MWSUG 2016 - Paper BB04

Simplifying Your %DO Loop with CALL EXECUTE

Arthur Li, City of Hope National Medical Center, Duarte, CA

ABSTRACT

One often uses an iterative %DO loop to execute a section of a macro repetitively. An alternative method is to utilize the implicit loop in the DATA step with the EXECUTE routine to generate a series of macro calls. One of the advantages in the latter approach is eliminating the needs of using indirect referencing. To better understand the use of the CALL EXECUTE, it is essential for programmers to understand the mechanism and the timing of macro processing to avoid programming errors. These technical issues will be discussed in detail in this paper.

INTRODUCTION

A common task for writing a macro program is to execute a macro program with different parameters. One often utilizes an iterative %DO statement to loop along the value of the parameters. Using the iterative %DO statement often involves the use of indirect referencing within the loop, which can be difficult to comprehend by a novice programmer. The DATA step EXECUTE routine provides an alternative approach to executing macros without using the %DO statement. The syntax of CALL EXECUTE is as follows:

CALL EXECUTE(*argument*);

The *argument* in CALL EXECUTE can be a text expression that is enclosed in single or double quotation marks, the name of a character variable, or a character expression that is resolved by the DATA step to a macro text expression or a SAS statement.

If the *argument* in CALL EXECUTE produces macro language elements that are enclosed in single quotes, those elements execute immediately during the DATA step execution phase. On the other hand, if the generated macro language elements that are enclosed in double quotes, those elements will execute immediately during the DATA step compilation phase. Single quote notation is commonly used in most applications.

If *argument* in CALL EXECUTE produces SAS language statements, or if the macro language elements from ARGUMENT generate SAS language statements, the SAS language statements will be placed into the input stack as an additional program code and will execute after the end of the current DATA step's execution.

AN EXAMPLE BY USING THE %DO STATEMENT

Suppose that we have a series of text files, named *Atascadero_F.txt*, *Atascadero_M.txt*, *LongBeach_F.txt*, *LongBeach_M.txt*, *Riverside_F.*txt, *Riverside_M.txt*, etc. The content format for each file is identical. By looking at the file name, you will be able to identify the subjects' residential city and gender. For example, *Riverside_F.txt* contains information for females living in Riverside.

Program 1 is a macro program for reading one single text file by providing city name and gender information.

Program 1:

```
%macro cr1File(city, gender);
data &city&gender;
    infile "C:\Users\Arthur\Documents\CALL Execute\data\&city._&gender..txt";
    input id $ 1-3 race 4 age 6-8;
    *create variable townname and gender;
    townname ="&City";
    gender = "&gender";
run;
%mend cr1File;

%cr1File(Atascadero, M)
```

To read multiple files for both genders and then concatenating them together, you can use iterative loops within a macro program, like in Program 2.

Program 2:
```
%macro crAll(citylist);
    %let num = %sysfunc(countw(&citylist));
    %local i;
    %do i =1 %to &num;
        %let city&i = %scan(&citylist, &i);
            %cr1File(&&city&i,M);
            %cr1File(&&city&i,F);
    %end;

    data allCityGender;
        set
            %do i =1 %to &num;
                &&city&i..M
                &&city&i..F
            %end;
            ;
    run;
%mend crAll;
%crAll(Atascadero LongBeach Riverside)
```

In program 2, the first iterative %DO loop is used to call the CR1FILE macro to create an individual file for male and female separately. The second %DO loop in the DATA step, that is nested within the SET statement, is used to stack all the files together in the end. In order to properly use the %DO loop, the indirect referencing is needed to generate references to a series of macro variables.

ALTERNATIVE SOLUTION BY USING CALL EXECUTE

We can accomplish the same task by using CALL EXECUTE like in Program 3. Instead of passing a list of cities as the parameter value, the names of the city are stored in a SAS dataset, which will be used as the value for calling the macro CRALL_NEW.

Program 3:
```
%macro crAll_new(cityDat);
    data _null_;
        set &cityDat;
        call execute(cats('%cr1File(', city, ',', "M", ')'));
        call execute(cats('%cr1File(', city, ',', "F", ')'));
    run;

    data _null_;
        set &cityDat end=last;
        if _N_ = 1 then call execute('data all; set');
        call execute(cats(city, 'M'));
        call execute(cats(city, 'F'));
        if last then call execute('; run;');
    run;
%mend crAll_new;

data city;
    length city $20;
    input city;
datalines;
Atascadero
LongBeach
Riverside
;
%crAll_new(city)
```

The first DATA step in the macro program is used to call the CR1FILE macro multiple times, depending upon the number of cities that are stored in the given dataset. The implicit loop from the DATA step, along with the CATS function, generates macro calls that are used in the EXECUTE routine. For example, when the DATA step reads the first observation from &CITYDAT, the value of the DATA step variable CITY contains “Atascadero.” The CATS function then generates the following macro calls:

```
%cr1File(Atascadero, M)
%cr1File(Atascadero, F)
```

