Paper 1476-2017

SAS Macros – Advanced Techniques

Author Dari Mazloom, USAA

ABSTRACT

The SAS macro language provides a powerful tool to write a program once and re-use it many times in multiple places. A repeatedly-executed section of a program can be wrapped into a macro which can then be shared among many users. A practical example of a macro can be a utility that takes in a set of input parameters, performs some calculations, and sends back a result; such as an interest calculator.

In general, a macro modularizes a program into smaller and more manageable sections, and encapsulates repetitive tasks into re-usable code. Modularization can help the code to be tested independently. This paper provides an introduction to writing macros. It introduces the user tothe basic macro constructs and statements.

This paper covers the following advanced macro subjects

- Using multiple &'s to retrieve/resolve the value of a macro variable
- Creating a macro variable from the value of another macro variable
- Handling special characters
- the EXECUTE statement to pass a DATA step variable to a macro
- Using the EXECUTE statement to invoke a macro
- Using %RETURN to return a variable from a macro

INTRODUCTION

This paper offers a set of advanced techniques to help the SAS practitioners to write a macro can help a program to be more organized, better managed, and independently tested. Due to their re-usability, macros are also great tools for codes that are frequently re-used.

TOPIC 1: USING MULTIPLE &'S TO RETRIEVE/RESOLVE THE VALUE OF A MACRO VARIABLE

First, we will discuss the role of an ampersand (&) as it is related to macro variables.

An ampersand is used to retrieve the value of a macro variable. This is the same as de-referencing a pointer in C or C++.

Example:

```
%LET BIRTH_DAY = 01JAN2016;
/*The & is used to retrieve the value of BIRTH_DAY*/
%PUT BIRTH_DAY=&BIRTH_DAY;
```

```
The log will display: BIRTH_DAY=01JAN2016
```

Now, let's join or concatenate two macro variables:

```
%LET COLOR1=RED;
%LET INDEX=1;
```

The goal is to concatenate the word COLOR with the value of index which is 1;
resulting in the value of variable COLOR1 which is RED and place it in variable COLOR2;

```
%LET COLOR2=&COLOR&INDEX;
```

The above LET statement is only scanned once and the result is &COLOR1.
Now, we will display the value of COLOR2 in the log:

```
%PUT COLOR2=&COLOR1;
```

The log will display the following warning:

```
WARNING: Apparent symbolic reference COLOR not resolved.
&COLOR1
```

The solution is to force the macro compiler to go thru a second scan which
can be accomplished by using double ampersands which will resolve to one ampersand.

Let discuss the two scans:
When &&COLOR&INDEX goes thru the first scan, it results to &COLOR1.
The second scan resolves &COLOR1 to its value, RED, which is then assigned
to variable COLOR2.

Here is the corrected code:

```
%LET COLOR1=RED;
%LET INDEX=1;
%LET COLOR2=&&COLOR&INDEX;
%PUT COLOR2=&COLOR1;
```

```
The log will display: COLOR2=RED
```

TOPIC 2: CREATING A MACRO VARIABLE FROM THE VALUE OF ANOTHER MACRO VARIABLE MACRO

At times, there may be a need to create a new variable or column in a dataset from the value of another variable. The following macro creates three new variables (RED, WHITE, and BLUE) from three existing variables (COLOR1, COLOR2, and COLOR3):

```
%MACRO M2();
   %LET COLOR1=RED;
   %LET COLOR2=WHITE;
   %LET COLOR3=BLUE;
   %DO INDEX=1 %TO 3;
      %IF (&INDEX=1) %THEN %LET &&COLOR&INDEX = BRIGHT RED;
      %IF (&INDEX=2) %THEN %LET &&COLOR&INDEX = OFF WHITE;
      %IF (&INDEX=3) %THEN %LET &&COLOR&INDEX = DARK BLUE;
   %END;
   %PUT &RED;
   %PUT &WHITE;
   %PUT &BLUE;
%MEND M2;
%M2;
```

```
The log will display:
BRIGHT RED
OFF WHITE
DARK BLUE
```

TOPIC 3: HANDLING SPECIAL CHARACTERS

When writing macros, special considerations need to be paid to special characters such as unmatched apostrophes or single quotes.

The following set of statements will not compile since the macro variable NAME has an unmatched apostrophe:

```
%LET NAME=O'CONNOR;
%PUT &NAME;
```

Solution: we will use the macro function %STR to mask the single apostrophe so that the statement would compile:

```
%LET NAME=O%STR(%')CONNOR;
%PUT &NAME;
```

```
The log will display: O'CONNOR
```

TOPIC 4: USING EXECUTE STATEMENT TO PASS DATA STEP'S VARIABLE TO A MACRO

All macro calls in a DATA step execute prior to the completion of the data step.

