PhUSE EU Connect 2018

Paper CT13

Macro Variables and the Many Ways to Create Them

Sarah Berenbrinck, Independent, UK

ABSTRACT

The paper will show some ways of creating and using macro variables to support SAS ® programming. Creation will cover %let, %global, %local, call symput, proc sql and within %macro statements and the difference between keyword parameters and positional parameters. Usage will include the basics of using macros to increase efficiency and creating dynamic titles and headers. It will also show how %put _all_ ; can be used to review the macro variables in your current session both system-generated and user-defined.

INTRODUCTION

Macro variables and user created macros are features in SAS and there are many ways to create and use them.

DATA (FOR LATER)

Here is some code to create a dataset (using datalines) which will be used to demonstrate some things later in this paper.

```
data phuse1 ;
  length year 8 city $9 country $8 ;
  input year city $ country $ ;
datalines ;
2016 Barcelona Spain
2017 Edinburgh Scotland
2018 Frankfurt Germany
;
run ;
```

MACRO VARIABLES

We will start with macro variables.

%LET

I would consider %let the simplest way to create a macro variable.

```
%let testmacvar1=Frankfurt ;
```

This line of code creates a macro variable called testmacro1 with the value Frankfurt.

%PUT

%put is a simple way to see macro variables you have created, you need to use an & before the macro name to alert SAS that it is reading a macro rather than standard text.

```
%put &testmacvar1 ;
```

This will display the macro value in the log.

```
454        +%put &testmacvar1 ;
Frankfurt
```

%put will also displace regular text along with macros.

```
%put The problem value is &testmacvar1 ;
```

```
530        +%put The problem value is &testmacvar1 ;
The problem value is Frankfurt
```

CALL SYMPUT

An example of call symput to create a macro variable, this works within a datastep.
In these examples I will use _null_ in the data statement as I don't want to create a dataset.

```
data _null_ ;
  call symput('testmacvar2', "Frankfurt") ;
run ;

%put &testmacvar2 ;
```

```
611         +%put &testmacvar2 ;
Frankfurt
```

The first parameter is the macro variable name and the second parameter is the value to be assigned to that macro.
If you define either of these as text then you need to enclose them in quotes, single or double will work for each entry.

You could use values from a dataset for the value of the macro variable.

```
data _null_ ;
  set phuse1 ;
  if year=2018 then call symput('testmacvar3', city) ;
run ;

%put &testmacvar3 ;
```

```
822         +%put &testmacvar3 ;
Frankfurt
```

You could also use values from a dataset for the macro names as well as the values, this will create multiple macro variables in 1 data step (assuming you have multiple records in your dataset).

```
data _null_ ;
  set phuse1 ;
  call symput(city, country) ;
run ;

%put &Barcelona &Edinburgh &Frankfurt ;
```

```
2630        +%put &Barcelona &Edinburgh &Frankfurt ;
Spain    Scotland Germany
```

Notice that as the countries are 8 character variables there is space at the end of Spain, this could be fixed with trim.

```
data _null_ ;
  set phuse1 ;
  call symput(city, trim(country)) ;
run ;

%put &Barcelona &Edinburgh &Frankfurt ;
```

```
2712        +%put &Barcelona &Edinburgh &Frankfurt ;
Spain Scotland Germany
```

This could also be fixed with call symputx.

```
data _null_ ;
  set phuse1 ;
  call symputx(city, country) ;
run ;
```

PROC SQL - INTO

An example of proc sql and into to create a macro variable.
In these examples I will use noprint as I don't want the sql output in an output window or file.

```
proc sql noprint ;
  select city into :testmacvar4
  from phuse1
  where year=2018 ;
quit ;

%put &testmacvar4 ;
```

```
987         +%put &testmacvar4 ;
Frankfurt
```

sql can be used to create multiple macro variables in a single step. In sql they have to be named in sequence.

```
proc sql noprint ;
  select city into :testmacvarsql1-:testmacvarsql3
  from phuse1
```

```
quit ;

%put &testmacvarsql1 &testmacvarsql2 &testmacvarsql3 ;
```

```
1069        +%put &testmacvarsql1 &testmacvarsql2 &testmacvarsql3 ;
Barcelona Edinburgh Frankfurt
```

sql can be used to put multiple values in a single macro.

```
proc sql noprint ;
  select city into :testmacvarsql separated by ", "
  from phuse1
quit ;

%put &testmacvarsql ;
```

```
1151        +%put &testmacvarsql ;
Barcelona, Edinburgh, Frankfurt
```

