Using Macro Variable Lists to Create Dynamic Data-Driven Programs

JOSHUA M. HORSTMAN
NESTED LOOP CONSULTING
SEPTEMBER 28, 2023

1

Introduction

- The term "macro variable list" refers to a way of using macro variables, not a distinct language element.
- Macro variables are used to store a series of values dynamically, typically originating from our data.
- Macro variable lists allow us to:
  - Build dynamic programs
  - Create data-driven programming logic
  - Eliminate hard-coded data dependencies
  - Confuse our co-workers and ensure job security!

2

2

Section 1

MACRO LANGUAGE REVIEW

3

Macro Processing: A Gross Simplification

- When a SAS program is submitted:
  - Word scanner parses statements into tokens.
  - Tokens are sent to compiler for syntax checking.
  - Execution occurs when step boundary is reached.
- If the word scanner detects macro triggers (% or &):
  - Macro elements routed to macro processor.
  - Macro variables resolved and macro statements executed.
  - Output from macro processor must be rescanned for additional macro language elements.

4

4

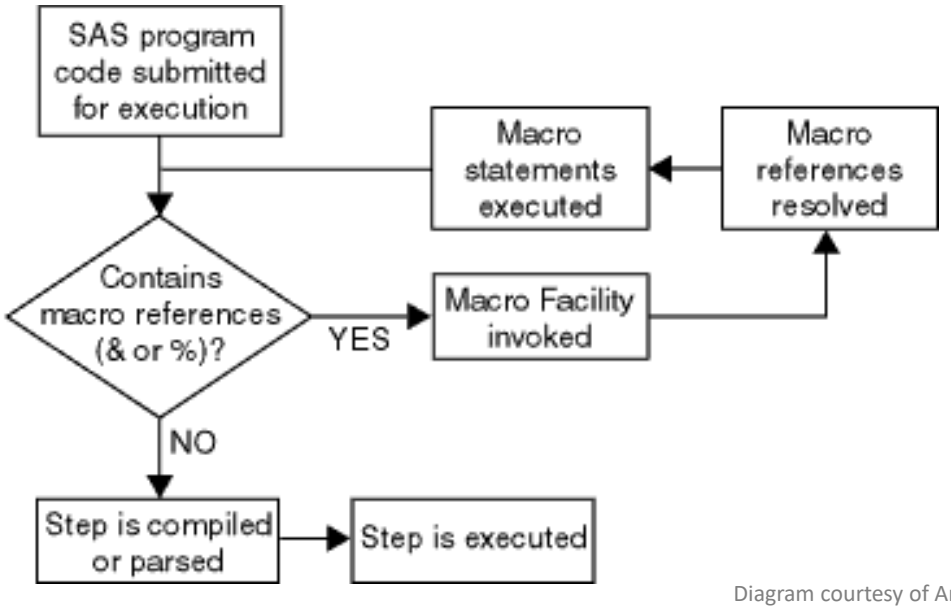


Diagram courtesy of Art Carpenter. Used with permission.

5

5

Creating Macro Variables using %LET

- Assigning a value to a macro variable:

```
%let output_path = C:\temp;
```

- Subsequent references to macro variable replaced with value by macro processor:

```
filename myfile "&output_path\myfile.txt";
```

becomes

```
filename myfile "C:\temp\myfile.txt";
```

6

6

Limitations of %LET

VIEWTABLE: Sashelp.Class (Student Data)

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84

- Macro processor assigns value before SAS code executes.

```
data _null_;
  set sashelp.class;
  where name='Alfred';
  %let alfred_age = age;
run;
```

This will not have the desired effect.

- Macro variable **alfred_age** is literally assigned the value "**age**".
- SAS compiler only sees this:

```
data _null_;
  set sashelp.class;
  where name='Alfred';
run;
```

7

7

Creating Macro Variables at Execution Time using the DATA step

- SYMPUTX routine assigns macro variable values during DATA step:

```
data _null_;
  set sashelp.class;
  where name='Alfred';
  call symputx("alfred_age",age);
run;
```

Macro variable name

Value to be assigned

- Macro variable **alfred_age** will be assigned the value "**14**".

8

8

Creating Macro Variables at Execution Time using PROC SQL

- INTO clause assigns macro variable values during PROC SQL:

```
proc sql noprint;
  select age
    into :alfred_age
    from sashelp.class
    where name='Alfred';
quit;
```

Value to be assigned

Macro variable name

- Macro variable **alfred_age** will be assigned the value "**14**".

