

```
1  make-unique-sort.sas
2  PROC Sort data = sashelp.Shoes
3      (keep = Region)
4      out = Regions
5      nodupkey;
6      by Region;
```

- `proc sql`

```
1  make-unique-sql-create.sas
2  PROC SQL; create table Regions as
3      select distinct Region
4      from sashelp.Shoes;
5  quit;
```

- make table into macro array, using either:
 - 1. symputx Ex02, pg. 16
 - 2. sql into Ex03, pg. 16then call in macro `%do loop`: Ex04, pg. 17
- make table into macro variable, Ex05, pg. 17
pick item using `%scan` in `%do loop` Ex06, pg. 17
- make table into macro variable containing calls, Ex07, pg. 17
execute calls in macro variable Ex08, pg. 18
- read table, write calls to file and include Ex09, pg. 18
- read table, call execute generates calls Ex11, pg. 19

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, ..., ItemN, 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, pg. 16.

Line 17 lists the local macro variables.

```

1  %Include Project (make-unique-sort);
2  %Include Project (Report-Region-macro);
3  %Macro Demo(InData    = Regions
4          ,InVar      = Region
5          ,Testing     = 0);
6
7  %local I              ; %let I              = 0;
8  %local NmbrItems; %let NmbrItems = 0;
9
10 DATA _Null_;
11 do  I = 1 to              Nrows;
12     set  &InData. nobs = Nrows;
13     call symputx('Item' !! left(put(I,8.))
14               ,&InVar.,      '1');
15
16     end;
17 call symputx('NmbrItems', Nrows, '1');
18 run;
19 %if &Testing. %then %put _local_;
20
21 %do I = 1 %to &NmbrItems.;
22     %Put *Report(&InVar. = &&Item&I.);
23     %Report(&InVar. = &&Item&I.);
24
25 %end;
26 run; %Mend Demo;
27
28 options mprint; %*echo macro statements;
29 %Demo(InData    = Regions
30       ,InVar      = Region
31       ,Testing     = 1);

```

```

1  DEMO NMBRITEMS 10
2  DEMO INDATA Regions
3  DEMO I 0
4  DEMO ITEM1 Africa
5  DEMO ITEM10 Western Europe
6  DEMO TESTING 1
7  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.

```

1  *Report(Region = Africa)
2  MPRINT(REPORT):  ods html file = "Africa.html" style = journal;
3  NOTE: Writing HTML Body file: Africa.html
4  MPRINT(REPORT):  Title "Sales in Africa";
5  MPRINT(REPORT):  PROC Report data = sashelp.Shoes nowindows;
6  MPRINT(REPORT):  where Region = "Africa";

```

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

```

1  08/08/2008 08:36 AM 17,407 Africa.html
2  08/08/2008 08:36 AM 15,603 Asia.html
3  08/08/2008 08:36 AM 16,327 Canada.html
4  08/08/2008 08:36 AM 15,992 Central America-Caribbean.html
5  08/08/2008 08:36 AM 15,973 Eastern Europe.html
6  08/08/2008 08:36 AM 15,615 Middle East.html
7  08/08/2008 08:36 AM 16,692 Pacific.html
8  08/08/2008 08:36 AM 17,057 South America.html
9  08/08/2008 08:36 AM 16,341 United States.html
10 08/08/2008 08:36 AM 17,404 Western Europe.html

```

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

See Ex01, pg. 16.

Report-Africa.lst				
1	Sales in Africa			
2		Subsidiary		
3		Addis Ababa	Algiers	Cairo Johannesburg
4	Product			
5	Boot	\$29,761	\$21,297	\$4,846 \$8,365
6	Men's Casual	\$67,242	\$63,206	\$360,209 .
7	Men's Dress	\$76,793	\$123,743	\$4,051 .
8	Sandal	\$62,819	\$29,198	\$10,532 \$17,337
9	Slipper	\$68,641	\$64,891	\$13,732 \$39,452
10	Sport Shoe	\$1,690	\$2,617	\$2,259 \$5,172
11	Women's Casual	\$51,541	.	\$328,474 .
12	Women's Dress	\$108,942	\$90,648	\$14,095 \$42,682

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`, ..., `ItemN`.

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`

