Paper 154-30

No %'s or &'s: Macros Inside the SAS® DATA Step

William C. Murphy
Howard M. Proskin & Associates, Inc., Rochester, NY

ABSTRACT

Macro programming is one of the most powerful and flexible techniques in the SAS system. Macro programs can be used to automate repetitive tasks and for conditional execution of program statements. For some, however, the syntax of macro code may sometimes be confusing and hard to follow. But most macro tasks can be accomplished with the use of a CALL EXECUTE routine within a DATA step. The argument to the Call execute routine is a text string or variable containing a text string. The Call execute resolves this argument and executes it after the end of the calling DATA step. The text string argument can be coded to execute another DATA step or to run a PROC or both. In fact, you can use the CALL EXECUTE routine within a DATA step to write a complete SAS program that can perform a variety of tasks, under the control of the calling DATA step.

INTRODUCTION

Macro programs are wonderful! Macro programs are great! Macros allow you to simplify repetitive tasks. They permit you to conditionally execute DATA steps or PROCs. And to a large extent, macros are totally unnecessary. Nearly anything that can be done with a macro program can be performed by a DATA step with the use of the CALL EXECUTE statement.

THE CALL EXECUTE STATEMENT

The CALL EXECUTE statement is a DATA step routine that has one argument. That argument is a text string or a DATA step variable that resolves to a text string. The text string can consist of any legitimate SAS program code. This code will be resolved after the DATA step containing the CALL EXECUTE statement finishes. (If you chose to include a macro reference in the CALL EXECUTE, the resolution will be a little different - See the *SAS OnLIneDoc*.) For example, I can easily imbed a PROC in a DATA step by writing

```
data two;
       set one;
       call execute ('proc sort data=two; by id; run;');
                ...
       (More DATA step statements)
                ...
       run;
```

This DATA step will not only create data set *two*, but will sort the data set by the variable *id*. The sorting will occur after the DATA step is finished executing. The SAS log will show the source code generated by the CALL EXECUTE function prefixed with '+'. However, one should be warned that this CALL EXECUTE will be invoked for every observation read into the DATA step. You would end up with data set *two* being repeatedly sorted. But, just as with any DATA step statement, the CALL EXECUTE can be subjected to conditional execution. We could therefore rewrite the statement in the above DATA step to

```
if _n_=1 then call execute ('proc sort data=two; by id; run;');
```

where the PROC SORT will now only be executed once. Although we chose to invoke the CALL EXECUTE with the first observation (_n_=1), we could have chosen virtually any observation since the CALL EXECUTE code is only run after the DATA step is finished executing. More to the point this example shows that the CALL EXECUTE can be controlled by the DATA step.

By expanding the text argument and using the abilities of the DATA step, we can use CALL EXECUTE statements to write whole programs. In other word, a DATA step with a CALL EXECUTE routine can perform the same function as a macro program. In fact, the code using the CALL EXECUTE method is often shorter and easier to understand than

macro code. To illustrate this we will contrast the code for two macro routines (one for documenting the database and another for writing to Excel) with the functionally equivalent DATA step containing a CALL EXECUTE.

DOCUMENTING THE DATABASE

You can readily use the SAS system to provide documentation of your database. Such documentation would consist of lists of the variables in each data set, interleaved with sample print outs from the associated data set. To accomplish this with a macro, you would first need to read the data set names into macro variables and then use a %DO loop to cycle through each name. The main part of the macro would look like this:

```
%do i=1 %to &nDataSet;

    title3 "Structure of Data Set &&DataSet&i";
    proc contents data=&library..&&DataSet&i;
       run;

    title3 "Partial Listing of Data Set  &&DataSet&i";
    proc print data=&library..&&DataSet&i(obs=5);
       run;

    %end;
```

where &&DataSet&i contains the name of the data set and &nDataSet is the total number of data sets in the library &library (for a complete listing of the macro see Murphy 2003a).

But this same documentation can readily be done in one DATA step without the bookkeeping chores of the macro program. To create the DATA step, first we get a list of the desired data sets (contained in the library *Project*) from the SAS dictionary:

```
data _null_;
      set sashelp.vtable(where=(libname='Project'));
```

Then we define a variable with the full name of the data set:

```
     file=compress(libname||'.'||memname);
```

Finally we invoke a CALL EXECUTE:

```
call execute("   title3 'Structure of Data Set "||trim(memname)||"';
                 proc contents data="||file||";
                      run;
                 title3 'Listing of Data Set "||trim(memname)||"';
                 proc print data="||file||"(obs=5);
                      run;                                  ");
```

The argument of the CALL EXECUTE looks very similar to the code within our macro loop. In fact, we deliberately parsed the argument to look like SAS code. However, the argument to the CALL EXECUTE is one long text string. Instead of using &&Data&i, the content of the variable *memname* is concatenated with the rest of the TITLE statement and instead of using &&library..&&Data&i in the PROC calls, the content of the variable *file* is concatenated to the PROC text. In short, we have replaced the macro variables by data set variables. The looping is accomplished by the normal iteration of the DATA step and the variable &nDataSet is effectively replaced by the number of observations inputted by the SET statement.

