Paper SS-09

# Introduction to SAS® Macro Language

John Myers, Virginia Commonwealth University, Richmond, VA


## ABSTRACT

A step by step process is presented to develop programs using the SAS macro language. Examples are given to create macro variables, to build macro definitions and to use conditional and iterative processing. This presentation would be useful to those who have some experience with base SAS and who want to begin using the SAS macro language.


## INTRODUCTION

The SAS macro language provides statements that you can use to modify your source program. It allows you to write statements that change your source program at run time.

I use the SAS macro language when I have repetitive tasks. For example, if I have a program that I need to run for different variables or different data sets, I would write a macro since I can use it repetitively and it allows me to easily change variable names or data set names. If I need to run a program two or three times, I would just repeat the code. When I repeat program code, I need to be careful to change the variable names or data set names every time they are used in the program. However, if I have to run a program ten or twenty times, I would write a macro. You need to decide when you need a macro. You want to keep your program simple because it takes more time to write, test and debug a program that uses the macro language.

When I started to use the macro language, I thought that all the SAS macro statements were resolved before the program began to execute but that is not the case. When you run your SAS program, the source statement processor starts at the beginning of your program and it reads and checks your program statements. If the source statement processor finds a macro statement then it transfers control to the macro statement processor. The macro processor reads and checks the macro statements. When macro processor is finished, it returns control to the source statement processor. When the source statement processor reaches a “step boundary” then SAS executes that one step of your program. After that step executes, control returns to the source statement processor and the process repeats. A step boundary is a RUN, DATA, or PROC statement. You should always use RUN statements when you write programs with macros. If you do not, the macro could terminate and SAS could still be looking for a step boundary.

## DEVELOPMENT PROCESS

The following diagram gives a step by step process to develop a program using the SAS macro language.

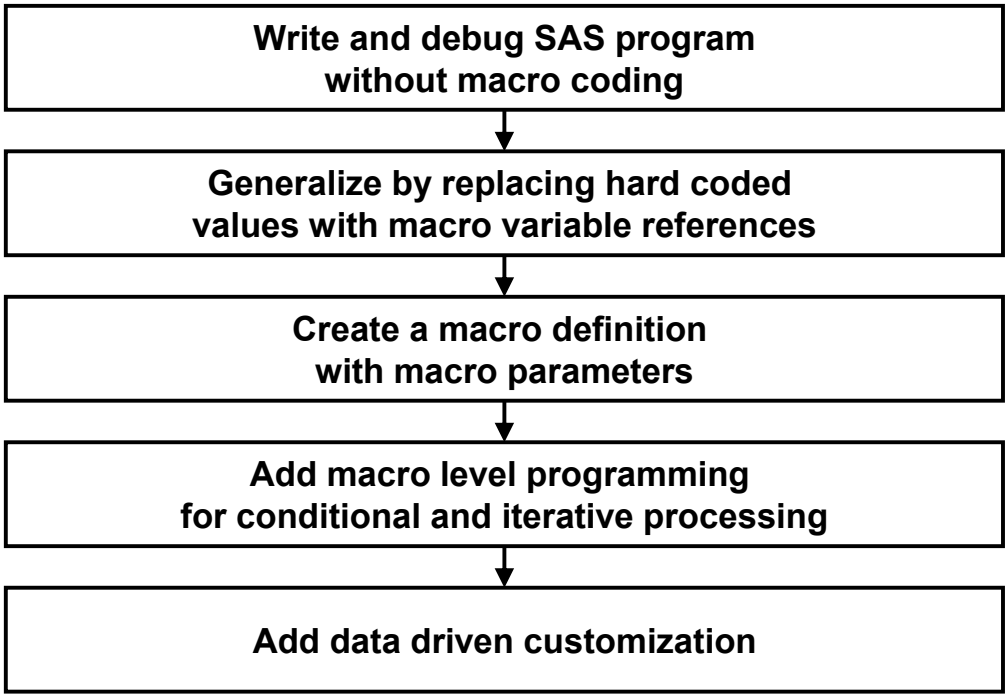



**Figure 1**

Figure 1 shows a step by step process to develop a program using the SAS macro language.