```

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
13 run;
14 %If &Testing. %then %Put _local_;
15
16 %do I = 1 %to &SqlObs.;
17     %Put *Report(Region = &&Item&I.);
18     %Report(Region = &&Item&I.);
19 %end;
20 run; %Mend Demo;
21 %Demo(InData = sashelp.Shoes
22     ,InVar = Region
23     ,Testing = 1);
24
25 %Demo(InData = sashelp.Shoes
26     ,InVar = Region
27     ,Testing = 0);

```

```

1  %Put SysMaxLong<&SysMaxLong.>;
2  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 and Ex06, pg. 17.

```

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
17 run;
18 %if &Testing. %then %put List<&List.>;
19
20 %do I = 1 %to &SqlObs.;
21     %let Item = %scan(&List.,&I.,&Delimiter.);
22     %Put *&MacroName.(&InVar. = &Item.);
23     %if not &Testing. %then
24         %&MacroName.(&InVar. = &Item.);
25 %end;
26 run; %Mend Demo;

```

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

```

1  30      %Demo(InData      = sashelp.Shoes
2  31      ,InVar              = Region
3  32      ,MacroName          = ReportTest
4  33      ,Testing            = 1);

```

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

```

1  List<Africa+Asia+Canada+Central America/Caribbean+Eastern Europe+Middle
2  East+Pacific+South America+United States+Western Europe>
3  *ReportTest (Region = Africa)
4  *ReportTest (Region = Asia)

```

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

```

1  34      options mprint; %* view macro statements;
2  35      %Demo(InData      = sashelp.Shoes
3  36      ,InVar              = Region
4  37      ,MacroName          = ReportTest
5  38      ,Testing            = 0);
6  MPRINT(DEMO): PROC SQL noprint ;
7  MPRINT(DEMO): select distinct Region into :List separated by "+" from
8  sashelp.Shoes;
9  MPRINT(DEMO): quit;

```

...and see the macro calls are correct.

```

1  *ReportTest (Region = Africa)
2  REPORTTEST REGION Africa
3  MPRINT(REPORTTEST): run;
4  *ReportTest (Region = Asia)
5  REPORTTEST REGION Asia
6  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 and Ex08, pg. 17.

```

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,

```

1  25      %Demo(InData      = sashelp.Shoes
2      26      ,InVar        = Region
3      27      ,MacroName    = ReportTest
4      28      ,Testing      = 1);
5  MPRINT(DEMO):  PROC SQL ;
6  MPRINT(DEMO):  select distinct Region as Item, cat( '%' ,
7  "ReportTest(Region=\"" , trim(Item), ')"') as CallMacro into :Item, :List
8  separated by ' ' from sashelp.Shoes;
9  MPRINT(DEMO):  quit;
10 NOTE: The PROCEDURE SQL printed page 1.

```

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

```

1  Item              CallMacro
2  -----
3  Africa            %ReportTest (Region=Africa)
4  Asia              %ReportTest (Region=Asia)
5  Canada            %ReportTest (Region=Canada)

```

When not testing

```

1  29      %Demo(InData      = sashelp.Shoes
2  30      ,InVar            = Region
3  31      ,MacroName        = ReportTest
4  32      ,Testing          = 0);
5  MPRINT(DEMO):  PROC SQL noprint ;
6  MPRINT(DEMO):  select distinct Region as Item, cat( '%' ,
7  "ReportTest(Region=\"" , trim(Item), ')"') as CallMacro into :Item, :List
8  separated by ' ' from sashelp.Shoes;
9  MPRINT(DEMO):  quit;