9

9

Section 2

CREATING MACRO VARIABLE LISTS

10

Horizontal vs. Vertical Macro Variable Lists

- Horizontal list: a list of values in a single macro variable

```
%let origin_list = Asia Europe USA;
```

- Choose your delimiter carefully:

```
%let origin_list = Asia~Europe~USA;
```

- Vertical list: a separate macro variable for each item

```
%let origin1 = Asia;
%let origin2 = Europe;
%let origin3 = USA;
```

- We want to create these dynamically, not by hard-coding!

11

11

Creating a Vertical Macro Variable List Using the DATA Step

```
proc sort data=sashelp.cars
    out=unique_origins(keep=origin) nodupkey;
  by origin;
run;

data _null_;
  set unique_origins end=eof;
  call symputx(cats('origin',_n_),origin);
  if eof then call symputx('numorigins',_n_);
run;
```

```
11   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

12

12

Creating a Horizontal Macro Variable List Using the DATA Step

```
data _null_;
  set unique_origins end=eof;
  length origin_list $200;
  retain origin_list;
  origin_list = catx('~',origin_list,origin);
  if eof then do;
    call symputx('origin_list',origin_list);
    call symputx('numorigins',_n_);
  end;
run;
```

```
30   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN_LIST Asia~Europe~USA
```

13

13

Creating a Vertical Macro Variable List Using PROC SQL

```
proc sql noprint;
  select distinct origin into :origin1-
    from sashelp.cars
   order by origin;
  %let numorigins = &sqlobs;
quit;
```

```
11   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

14

14

Creating a Horizontal Macro Variable List Using PROC SQL

```
proc sql noprint;
  select distinct origin
    into :origin_list separated by '~'
    from sashelp.cars
   order by origin;
  %let numorigins = &sqlobs;
quit;
```

```
30   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN_LIST Asia~Europe~USA
```

15

15

Section 3

USING MACRO VARIABLE LISTS

16

```
30   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN_LIST Asia~Europe~USA
```

Using Horizontal Macro Variable Lists

- Access individual list elements using %SCAN function:

```
%scan(&origin_list,1,~)   → Resolves to: Asia
%scan(&origin_list,2,~)   → Resolves to: Europe
%scan(&origin_list,3,~)   → Resolves to: USA
```

- Use loop counter as index for %SCAN function:

```
%do i = 1 %to &numorigins;
  %put Item &i: %scan(&origin_list,&i,~);
%end;
```

```
Item 1: Asia
Item 2: Europe
Item 3: USA
```

17

17

Using Vertical Macro Variable Lists

- Access individual list elements using macro variable reference:

```
&origin1   → Resolves to: Asia
&origin2   → Resolves to: Europe
&origin3   → Resolves to: USA
```

```
11   %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

- Cannot use **&origin&i**
- Macro processor interprets this as two macro variable references:
- Macro variable **origin** does not exist.

18

18

Using Vertical Macro Variable Lists

- Instead, use **&&origin&i**.

Original:	**&&origin&i**	
1st pass:	**&origin1**	(&& resolves to &, origin is just text, &i resolves to 1)
2nd pass:	**Asia**	(resolved value of macro variable origin1)

- Use in a loop:

```
%do i = 1 %to &numorigins;
  %put Item &i: &&origin&i;
%end;
```

```
Item 1: Asia
Item 2: Europe
Item 3: USA
```

19

19

Section 4

DATA-DRIVEN PROGRAMMING EXAMPLES

20

Example #1: Splitting a Data Set (1 of 2)

```
%macro split_data;

  proc sql noprint;
    select distinct origin into :origin1-
      from sashelp.cars;
    %let numorigins = &sqlobs;
  quit;
```

GOAL: Split SASHELP.CARS into a separate data set for each value of ORIGIN.

Create a vertical macro variable list

21

21

Example #1: Splitting a Data Set (2 of 2)

```
  %do i = 1 %to &numorigins;
    data cars_&&origin&i;
      set sashelp.cars;
      where origin = "&&origin&i";
    run;
  %end;

%mend split_data;
```

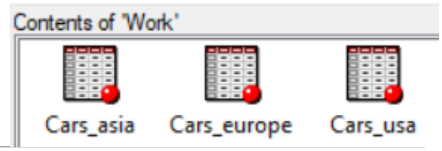
Loop through each value.

Generate a DATA step to create corresponding subset.

22

22

Example #1: The Result




```
NOTE: There were 158 observations read from the data set SASHELP.CARS.
      WHERE origin='Asia';
NOTE: The data set WORK.CARS_ASIA has 158 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.03 seconds


NOTE: There were 123 observations read from the data set SASHELP.CARS.
      WHERE origin='Europe';
NOTE: The data set WORK.CARS_EUROPE has 123 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds


NOTE: There were 147 observations read from the data set SASHELP.CARS.
      WHERE origin='USA';
NOTE: The data set WORK.CARS_USA has 147 observations and 15 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```


Example #2: Dynamic Report Creation (1 of 2)

```
%macro graph_stocks;
  proc sql noprint;
    select distinct stock
      into :stock_list separated by '~'
      from sashelp.stocks;
    %let numstocks = &sqlobs;
  quit;
```

GOAL: Create separate plot for each value of STOCK in SASHELP.STOCKS data set.