#### STEP 1. WRITE AND DEBUG YOUR SAS PROGRAM WITHOUT MACRO CODING

This means that you write and debug your source program for one iteration. In my case it would be for one set of variables or for one data set. You need to debug the source program first because the error messages for both the macro statements and the source statements are printed on the log file and it gets very confusing if you have error messages from both the source statements and macro statements.

#### STEP 2. GENERALIZE BY REPLACING HARD CODED VALUES WITH MACRO VARIABLE REFERENCES

This is where you define your macro variables.

#### STEP 3. CREATE A MACRO DEFINITION WITH MACRO PARAMETERS

This is where you choose what source statements to put in the macro.

#### STEP 4. ADD MACRO LEVEL PROGRAMMING FOR CONDITIONAL AND ITERATIVE PROCESSING

This is where you add macro statements that will run different source statements using %IF statements or repeat source statements using %DO statements.

**STEP 5. ADD DATA DRIVEN CUSTOMIZATION**

This is where you add macro statements to run different source statements depending on the data. When I first learned about the SAS macro language, I thought it was dangerous because it can generate a different program every time it runs. You need to limit the number of possible programs that can be generated, to test the program very carefully and to check the log every time it runs to make sure it did what you wanted it to do.

**Marketing Data**

| region | city | product | sales | revenue |
|---|---|---|---|---|
| Northeast | New York | sedan | 400 | 100 |
| Northeast | Boston | coupe | 800 | 200 |
| Midwest | Chicago | minivan | 200 | 50 |
| Midwest | Detroit | suv | 100 | 25 |
| Northwest | Seattle | sedan | 600 | 150 |
| Northwest | Boise | coupe | 20 | 5 |
| Southeast | Raleigh | minivan | 200 | 50 |
| Southeast | Atlanta | suv | 400 | 100 |
| Southwest | Dallas | sedan | 1000 | 250 |
| Southwest | Houston | coupe | 2000 | 500 |

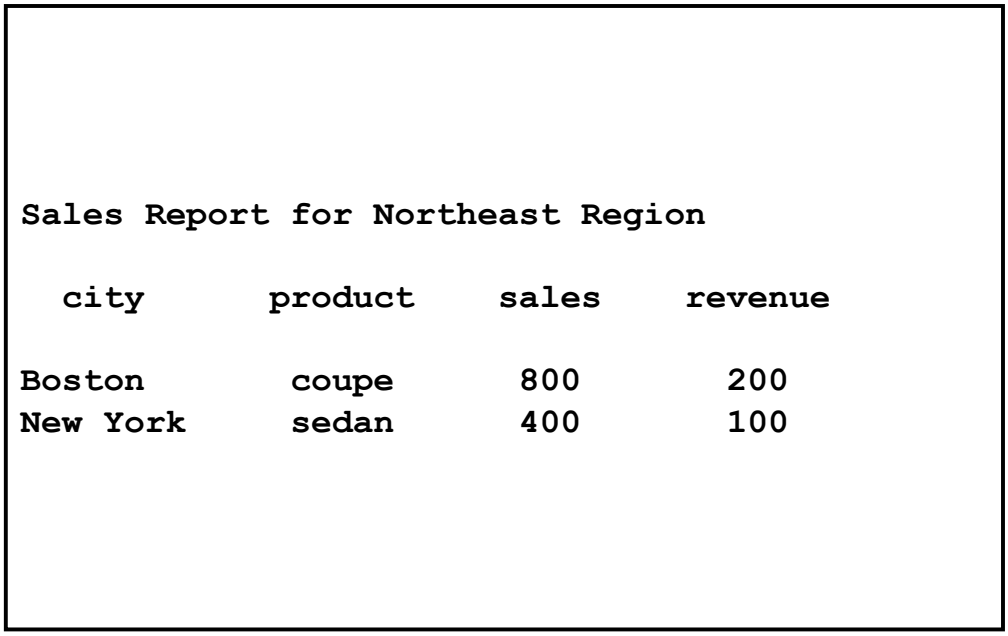
## Figure 2

Figure 2 shows test data for the example program. This data set contains marketing data for a number of products with the number of sales at different locations.