```

...the calls are executed:

```

1  MPRINT(DEMO):  run;
2  REPORTTEST REGION Africa
3  MPRINT(REPORTTEST):  run;
4  REPORTTEST REGION Asia
5  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, pg. 18.

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

```
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;
```

```
1  45 %Demo(InData      = Regions
2  46     ,InVar          = Region
3  47     ,MacroName      = ReportTest
4  48     ,Testing        = 1);
5
6  NOTE: The file TEMPFILE is:
7
8      File Name=D:\...\sas\zCallMacro.txt,
9      RECFM=V,LRECL=256
10
11 NOTE: 10 records were written to the file TEMPFILE.
12
13 Stmnt=%ReportTest(Region=Africa)
14 Stmnt=%ReportTest(Region=Asia)
15 Stmnt=%ReportTest(Region=Canada)
```

...and the TempFile:

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

When not testing the TempFile is %Included. 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.

```

1 40          write-calls-to-file-include.log
2 41          %Demo(InData    = Regions
3 42              ,InVar      = Region
4 43              ,MacroName  = ReportTest
5 44              ,Testing    = 1);
6
7 NOTE: The file TEMPFILE is:
8       File Name=D:\...\sas\zCallMacro.txt,
9       RECFM=V,LRECL=256
10
11 NOTE: 10 records were written to the file TEMPFILE.
12
13          write-calls-to-file-include.log
14
15 NOTE: %INCLUDE (level 1) file TEMPFILE is file
16       D:\...\sas\zCallMacro.txt.
17
18 49          +%ReportTest(Region=Africa)
19 REPORTTEST REGION Africa
20
21 50          +%ReportTest(Region=Asia)
22 REPORTTEST REGION Asia

```

CALL EXECUTE

The Call Execute data step routine is the successor to Write To File then 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, pg. 19.

```

1          write-calls-call-exec-demo.sas
2 %Include Project(make-unique-sort);
3 %Include Project(ReportTest);
4 %Macro Demo(InData    =
5              ,InVar    =
6              ,MacroName =
7              ,Testing   = 0);
8 DATA _Null_;
9 do until(EndoFile);
10    set &InData.    end = EndoFile;
11    call execute(
12        cats('%', "&MacroName.(&InVar.="
13            ,      &InVar.
14            ,')' %*end macro call;
15            ) %*end cats;
16            ); %*end execute;
17
18    end;
19 stop; run; %*calls executed here;
20 run; %Mend Demo;
21 %Demo(InData    = Regions
22       ,InVar     = Region
23       ,MacroName = ReportTest
24       ,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.

```

1 28      %Demo(InData    = Regions
2 29          ,InVar      = Region
3 30          ,MacroName   = ReportTest
4 31          ,Testing     = 1);
5
6 REPORTTEST REGION Africa
7 REPORTTEST REGION Asia

```

...here are the SAS statements:

```

1 NOTE: CALL EXECUTE generated line.
2 1      + run;
3 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`.

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, pg. 19.

```

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 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.

```

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.

```

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 the statements written to the log while testing.

```

1  Stmtnt=%ReportTest (Var=Region, Value=Africa)
2  Stmtnt=%ReportTest (Var=Region, Value=Asia)

```

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

```

1  NOTE: CALL EXECUTE generated line.
2  1      + %ReportTest (Var=Region, Value=Africa)
3  REPORTTEST VALUE Africa
4  REPORTTEST VAR Region
5  (where = (Region eq "Africa"))
6  MPRINT (REPORTTEST):  run;
7  2      + %ReportTest (Var=Region, Value=Asia)
8  REPORTTEST VALUE Asia
9  REPORTTEST VAR Region
10 (where = (Region eq "Asia"))
11 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.