The second DATA step utilizes CALL EXECUTE to generate the final dataset that concatenates all the generated datasets together. That is to say, it generates the code below:

```
data all;
    set AtascaderoM
        AtascaderoF
        LongBeachM
        LongBeachF
        RiversideM
        RiversideF;
run;
```

LIMITATION

If you invoke a macro by enclosing the macro call as the *argument* of CALL EXECUTE, the macro call will execute immediately. If the macro call generates any macro language element, such as the %IF-%THEN statement or macro references, these macro language elements execute immediately. However, any of the SAS language statements that are generated by the macro call will be pushed to the input stack and executed after the end of the current DATA step, which contains CALL EXECUTE. This will create problems if you invoke a macro that contains references for macro variables that are created by CALL SYMPUT(X). CALL SYMPUT(X) is not considered a part of the macro language; instead, it is just DATA step CALL routines. That is to say the macro references to the macro variables created by CALL SYMPUT(X) will execute before they are even created.

In Program 4, macro FOO creates a macro variable VALUE by using CALL SYMPUTX. This program illustrates the differences in invoking FOO between using CALL EXECUTE and without using CALL EXECUTE. The use of CALL EXECUTE in both DATA steps avoids use of indirect referencing.

<u>Program 4</u>:
```
option mprint mlogic symbolgen;
%macro foo;
    %local value;
    data bar;
        a = 5;
       call symputx('value', a);
    run;
    %put value inside macro foo: &value;
%mend;

%foo
%put value outside macro foo &value;

data _null_;
    call execute('%foo');
run;
%put value outside macro foo &value;
```

SAS Log from Program 10:

```
792  %foo
MLOGIC(FOO):  Beginning execution.
MLOGIC(FOO):  %LOCAL  VALUE
MPRINT(FOO):   data bar;
MPRINT(FOO):   a = 5;
MPRINT(FOO):   call symputx('value', a);
MPRINT(FOO):   run;

NOTE: The data set WORK.BAR has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MLOGIC(FOO):  %PUT value inside macro foo: &value
value inside macro foo: 5
MLOGIC(FOO):  Ending execution.
WARNING: Apparent symbolic reference VALUE not resolved.
793  %put value outside macro foo &value;
value outside macro foo &value
794
795  data _null_;
796      call execute('%foo');
797  run;

MLOGIC(FOO):  Beginning execution.
MLOGIC(FOO):  %LOCAL  VALUE
MPRINT(FOO):   data bar;
MPRINT(FOO):   a = 5;
MPRINT(FOO):   call symputx('value', a);
MPRINT(FOO):   run;
MLOGIC(FOO):  %PUT value inside macro foo: &value
value inside macro foo:
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MLOGIC(FOO):  Ending execution.

NOTE: CALL EXECUTE generated line.
1   + data bar;         a = 5;         call symputx('value', a);     run;

NOTE: The data set WORK.BAR has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


798  %put value outside macro foo &value;
value outside macro foo 5
```

When invoking FOO the first time without using CALL EXECUTE, the macro variable VALUE is created and stored in the local symbol table. The VALUE is then deleted from the local symbol table at the end of the macro execution. When using CALL EXECUTE to invoke FOO, the %LOCAL statement executes immediately, which assigns the macro variable VALUE to a NULL value. The DATA step within the macro FOO was pushed to the input stack. The %PUT statement executes next; notice that at this point, VALUE contains a null value. The DATA step that creates BAR executes after the execution of FOO, which creates the macro variable VALUE. Since the macro execution has already ended, the VALUE is then stored in the global symbol table, which is not what you intended.

CONCLUSION

The use of the EXECUTE routine provides an efficient solution to eliminate the need of using the iterative %DO loop along with indirect referencing in the macro program. However, knowing when best to use CALL EXECUTE and understanding the mechanism of macro processing are essential to writing an accurate macro program.

REFERENCES

Li, Arthur. Is Your Failed Macro Due To Misjudged “Timing”? SAS Global Forum 2012 Proceedings.
SAS Institute. (2010). *SAS® 9.4 Functions and CALL Routines Reference.* Cary, NC: SAS Institute.
Usov, Artur. Call Execute: Let Your Program Run Your Macro. PhUSE 2014 Proceedings.

CONTACT INFORMATION

Arthur X. Li
City of Hope National Medical Center
Division of Information Science
1500 East Duarte Road
Duarte, CA 91010 - 3000
Work Phone: (626) 256-4673 ext. 65121
Fax: (626) 471-7106
E-mail: arthurli@coh.org