Create a horizontal macro variable list

Example #2: Dynamic Report Creation (2 of 2)

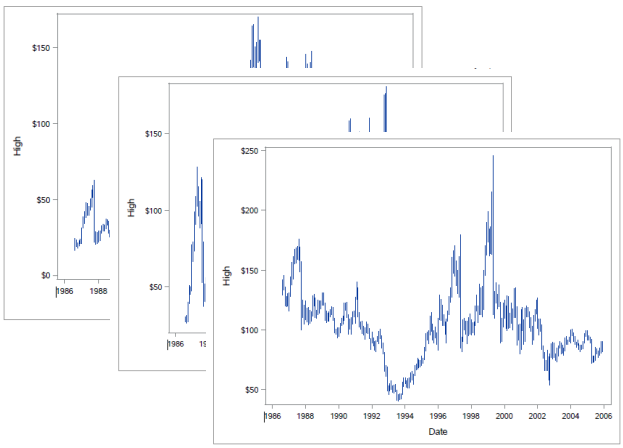
```
%do i = 1 %to &numstocks;
  ods pdf file="%scan(&stock_list,&i,~).pdf";
    proc sgplot data=sashelp.stocks;
      where stock = "%scan(&stock_list,&i,~)";
      highlow x=date high=high low=low;
    run;
  ods pdf close;
%end;
%mend graph_stocks;
```

Generate an SGPLOT with ODS PDF statements for each value in the list.

25

25

Example #2: The Result



26

26

Example #3: Comparing Multiple Data Sets (1 of 2)

```
%macro compare_all(lib1,lib2);
  proc sql noprint;
    select distinct a.memname into :ds1-
      from dictionary.tables a, dictionary.tables b
    where a.libname = upcase("&lib1")
        and b.libname = upcase("&lib2")
        and a.memname = b.memname and a.memtype = "DATA";
    %let numds = &sqlobs;
  quit;
```

GOAL: Compare all data sets common to two libraries.

Create a vertical macro variable list

27

27

Example #3: Comparing Multiple Data Sets (2 of 2)

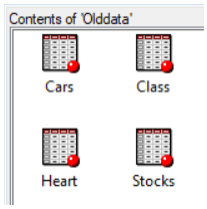
```
 %do i = 1 %to &numds;
    proc compare
      base    = &lib1..&&ds&i
      compare = &lib2..&&ds&i;
    run;
  %end;
%mend compare_all;
```

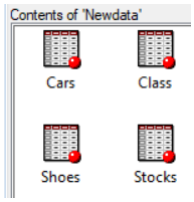
Generate the PROC COMPARE for each data set in the list.

28

28

Example #3: The Result






```
55   %compare_all(olddata,newdata)
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


NOTE: There were 428 observations read from the data set OLDDATA.CARS.
NOTE: There were 428 observations read from the data set NEWDATA.CARS.
NOTE: PROCEDURE COMPARE used (Total process time):
      real time           0.05 seconds
      cpu time            0.01 seconds


NOTE: There were 19 observations read from the data set OLDDATA.CLASS.
NOTE: There were 19 observations read from the data set NEWDATA.CLASS.
NOTE: PROCEDURE COMPARE used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


NOTE: There were 699 observations read from the data set OLDDATA.STOCKS.
NOTE: There were 699 observations read from the data set NEWDATA.STOCKS.
NOTE: PROCEDURE COMPARE used (Total process time):
      real time           0.06 seconds
      cpu time            0.03 seconds
```

29

29

Conclusion

- Macro variable lists are a powerful tool.
- Use them to build robust programs:
  - Include dynamic logic
  - Avoid hard-coding
  - Adapt to changes in data or computing environment
- Advantages:
  - Less likely to require change
  - Easier to maintain
  - Greater potential for reuse

30

30

Contact Information:

Joshua M. Horstman
Nested Loop Consulting LLC
josh@nestedloopconsulting.com

31