```

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);

```

```

1  Stmtnt=%ReportTest (Var=Sex, Value=F)
2  Stmtnt=%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	Dilorio and Abolafia [5, sugi29.237] provide a dozen examples, Fehd [9, sgf2007.028] 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	Dilorio and Abolafia [5, sugi29.237] provide a thorough comparison of sashelp views and sql dictionary tables
SQL	Dilorio and Abolafia [5, sugi29.237] provide examples using sql; Fehd [7, sugi29.070] uses proc sql to create macro arrays; Fehd [9, sgf2007.028] 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 Dilorio 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/Proceedings/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 SAS Global Forum*, 2007. URL <http://www2.sas.com/proceedings/forum2007/028-2007.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 J. Fehd
Centers for Disease Control
4770 Buford Hwy NE
Atlanta GA 30341-3724

<mailto:RJF2@cdc.gov>

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 search <http://www.sascommunity.org> for the HOW List Processing zip.

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

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

<http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-1>
 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.

```
1 rem name: SAS.bat for either V8 or V9
2 if exist "C:\Program Files\SAS\SAS Learning Edition 4.1\sas.exe" "C:\Program Files\S
3 if exist "C:\Programs\SAS\SAS 9.1\SAS.exe" "C:\Program Files\SAS Institute\SAS\V8\SA
4 if exist "C:\Programs\SAS\SAS 9.1\SAS.exe" "C:\Programs\SAS\SAS 9.1\SAS.exe"
5 if exist "C:\Program Files\SAS\SAS 9.1\SAS.exe" "C:\Program Files\SAS\SAS 9.1\SAS.exe"
6 if exist "C:\Program Files\SAS\SASFoundation\SAS 9.2\SAS.exe" "C:\Program Files\SAS\
7 goto EOF
8 :EOF
```

```
1 /*name: SASv8.cfg
2 06Jan08 RJF2 *** */
3 -config 'C:\Program Files\SAS Institute\SAS\V8\SASv8.cfg'
4 -SASinitialFolder '.'
```

```
1 /*name: SASv9.cfg
2 06Jun08 RJF2 *** */
3 -SASinitialFolder '.'
```

```
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     formdlm = ' '              /* 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\Ppresentations\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);
```

```
----- 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;
```

```
----- 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, '1' );
14 call symputx('Count ', Count , '1' );
15 run;
16 %Put _global_;%* Show macro variables;
17
18 %Put Region1<&Region1.>;
19 *task: show 3rd region;
```

1. see: make-unique-sort, pg. 4
2. see: Report-Region-macro, pg. 3
3. see: ReportTest, pg. 3

1. see: Report-Africa, pg. 2

```
----- 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 %Let I = &SQLOBS.;
17 %Put Region&I.<&&Region&I..>;
18
19 *task: show last region;
20 %Put Region&SQLOBS.<&&Region&SQLOBS..>;
```

1. see: make-unique-sort, pg. 4

```

1      * Ex04.sas: vertical to vertical: ;
2      *          macro array, do loop;
3
4      options nosource2;%*echo Include to log;
5      %Include Project(Report-Region-macro);
6      %Include Project(ReportTest);
7
8      options mprint;
9      %macro Demo;
10     %local I;
11
12     PROC SQL noprint;
13         select distinct   Region
14         into   :Region1 - :Region999
15         from   sashelp.Shoes;
16         quit;
17
18     %do I = 1 %to &SqlObs.;
19         %put *Report(Region = &&Region&I.);
20     %end;
21 %mend Demo;
22 %Demo;
23
24 *task      : execute macro Report;
25 *solution: see make-array-sql-into;

```

1. see: Report-Region-macro pg. 3

2. see: ReportTest, pg. 3

see: make-array-sql-into

1. see: Report-Region-macro, pg. 3

2. see: ReportTest, pg. 3

```

1      * Ex05.sas: vertical to horizontal: ;
2      *          write list to macro var;
3
4      PROC SQL noprint;
5          select distinct Region
6          into           :Regions
7          separated by '+'
8          from   sashelp.Shoes;
9          quit;
10
11     %put Regions |&Regions.;
12
13     *task      : use macro function scan;
14     *          show 5th region;
15     *put region5(scan(Regions,...));
16     *solution: see Ex06;