Note that using separated by or creating multiple macro variables in sql, the trailing spaces will be automatically trimmed.

```
proc sql noprint ;
  select country into :testtrimmacvarsql separated by ", "
  from phuse1
quit ;

%put &testtrimmacvarsql ;
```

```
1233        +%put &testtrimmacvarsql ;
Spain, Scotland, Germany
```

However, if only creating 1 macro variable they will remain.

```
proc sql noprint ;
  select country into :testmacvar5
  from phuse1
  where year=2016 ;
quit ;

%put test for spaces before &testmacvar5 after ;
```

```
1728        +%put test for spaces before &testmacvar5 after ;
test for spaces before Spain    after
```

This can be fixed using trimmed.

```
proc sql noprint ;
  select country into :testmacvar6 trimmed
  from phuse1
  where year=2016 ;
quit ;

%put test for spaces before &testmacvar6 after ;
```

```
1820        +%put test for spaces before &testmacvar6 after ;
test for spaces before Spain after
```

MACRO STATEMENTS

This is a way to create a piece of code that can be run multiple times. A macro needs to be started %macro and ended %mend and you can use parameters to change values for each run.

POSITIONAL PARAMETERS

The macro variables can be defined in a positional way. The macro name is not necessary in the mend statement but is very useful when reviewing old code or code written by someone else, so I would recommend always including it. The macro parameters are defined by their position in the statement.

```
%macro printj(city, number);
  title "output &number" ;
  proc print data=phuse1(where=(city="&city")) ;
```

```
  run ;
%mend printj ;

%printj(Barcelona, 1) ;
%printj(Edinburgh, 2) ;
%printj(Frankfurt, 3) ;
```

Note that the quotes used when calling the macro variables are double quotes. If single quotes are used SAS will not search for the macro variable but instead use the text as stated.

output 1

Obs	year	city	country
1	2016	Barcelona	Spain

When using positional parameters, you have to put the values in the correct order or you will get errors (or unexpected results)

```
%print(4, Barcelona) ;
```

```
1982       +%print(4, Barcelona) ;

NOTE: No observations were selected from data set WORK.PHUSE1.
NOTE: There were 0 observations read from the data set WORK.PHUSE1.
      WHERE city='4';
```

KEYWORD PARAMETERS

Keywords rather than positions are used to define macro parameters. Keyword parameters use defined keywords followed by an equal sign.

```
%macro printk(city=, number=);
  title "output &number" ;
  proc print data=phuse1(where=(city="&city")) ;
  run ;
%mend printk ;

%printk(city=Barcelona, number=1) ;
%printk(city=Edinburgh, number=2) ;
%printk(city=Frankfurt, number=3) ;
```

This gives exactly the same output.

output 1

Obs	year	city	country
1	2016	Barcelona	Spain

And while this requires more typing I think it is worth it as you do not need to get the positions correct and the macro call has clearer meaning (with descriptive parameter names).

```
%printk(number=4, city=Barcelona) ;
```

output 4

Obs	year	city	country
1	2016	Barcelona	Spain

With either positional or keyword you do not need to populate all of the parameters when calling the macro, however with positional you have to populate them in order, so it is easier to leave missing values when using keyword parameters.

```
%printk(city=Barcelona) ;
```

output

Obs	year	city	country
1	2016	Barcelona	Spain

With keyword parameters you can define the values of the parameters.

```
%macro printm(city=Frankfurt, number=x) ;
  title "output &number" ;
  proc print data=phuse1(where=(city="&city")) ;
  run ;
%mend printm ;
```

Then when you call the macro you can use some or all of the prepopulated values or redefine them.

```
%printm() ;
%printm(city=Barcelona) ;
%printm(city=Edinburgh, number=2) ;
```

output x

Obs	year	city	country
3	2018	Frankfurt	Germany

output x

Obs	year	city	country
1	2016	Barcelona	Spain

output 2

Obs	year	city	country
2	2017	Edinburgh	Scotland

MIXED PARAMETERS

You can mix keyword and positional parameters when defining macros.
If both positional and keyword parameters appear in a macro definition, positional parameters must come first.

