SAS Macro to Expedite Data Manipulation

Bhavya Chaudhary
Software Engineer, Snapdeal
New Delhi, India

Nikita Malik
Research Scholar, IP University
New Delhi, India

Dhwani Mohan
Information Technology, IP University
New Delhi, India

ABSTRACT

One of the most powerful and dynamic feature of SAS language is Macro facility. The facility reduces the amount of effort required to read and write SAS code. Macro generates the SAS code automatically and enables one to acquire the skill set to become an efficient SAS programmer. The use of macro has prevailed over time and has deprecated the need to write repetitive codes. With the help of this feature one small change can be made to echo throughout the program in no time. Macro presents a uniform and easy approach for manipulation of data. In this research we explain the basic fundamentals of macro facility to generate data-driven programs .Also, how to create and use macro variables and save them for future aspects.

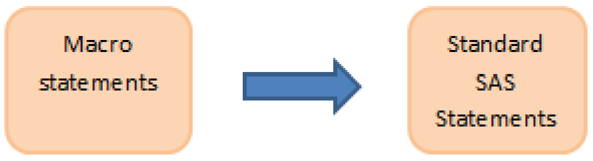
Keywords

Macro Variable, Macro Defination, Symput, Data, Sql, Error And Debugging

1. INTRODUCTION

This paper will discuss an easy approach to bridge the gap between a beginner and an expert SAS programmer. The macro facility comes as a part of base SAS that can be used to enhance and customize the code and make it more efficient. It is used to create, modify and write data with reduced requirement of the text strings manipulations. Macro can make the development and maintenance of production programs much easier. In some cases, macro may slow down the program due to an additional step before compilation and execution. However, the facility reduces the amount of coding to accomplish a task by providing modularized codes. It does make the program more complex but the benefits make its usage prevalent. This paper will not cover every aspect of the facility. However, it will help the novice to gain familiarity with the topic.

2. MACRO PROCESSOR



METAPROGRAMMING

When a program is written SAS compiles it and then executes it immediately. But when a macro is written, there is an additional step. Before the SAS code compiles, the macro statements are passed to a macro processor. The task of this macro processor is to resolve the macro to standard SAS code. This step is called meta-programming because a program is made to write another program.

3. MACRO AND MACRO VARIABLES

3.1 MACRO

The name of macro is prefixed with a percent sign (%). It is the large piece of program with complex logic encompassing DATA and PROC steps and macro statements like %DO, %END, %IF.

3.2 MACRO VARIABLES

The name of macro variables are prefixed with an ampersand sign (&). A macro variable is like standard data variable having single character value but it does not belong to data step.

4. WAYS TO CREATE MACRO VARIABLE

4.1 %LET

Creating a macro variable

The simplest and the most useful way to define a macro variable is through %LET. It is assigned statement that work like DATA step.

Syntax:
%LET macro-variable = text-string;
Here, %LET is followed by the name of macro variable, an equal sign (=) and the text string which is to be placed in the variable. The text string is neither character nor numeric but plain text. The text does not require quotation marks. Everything that we write on the right side of the equal sign is assigned to the variable.

Using a macro variable:

To use a macro variable add the ampersand prefix (&) to the macro variable. The processor does not resolve the macro in single quotation marks therefore we make use of double quotation marks.

%LET loop=10;

%LET heading=Macros;

1. Do i=1 to &loop;
   TITLE 'Name of the topic is &heading';

2. Do i=1 to &loop;
   TITLE "Name of the topic is &heading";

After being resolved by the processor, these statements would become

1. Do i=1 to 10;
   TITLE 'Name of the topic is &heading';

2. Do i=1 to 10;
   TITLE "Name of the topic is Macros";

Here the variable heading could not be resolved in the first part of statements due to presence of single quotes which are not recognized by macro processor.

The following program to determine the content and form of data set PATTERNS in WORK library.

PROC CONTENTS helps to enhance the documentation of data warehouse. The documentation is very important. Users and developers alike appreciate it because it makes the job easier.

```
PROC CONTENTS DATA=PATTERNS;
TITLE 'DATA SET PATTERNS';
RUN;
PROC PRINT DATA=PATTERNS (OBS=10);
RUN;
```

Now, this code works to determine the contents of one data set. To increase modularity, generalize the code and enhance its usage macro can be implemented with the same. It can be rewritten. %Let defines the macro variable whose value needs to be changed in accordance with the requirement.

```
%LET ABC = PATTERNS;
PROC CONTENTS DATA=&abc;
TITLE "DATA SET &abc";
RUN;
PROC PRINT DATA=&abc (OBS=10);

RUN;
```

The value of the macro variable can be changed by issuing a new %LET statement because the most recent definition is used at any time.