```

see: write-list-into-mvar-scan, pg. 6

```

1      * Ex06.sas: vertical to horizontal: ;
2      *          write list to macro var, scan;
3
4      options nosource2;%*echo Include to log;
5      %Include Project(Report-Region-macro);
6      %Include Project(ReportTest);
7
8      options mprint;
9      %macro Demo;
10     %local I Item;
11
12     PROC SQL noprint;
13         select distinct Region
14         into   :Regions separated by '+'
15         from   sashelp.Shoes;
16         quit;
17
18     %put Regions: |&Regions.;
19     %do I = 1 %to &SqlObs.;
20         %let Item = %scan(&Regions., &I., +);
21         %put *ReportTest(Region = &Item.);
22     %end;
23 run;%mend Demo;
24 %Demo
25
26 *task      : execute macro ReportTest;
27 *solution: see write-list-into-mvar-scan;

```

```

1      * Ex07.sas: vertical to horizontal: ;
2      *      write macro calls      ;
3      *      into macro var;
4
5      options  nosource2;%*echo Include to log;
6      %Include Project(Report-Region-macro);
7      %Include Project(ReportTest);
8
9      PROC SQL noprint;
10         create table      Regions as
11         select distinct Region
12         from sashelp.Shoes;
13         select cats( '*'
14                     , 'Report(Region='
15                     ,      Region
16                     ,      ')'
17                     )
18         into :List separated by ' '
19         from Regions;
20         quit;
21
22 %Put List<&List.>;
23
24 *task      : execute macro ReportTest;
25 *solution: see Ex08;

```

```

1      * Ex08.sas: vertical to horizontal;
2      *      write macro calls      ;
3      *      into macro var, execute;
4
5      options  nosource2;
6      %Include Project(Report-Region-macro);
7      %Include Project(ReportTest);
8
9      PROC SQL noprint;
10         create table      Regions as
11         select distinct Region
12         from sashelp.Shoes;
13         select cats( '%'
14                     , 'ReportTest(Region='
15                     ,      Region
16                     ,      ')'
17                     )
18         into :List separated by ' '
19         from Regions;
20         quit;
21
22 &List.;%*execute calls;

```

```

1      * Ex09.sas
2      *      add prefix and suffix;
3      *      write to file, include;
4
5      options  nosource2;%*echo Include to log;
6      %Include Project(make-unique-sort);
7      %Include Project(Report-Region-macro);
8      %Include Project(ReportTest);
9      options  mprint;
10
11 filename TempFile 'zCallMacroX.txt';
12
13 DATA _Null_;
14 attrib Stmnt length = $100;
15 file TempFile;
16
17 do until(EndoFile);
18     set Regions end = EndoFile;
19     Stmnt = cats( '*'
20                 , 'ReportTest(Region='
21                 ,      Region
22                 ,      ')' );
23     putlog Stmnt=;
24     put Stmnt;%*to file;
25     end;%* do until;
26 stop; run;
27
28 %Include TempFile/source2;
29 filename TempFile clear;
30 run;
31
32 *task      : execute macro ReportTest;
33 *solution: see write-calls-to-file-include;

```

see: write-calls-to-file-include, pg. 9

```

1  * Ex10.sas: vertical: ;
2  *      add prefix and suffix;
3  *      call execute without nrstr;
4
5  options  nosource2;%*echo Include to log;
6  %Include Project(make-unique-sort);
7  %Include Project(Report-Region-macro);
8  %Include Project(ReportTest);
9  %* NOTE: not works correctly!;
10 options  mprint;
11
12 DATA _Null_;
13 set  Regions;
14 call execute('%ReportTest(Region='
15              || Region || ' '));
16 *call execute('%Report      (Region='
17              || Region || ' '));
18 run;%*calls executed here;
19 run;