LOCAL/GLOBAL

When creating your own macros, you will likely wish to create macro variables, and SAS differentiates between macro variables that exist inside the macro (local) and those that exist through all of your SAS session (global).
In many cases this is fine as the macro variables that you create inside of a macro are only needed within that macro, but sometimes you may want to create macro variables inside a macro to use later in the program, so in this case you need to tell SAS that you want those macro variables to be global.

CREATION

You can declare a macro variable to be global by using %global <macro var name> before creating the macro variable.

```
%global testmacvar5 ;
%let testmacvar5=Amsterdam ;
```

Similarly, you can ensure a macro variable is local by using %local. This may be useful if you wish to use a macro variable name that is the same as a macro variable which already exists in your session, outside of the macro. By using %local this new local macro variable will not affect the global macro variable. Generally, it would be better to use a unique name to avoid confusion.
Macro variables created outside of macros will automatically be global and those created inside of macros will be local to that macro. Note that some systems will submit programs to a server wrapped in their own macros which complicates the issue of global macro variables.

USAGE

I will note some places where I use macros.

MACRO VARIABLES

More dynamic code by creating macro variables to populate headers and footnotes.
Counting number of subjects into a macro variable and using this to populate headers.
Putting MedDRA version in a macro variable and using that in the footnote.
Putting a list of PT terms into a macro to add to a footnote.

MACRO STATEMENTS

Increased efficiency and consistency by using macros to run bits of code multiple times.
Creating tables of the same structure and content on different subgroups.
Running counts or analysis on a number of different PT terms.
While debugging code you can wrap parts you don't want to run in a macro and not call that macro, this should be removed from final code as it doesn't meet programming guidelines on removing unused code and is very confusing for reviewers.

%PUT

This can be used to check values assigned to macro variables.
You can also use %put to add notes to the log.
If you are running a long or complicated program you can flag certain points of the program to check timing or progress.
When using where or if statements you can write to the log details of records which don't meet expected criteria.
If you are having issues with how a program runs but struggle to identify where in the log, then you can add distinctive text to the log to help your search.

%PUT _ALL_ ;

This will write all macro variables and their values to the log, this can be useful to see what you have created and also for reviewing SAS automatic macro variables. %put _local_ ; and %put _user_ ; are also possible.

SOME OF MY FAVORITES/COMMENTS:

Some of the variables I created making this paper, note that these are global as I use a system that runs code wrapped in a macro and I had to create global macros for them to appear with %put _all_ ;

```
GLOBAL TESTMACROSQL1 Barcelona
GLOBAL TESTMACROSQL2 Edinburgh
GLOBAL TESTMACROSQL3 Frankfurt
```

System date and time can be useful in headers, titles and footnotes.

```
AUTOMATIC SYSDATE 28SEP18
AUTOMATIC SYSDATE9 28SEP2018
AUTOMATIC SYSDAY Friday
```

My system is not local and so this was + 1 hour for me, local server time. This is the time the session started which may not be the current time.

```
AUTOMATIC SYSTIME 10:19
```

This is one way of checking the SAS version you are using which might be useful to explain how code is working and which features you have.

```
AUTOMATIC SYSVER 9.4
```

SAS has a macro of the number of observations in the last PROC SQL you ran, this can be useful if you want to check if datasets are empty and create a different report where that is the case.

```
SCSM SQLOBS 3
```

As my system runs code wrapped in a macro and connected to my log in, it knows who I am.

```
SCSM _EXEC_USERFIRSTNAME Sarah
```

CONCLUSION

Macros and macro variables can be used in many ways to aid programming and efficiency. Care should be taken when using them as we work in an industry where traceability and reuse of code are important, so we need to take care that macros are well commented and easy to follow.

REFERENCES

http://documentation.sas.com/?docsetId=mcrolref&docsetTarget=titlepage.htm&docsetVersion=9.4 SAS 9.4 Macro Language: Reference,

RECOMMENDED READING

I have not included any information on && and . when calling macros, these are very useful tools which are worth researching.

CONTACT INFORMATION

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

Sarah Berenbrinck
Independent
Email: sberenbrinck@gmail.com