Therefore, if a data step variable is passed to a macro, its value does not get resolved

when the macro is invoked. The following DATA step is invoking a macro and passes a non-macro variable.

The call will result in displaying the name of the variable and not its value since

the macro call does not resolve the variable's value and treats the variable like a

string constant:

```
%MACRO M4(NAME=);
   %PUT &NAME;
%MEND M4;
DATA _NULL_;
   ANY_NAME = "JOHN";
   /* ANY_NAME is passed as a string constant; and its value is NOT passed*/
   %M4(NAME=ANY_NAME);
RUN;
%M4(NAME=JOHN);
```

```
The log will display the name of the variable and not its value: ANY_NAME
```

The solution is to wait for the DATA step to complete first; and this is accomplished by calling

the macro with the EXECUTE statement and passing the macro's name and its parameters as

arguments of the EXECUTE statement.

Here, the entire CALL EXECUTE statement is treated like a SAS code and placed on

the stack. The code will execute once the DATA step completes:

```
%MACRO M4(NAME=);
   %PUT &NAME;
%MEND M4;
```

```
DATA _NULL_;
   ANY_NAME="JOHN";
   CALL EXECUTE('%M4A(NAME=' ||ANY_NAME||')');
RUN;
```

```
The log will display the value of the passed parameter: JOHN
```

The CATS function can also be used to concatenate the macro name with its parameters:

```
CALL EXECUTE(CATS('%M4(NAME=', ANY_NAME,')'));
```

TOPIC 5: USING EXECUTE STATEMENT TO INVOKE A MACRO

If a DATA step invokes a macro, the macro is executed prior to completion of the Data step. This can present an issue if the macro contains DATA steps and if those data steps are dependent on parameters from the calling data step.

The following macro contains two DATA steps which are dependent on an input parameter:

```
%MACRO POINTS(IN_POINTS=);
   DATA _NULL_;
      LENGTH FIRST_POINTS 8;
      FIRST_POINTS = &IN_POINTS;
      CALL SYMPUT("FIRST_POINTS", FIRST_POINTS);
   RUN;

   DATA _NULL_;
      LENGTH FINAL_POINTS 8;
      FINAL_POINTS = &FIRST_POINTS * 1.5;
      CALL SYMPUT("FINAL_POINTS", FINAL_POINTS);
   RUN;
   %PUT FIRST_POINTS=&FIRST_POINTS;
   %PUT FINAL_POINTS=&FINAL_POINTS;
%MEND POINTS;
```

Now, we will invoke the macro from a DATA step.

```
DATA _NULL_;
```

```
   MY_POINTS = 95;
   CALL EXECUTE(CATS('%POINTS (IN_POINTS=', MY_POINTS, ')'));
RUN;
```

The log will display warning:

```
WARNING: Apparent symbolic reference FIRST_POINTS not resolved.
FIRST_POINTS=&FIRST_POINTS
```

The log will also display warning:

```
WARNING: Apparent symbolic reference FINAL_POINTS not resolved.
FINAL_POINTS=&FINAL_POINTS
```

Solution: Since the execute macro statement suspends the data steps inside the macro, the execute macro statement needs to wait until the data step, which calls the macro, completes. This can be accomplished by using function **%NRSTR** to mask the instructions during compile time and place the macro call on the input queue.

```
DATA _NULL_;
MY_POINTS = 95;
CALL EXECUTE(CATS('%NRSTR(%POINTS(IN_POINTS=', MY_POINTS , '))'));
RUN;
```

The log will display:

```
NOTE: CALL EXECUTE generated line:
%M4(IN_POINTS=95)
FIRST_POINTS=95
FINAL_POINTS=142.5
```

TOPIC 6: USING %RETURN TO RETURN A VARIABLE FROM A MACRO

%RETURN statement can be used to return a variable from a macro.

The following macro takes in two input parameters, performs a calculation, and returns the result of the calculation:

```
%MACRO CalcCost(AMOUNT=, INTEREST=);
      %LET TOT_COST=%EVAL(&AMOUNT + ((&AMOUNT * &INTEREST) / 100));
      &TOT_COST;
      %RETURN;
```

```
%MEND CalcCost;
```

Here is how the macro is called:

```
%LET FINAL_COST = %CalcCost(AMOUNT=1000, INTEREST=10);
%PUT FINAL_COST = &FINAL_COST;
```

```
The log will display:
FINAL_COST = 1100
```

CONCLUSION

This paper provided a set of advanced techniques that can be used to write more complex macros.

REFERENCES

SAS(R) 9.3 Language Reference: Concepts, Second Edition

ABOUT THE AUTHOR

Dari Mazloom is a lead business intelligence advisor with over thirty years of practical experience in a number of programming languages such as SAS, C#, C++, JAVA, VB, VBA, Smalltalk, COBOL, Easytrieve, JCL..