```

See explanation at Caveat, pg. 10

see: write-calls-call-execute, pg. 11

```

1  * Ex11.sas: vertical: ;
2  *      add prefix and suffix;
3  *      call execute with nrstr;
4
5  options  nosource2;%*echo Include to log;
6  %Include Project(make-unique-sort);
7  %Include Project(Report-Region-macro);
8  %Include Project(ReportTest);
9  %Include Project(ReportTestVarValue);
10 options  mprint;
11
12 DATA _Null_;
13 set  Regions;
14 call execute(
15     cats(
16         '%nrstr('
17             , '%ReportTest(Region='
18                 , Region
19                 , ' ')*%end macro;
20             , ')*'
21             , '%end nrstr;
22             , '%end cats;
23             , '%end execute;
24 run;%*calls executed here;
25 run;
26
27 *task: execute ReportTest
28 *      with the Var=
29 *      Value= parameters;
30 *solution: see write-calls-call-execute;
31 *solution: see Ex11-solution;

```

SOLUTIONS TO EXAMPLES

```

18      _____ S02.sas _____
19      %Put Region1<&Region1.>;
20
21      *task: show 3rd region;
22      %Put Region3<&Region3.>;

```

```

14      _____ S03.sas _____
15      %Let I = 3;
16      %Put Region&I.<&&Region&I..>;
17
18      *task: show last region;
19      %*solution 1;;
20      %Let I = &SqlObs.;
21      %Put Region&I.<&&Region&I..>;
22
23      %*solution 2;;
24      %Put Region&SqlObs.<&&Region&SqlObs..>;

```

```

18      _____ S04.sas _____
19      %do I = 1 %to &SqlObs.;
20          *task: execute macro Report;;
21          %put *Report (Region = &&Region&I.);
22          %Report (Region = &&Region&I.);
23      %end;

```

```

13      _____ S05.sas _____
14      *task      : use macro function scan;
15      *          show 5th region;
16      %put reg5(%scan(&Regions,5,+));

```

```

19      _____ S06.sas _____
20      %do I = 1 %to &SqlObs.;
21          %let Item = %scan(&Regions., &I., +);
22          *task: execute macro ReportTest;
23          %put *ReportTest (Region = &Item.);
24          %ReportTest (Region = &Item.);
25      %end;

```

```

19      _____ S09.sas _____
20      %*task      : execute macro ReportTest;
21      %*Stmnt = cats( ' *';
22      Stmnt = cats( '% '
23                  , 'ReportTest (Region='

```

```

14      _____ S11.sas _____
15      %*          , '%ReportTest (Region=';
16      call execute(
17          cats(
18              '%nrstr('
19              , '%ReportTest (Var=Region, Value='
20              , Region

```

```

1      _____ S02.log _____
2      26      *task: show 3rd region;
3      27      %Put Region3<&Region3.>;
4      Region3<Canada>

```

```

1      _____ S03.log _____
2      18      %*solution 1;;
3      19      %Let I = &SqlObs.;
4      20      %Put Region&I.<&&Region&I..>;
5      Region10<Western Europe>

```

```

1      _____ S04.log _____
2      *Report (Region = Africa)
3      MPRINT(REPORT): ods html file="Africa.html";
4      NOTE: Writing HTML Body file: Africa.html

```

```

1      _____ S05.log _____
2      15      %put reg5(%scan(&Regions,5,+));
3      reg5(Eastern Europe)

```

```

1      _____ S06.log _____
2      *ReportTest (Region = Africa)
3      REPORTTEST REGION Africa
4      MPRINT(REPORTTEST): run;
5      MPRINT(DEMO): ;

```

```

1      _____ S09.log _____
2      76      +%ReportTest (Region=Africa)
3      REPORTTEST REGION Africa
4      MPRINT(REPORTTEST): run;

```

```

1      _____ S11.log _____
2      1      + %ReportTest (Var=Region, Value=Africa)
3      REPORTTEST VALUE Africa
4      REPORTTEST VAR Region
5      (where = (Region eq "Africa"))
6      MPRINT(REPORTTEST): run;

```