**WRITE AND DEBUG SAS PROGRAM WITHOUT MACRO CODING**

```
proc sort data=lib1.marketing (where=(region="Northeast")) ❶
  out=sales;
by city product;
run;
title1 "Sales Report for Northeast Region"; ❷
proc print data=sales noobs;
var city product sales revenue;
run;
```

This example shows the source program without macro coding. The first step is a PROC SORT with a WHERE clause ❶ that selects a subset of the data for the Northeast region. The second step is a PROC PRINT with a TITLE statement ❷ that produces the sales report for the Northeast region.



Sales Report for Northeast Region

| city | product | sales | revenue |
|---|---|---|---|
| Boston | coupe | 800 | 200 |
| New York | sedan | 400 | 100 |

**Figure 3**

Figure 3 shows the report for the example program without macro coding. This report gives the sales for the Northeast region.

### GENERALIZE BY REPLACING HARD CODED VALUES WITH MACRO VARIABLE REFERENCES

```
%let region = Northeast; ❶
%let sortvar = city product; ❷
proc sort data=lib1.marketing (where=(region="&region")) ❸
  out=sales;
by &sortvar; ❹
run;
title1 "Sales Report for &region Region"; ❺
proc print data=sales noobs;
var &sortvar sales revenue; ❻
run;
%put region=&region sortvar=&sortvar; ❼
```

This example shows how to create macro variables. Macro variables are used to substitute text in your source program. The %LET statement is used to define macro variables. The %LET statement can be used anywhere in your program in macros or in “open code” (outside macros). In the first %LET statement ❶, I define a macro variable named REGION. All macro variables have character values. In the %LET statement, the value is all the characters from the equal sign to the semicolon. Do not use quotation marks before and after the value in the %LET statement. If you do, the quotation marks become part of the value. Upper and lower case letters are preserved in macro variable values.

Every time the macro variable &REGION appears in the program the value “Northeast” will be substituted for it. The macro variable &REGION appears in the WHERE clause ❸ and in the TITLE statement ❺. In the TITLE statement, you need to use double quotes around the value. If you use single quotes, SAS will not substitute the macro variable value. Macro variables and data variables can have the same names because the values are stored in different places. Macro variable values are stored in the symbol table and data variable values are stored in the data vector.

In the second %LET statement ❷, I define a macro variable named SORTVAR. Blank spaces in the middle of a macro variable value are preserved but SAS trims spaces before and after the text. Every time the macro variable &SORTVAR appears in the program the value “city product” will be substituted for it. The macro variable &SORTVAR appears in the BY statement ❹ and in the VAR statement ❻.

The SAS system option SYMBOLGEN can be used to print the macro variable value on the log each time a macro variable is resolved. Alternatively, you can use the %PUT statement to write a message to the log file. Since %PUT is a macro statement, you do not use quotation marks around the text. The %PUT statement ❼ displays the names and values of the macro variables on the log file. The %PUT statement is used to debug your program. There is no %FILE statement, so you cannot create a report with the %PUT statement.

## CREATE A MACRO DEFINITION WITH MACRO PARAMETERS

```
%macro sales (region=, sortvar=city product); ❶
proc sort data=lib1.marketing
  (where=(region="&region")) out=sales;
by &sortvar;
run;
title1 "Sales Report for &region Region";
proc print data=sales noobs;
var &sortvar sales revenue;
run;
%put region=&region sortvar=&sortvar;
%mend sales; ❷
%sales (region=Northeast); ❸
```

This example shows how to create a macro definition. A macro definition starts with the %MACRO statement ❶ and ends with the %MEND statement ❷. A macro definition can contain source statements and macro statements. The macro name SALES is given on the %MACRO statement and the %MEND statement. When SAS reads your macro, it is compiled but is not run until you call it. To call a macro, you put a % sign in front of the macro name as in the %SALES statement ❸.

The %MACRO statement can have macro parameters in parenthesis after the macro name. Macro parameters are macro variables that you define on the %MACRO statement. Macro parameters can be positional parameters or keyword parameters. Keyword parameters are followed by an equal sign and may be given default values on the %MACRO statement. On the call statement, keyword parameters can be given in any order but you need to give both the name and the value for each parameter. Positional parameters do not have any equal signs on the %MACRO statement and you cannot give default values. On the call statement, positional parameter values must be given in the same order as they were defined on the %MACRO statement and you do not use the parameter names. If you use both positional and keyword parameters, the positional parameters must be given first in the list. It is good practice to always use keyword parameters because they more clearly document your program.