Displaying macro variable

The %PUT statement is analogous to DATA step. It writes text and value of macro variable into current SAS System log. The quotation marks are not required for the statement because there is no need to distinguish between a variable name and text string.

```
%LET abc = patterns;

 %PUT **** selected data set is &abc;
```

To see the current value of all the variables created we can make use of the following:

```
%PUT _user_;
```

4.2 MACRO PARAMETERS

Creating modular code with Macros

SAS allows to create package of bug-free codes and use it repeatedly within a single or many SAS programs.

Syntax:

```
%MACRO macro-name;

Text…

%MEND macro-name;
```

%MACRO tells the beginning of a macro, while %MEND indicates the end. Marco-name can be up to 32 characters long. The macro-name with MEND is optional but is a good practice as it makes the code easier to debug.

Invoking a macro

Once the macro has been defined it can by invoked by the following:

```
%macro-name
```

Following is the example of macro usage:

```
%LET abc = PATTERN;

%LET OBS = 10;

%MACRO DEMO;

PROC CONTENTS DATA=&abc;

TITLE "DATA SET &abc";

RUN;

PROC PRINT DATA=&abc (OBS=&abc);

TITLE2 "FIRST &obs OBSERVATIONS";

RUN;

%MEND DEMO;
```

The above macro can be invoked by %DEMO. Combination of macro with %LET increased the efficiency. It gets enhanced further when macros are parameterized.

Adding parameters to macros

The value of parameters can be set when a macro is invoked. To add parameters a list of macro variables is specified in parenthesis.

Syntax:

```
%MACRO macro-name (parameter1=, parameter2= ,...parameter n=);

Text…

%MEND macro-name;
```

The above program can be coded using parameterized macro as:

```
%MACRO DEMO (abc,obs);

PROC CONTENTS DATA=&abc;

TITLE "DATA SET &abc";

RUN;

PROC PRINT DATA=&abc (OBS=&obs);

TITLE2 "FIRST &obs OBSERVATIONS";

RUN;

%MEND DEMO;
```

The macro call for %DEMO could be %DEMO(PATTERNS,10).

It is not required to give all parameters a value. Alternative invocations of the %LOOK macro might include:

```
 %DEMO()

 %DEMO(PATTERNS)

 %DEMO(,10)
```

The difference in these two versions of %DEMO is in the %MACRO statement. The parameters allow us to create &abc and &OBS as local macro variables and we are not required to modify the macro itself. The first value in macro call is assigned to the macro variable that is listed first in the macro statement's parameter list because of the positional feature. Multiple parameters need to use commas to separate their values.

4.3 Iterative %Do

%DO loops in macro are analogous to the DO loop in DATA step. %DO indicates the beginning of the macro section and % END is the end. This section is treated as a unit and is called a %DO Group.

Differences between iterative %DO and DO statement are:

• The %WHILE and %UNTIL specifications cannot be added to the increments

• Increments are integer only

• Only one specification is allowed.

Syntax:

```
 %DO macro-variable = start %TO stop

Text . . .

%END;
```

The iterative %DO defines and increments the macro variable. The following program generates a series of data steps:

```
%macro generate(times);
  %do i=1 %to &times;
    data month&i;
      infile in&i;
      input product cost date;
    run;
  %end;
%mend generate;
%generate(3)
```

On execution of the generate macro, it creates:

```
DATA MONTH1;
   INFILE IN1;
   INPUT PRODUCT COST DATE;
RUN;
DATA MONTH2;
   INFILE IN2;
   INPUT PRODUCT COST DATE;
RUN;
DATA MONTH3;
   INFILE IN3;
   INPUT PRODUCT COST DATE;
RUN;
```

In the above example a stopping value for variable i has been provided. When times reaches the maximum limit the iteration stops.

It is used to write dynamic programs that define the path and logic of execution on its own and is data independent. The dynamic functionality makes the code more reusable.

4.4 Into Inproc Sql

PROC SQL writes macro directly into symbol table. The value is put from data set into macro variable. INTO has the ability to populate macro variable with delimited list of multiple values. It overcomes limitations in hard coding values, including the possibility of resource constraints, typographical errors, and does not account for dynamic data.

Syntax:
INTO: macro-var-specification-1 <...,: macro-var-specification-n>

Here, the INTO clause performs a role similar to SYMPUT routine and follows the scoping rules for %LET statement. INTO clause can assign the value of data to macro variable for the SELECT statement and creates the variable if it does not exist. With the SELECT statement INTO clause can only be used in outer query. It cannot be used with creation of a table or a view.