Looking at this sample, you might say that the macro code is about as large as the CALL EXECUTE code. However, the DATA step containing the CALL EXECUTE is complete and will perform the documentation process as written. The macro code however, needs to be set up by storing the data set names in the variables &&DataSet&i, the number of data sets must be determined and stored in the variable &nDataSet, and finally the various booking chores associated with a macro (*i.e*. declaring variables %LOCAL, deleting temporary data sets created by the macro, *etc.*) must be preformed.

WRITING TO EXCEL

In the above example, we illustrated how you can imbed PROCs and TITLE statements into a DATA step. However, you can also insert a DATA step within a DATA step. Using this process, we can write any data to an Excel spreadsheet. In a previous presentation (Murphy 2003b), we displayed a macro for performing this operation. First, we set up a data set containing the output information:

```
data one;
       infile datalines missover;
       input cell $ value;
       datalines;
r6c6          18
r6c8          59
...
       ;
       run;
```

where the second variable, *value*, is to be written to the spreadsheet position given by the variable *cell* (*e.g.* r6c6 is row 6 column 6). After the initialization, the macro is used to execute a %DO loop. In the first part of this loop, SAS Component Language (SCL) is used to OPEN and access data set *one*. The number of cells to be written is stored in &nCell and each value of the variables *value* and *cell* is stored into the corresponding macro variables &Value and &Cell. After this, we continue with %DO loop as follows:

```
%do i=1 %to &nCell;
       ...
   (Macro and SCL statements)
       ...

   filename spread dde "excel|Template!&Cell" notab;

   data _null_;
       file spread;
       put "&Value";
       stop;
       run;
   %end;
```

where the filename statement establishes a connection to the Excel spreadsheet named *Template* at the row and column position specified by &Cell. The DATA step then uses a PUT statement to write &Value to the spreadsheet.

But again, the same process can be done in a single DATA step:

```
data _null_;
       set one;
       sheet=compress('excel|Template!'||cell);
       call execute("filename spread dde '"|| sheet||"' notab;
                     data _null_;
                            set one(where=(cell='"||cell||"'));
                            file spread;
                            put value;
                            run;"        );
       run;
```

where once more the code is similar to the macro coding. The CALL EXECUTE again has a single text string as an argument. The text string consists of a FILENAME statement concatenated with information necessary to establish the connection to the spreadsheet, followed by a DATA step string to select and write the data. The variable *sheet* is created separately to prevent any extraneous spaces from being in the text that contains the spreadsheet location information. In the CALL EXECUTE DATA step, we no longer need rely on an externally created macro variable for output but instead on the content of the DATA step. The WHERE option on the DATA step assures that only one cell is written at a time.

Again, looking at this sample, you might say that the macro code is about as large as the CALL EXECUTE code. However, as in our previous example, the DATA step containing the CALL EXECUTE is complete and will perform the spreadsheet writing process as written. The macro code however, needs to be set up by counting the number of cell to be written, &nCell, and the %DO loop must be supplemented with SCL code to obtain &Cell and &Value. Of course, the various bookkeeping chores associated with a macro must also be preformed.

CONCLUSION

CALL EXECUTE is a very powerful routine. It can be used to execute PROCs, DATA steps and other SAS statements under the control of a data set. Consequently, DATA steps using CALL EXECUTE statements can readily replace macro programs for many tasks in the SAS system. The code using CALL EXECUTE is often shorter and more easily understood than the corresponding macro code. In short, the CALL EXECUTE routine provides a powerful alterative to macro programming for the automation of a variety of tasks.

REFERENCES

SAS Institute Inc. (2002). "Macro Language Dictionary: EXECUTE Routine". *SAS OnLineDoc® 9*. SAS Institute Inc., Cary, NC. (url: http://v9doc.sas.com/sasdoc).

Murphy, W.C. (2003a), "Using a SAS® Macro to Document the Database", *Proceedings of the Twenty-Eight Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, paper 91-28.

Murphy, W.C. (2003b), "Filling Report Templates with the SAS® System and DDE", *Proceedings of the Twenty-Eight Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, paper 217-28.

CONTACT INFORMATION

Your comments and questions are values and encouraged. Contact the author at

William C. Murphy
Howard M. Proskin & Associates, Inc.
300 Red Creek Dr., Suite 330
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com