## ADD MACRO LEVEL PROGRAMMING FOR CONDITIONAL AND ITERATIVE PROCESSING

```
%macro sales (region=, sortvar=city product) / minoperator; ❶
%let reglist = Northeast Midwest Northwest Southeast Southwest; ❷
%if &region in &reglist %then ❸
  %do; ❹
  proc sort data=lib1.marketing (where=(region="&region")) out=sales;
  by &sortvar;
  run;
  title1 "Sales Report for &region Region";
  proc print data = sales noobs;
  var &sortvar sales revenue;
  run;
  %end;
%else %put Invalid value for region=&region; ❺
%mend sales;
%sales (region=Northeast);
```

This example shows macro level programming for conditional and iterative processing. In this example, I check the macro variable &REGION to contain a valid value. The %LET statement ❷ assigns a list of valid values to the macro variable &REGLIST. The %IF statement ❸ checks &REGION to contain a value in &REGLIST using the IN operator. When you use the IN operator you need to put the MINOPERATOR option on the %MACRO statement ❶. If the condition is true, the %THEN clause is executed and the %DO group ❹ runs the PROC SORT and PROC PRINT steps. If the condition is false, the %ELSE clause ❺ is executed and the %PUT statement prints a message on the log. In this example, the report will print only if the value of &REGION is in the list of valid values.

## ADD DATA DRIVEN CUSTOMIZATION

```
%macro sales (region=,sortvar=city product) / minoperator;
proc sql noprint; ❶
  select distinct region
  into :reglist separated by " " ❷
  from lib1.marketing;
  quit;
%put reglist=&reglist; ❸
%if &region in &reglist %then ❹
  %do; ❺
  proc sort data=lib1.marketing (where=(region="&region")) out=sales;
  by &sortvar;
  title1 "Sales Report for &region Region";
  proc print data = sales noobs;
  var &sortvar sales revenue;
  run;
  %end;
%else %put No customers on file from region=&region; ❻
%mend sales;
%sales (region=Northeast);
```

This example shows data driven customization. In this example, I check the macro variable &REGION to contain a value which appears in the marketing data set. The PROC SQL step ❶ gets all the distinct values from data variable REGION and assigns these values to the macro variable REGLIST. The INTO clause ❷ in PROC SQL is used to build macro variable REGLIST. In the INTO clause, the macro variable name is preceded with a colon instead of an ampersand. The %PUT statement ❸ prints the macro variable name and value on the log. The %IF statement ❹ checks &REGION to contain a value in &REGLIST using the IN operator. If the condition is true, the %THEN clause is executed and %DO group ❺ runs the PROC SORT and PROC PRINT steps. If the condition is false, the %ELSE clause ❻ is executed and the %PUT statement prints a message on the log. In this example, the report will print only if there are observations with the value of &REGION in the data set.

## CONCLUSION

This paper presented a step by step process to develop programs that use the SAS macro language. Examples were given to create macro variables, to build macro definitions and to use conditional processing and iterative processing. You want to keep your program simple because it takes more time to write, test and debug a program that uses the macro language.

## REFERENCES

1. Art Carpenter, “Carpenter’s Complete Guide to the SAS Macro Language Second Edition” , 2004, SAS Institute, Cary, NC

2. Delwiche, LD and SJ Slaughter, “The Little SAS Book”, 2008, Fourth Edition, SAS Institute Inc., Cary, NC.

3. SAS e-learning course, “Macro Language 1: Essentials”, SAS Institute Inc., Cary, NC.

## CONTACT INFORMATION

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

| | |
|---|---|
| Name: | John Myers |
| Enterprise: | Virginia Commonwealth University |
| Address: | 800 East Leigh Street |
| City, State ZIP: | Richmond, VA 23219 |
| Work Phone: | 804-828-8108 |
| E-mail: | jmmyers@vcu.edu |