The following code counts the number of observations that contain a specified string in the table column name. The string is placed in a macro variable (&abc) and the SQL COUNT function is used to count the observations that match the WHERE clause.

```
%let abc = TABLE;
proc sql noprint;
select count(*) into :nobs
from Repository(where=(name=:"&abc"));
quit;
%put number of Repository for &abc is &nobs;
```

The macro variable is preceded by the ampersand when used within SQL step.

4.5 CALL SYMPUT

SAS does not know the value of data until execution phase. Due to the timing issue of macro statement such as %Let, it cannot be used to assign value in DATA step variable to macro variable.

In following code, the requirement is to assign value of name to variable &nameval:

```
data new;
 set old;
 %let nameval =name;
run;
```

Here, %Let executes long before DATA set completes its compilation. Value assigned in nameval is name in lowercase. To overcome this we use SYMPUT.

SYMPUT is DATA step call routine and not macro statement. It directly assign values of data set variables to macro variables.

Syntax :

CALL SYMPUT (macro_varname,value);

Where macro_varname should be enclosed in quotation marks. Value can be the name of a variable whose value is used.

The title in the following program contains macro variable shape that is defined in the previous DATA step. &shape will contain the value of the data set variable shape.

```
data pattern1;

set patterns;

where reg='1';

call symput('shape',shape);

run;

title "Region 1 data for &shape";
```

It will reassign the value of &shape for each observation that meets the WHERE criteria.The macro variable &shape will contain the value of the DATA variable shape from the last observation read from the data set patterns.

Caution:

A macro variable created with CALL SYMPUT cannot be used in the same DATA step because SAS does not assign a value to macro variable until DATA step executes.

5. DEBUGGING MACRO ERRORS

System options have made the debugging of macros easier. There are five system options that affects the messages SAS writes in log. The default setting is in bold.

1) **MERROR**|NOMERROR

When SAS has trouble finding a macro, and MERROR option is on then the following message is printed.

WARNING: Apparent invocation of macro not resolved.

2) **SERROR**|NOSERROR

When SAS trouble to resolve a macro variable and SERROR option is on then the following message is printed.

WARNING: Apparent symbolic reference not resolved.

3) MLOGIC|**NOMLOGIC**

When MLOGIC option is on SAS prints the message in log describing the actions of macro processor.

4) MPRINT|**NOMPRINT**

When MPRINT option is on SAS prints the message in log showing the SAS statement generated by macro.

5) SYMBOLGEN|**NOSYMBOLGEN**

When SYMBOLGEN option is on SAS prints the message in log showing the value of each macro variable after resolution.

6. CONCLUSION

In this paper, the main focus is on SAS macros. It explains how macro can be used to enhance the programs. Macro enables to build complex and repetitive codes. Starting with the %LET statement, to macro definitions macros enable to build complex and repetitive code with ease. Macro variables and macro definitions must be defined before they can be invoked. Macros are essentially strings of data saved in a buffer available to every subsequent data step or procedure in the current SAS session. The execution of macros in code is simply a replacement of the code generated by the macro definition at the point where it is invoked. The use of macros in SAS code can enhances the modularity of code. It generates data driven programs that require less maintenance. SAS provides system options for debugging any kind of errors with macro. The facility helps generate reusable codes which can be used to give stupendous results in specified time limits. Macros can be used to provide and more robust and confident products and solutions for business strategy and data analysis in fields like banking, communication, health care, travel and tourism.

7. REFERENCES

[1] Burlew, Michele M. *SAS Macro Programming Made Easy*. SAS Institute, 2014

[2] Carpenter, Art. Carpenter's complete Guide to the SAS Macro language. SAS Institute, 2004.

[3] Dimaggio, Charles. "Introduction." *SAS for Epidemiologists*. Springer New York, 2013. 1-5.

[4] Slaughter, Susan J., and Lora D. Delwiche. "SAS macro programming for beginners." *Proceedings of the 29th SAS* (2004).

[5] Carpenter, Arthur L. "Five Ways to Create macro Variables: A Short Introduction to the macro language." *Proceedings of the 8th Annual Pharmaceutical Industry SAS Users Group Conference*. 2005.

[6] Delwiche, Lora D., and Susan J. Slaughter. *The Little SAS Book: A Primer: A Primer*. SAS Institute, 2012.

[7] Sadof, Michael G. "1Macros from Beginning to Mend A Simple and Practical Approach to the SAS® Macro Facility." (1999).

[8] SAS Institute. *SAS Macro Language: Reference, Version 8*. SAS institute, 1999.

[9] Slaughter, Susan J., and Lora D. Delwiche. "SAS macro programming for beginners." *Proceedings of the 29th SAS* (2004).

[10] Weise, Daniel, and Roger Crew. "Programmable syntax macros." *ACM SIGPLAN Notices*. Vol. 28. No. 6. ACM, 1993.