CALL EXECUTE for everyone!

Examples for programmer, statistician, and data manager

Daniel Boisvert, Genzyme, Cambridge MA
Shafi Chowdhury, Shafi Consultancy, London England

Abstract:

Everything changes. The data, the program names everything! This is a hard fact to learn when first working with clinical trials. But, it is true. Creating programs that only work when everything is static is not a good idea because, guess what, something is going to change. CALL EXECUTE helps to solve this problem by automatically adapting code to the data. This paper will present three examples of its uses: one for programmers, one for statisticians and one for data managers.

1. Programmers - Run all programs in a directory
2. Statisticians - Create consistent symbols for SAS/GRAPH®
3. Data Manager - Search for data through a library

While these three examples are superficially unrelated, their continuity comes with the fact that they are data driven. When the data changes behind the program no change is needed to the program code. This paper is aimed at a more experienced programmer who understands complex DATA step manipulation.

Introduction:

When writing effective CALL EXECUTE code one should work backwards:

- Write, in open code, the basic statement which should be run.
- Pick out the piece(s) of the code which are to be replaced by data set variables.
- Create a data set which contains the values you want to put into your basic statement
- Write a DATA step with CALL EXECUTE

Remember:

- The DATA step operates as a loop; processing one observation at a time.
- CALL EXECUTE writes the code as the DATA step is processed and executes it at the end of the step.
- The generated code can use the values from SAS data set variables.
- Keep it simple. Super complicated code causes headaches.

Examples:

1. Programmer - Run all programs in a directory
A time will come when the programmer needs to run all programs in a certain directory. This can be accomplished in many ways. The most common is to create one program with a series of %INCLUDE statements. This invariably leads to forgotten programs or incorrect %INCLUDE statements when program names change. A better solution is to write the %INCLUDE statements, automatically, based on the names of the programs in the specified folder.

Start by writing the code we want to be looped in our CALL EXECUTE, and finding the piece of this code we wish to replace with the program names.

```
%INCLUDE "C:sascodes\replaceME.sas;"
```

Now that we have the code, we see that our data set needs to include the names of all of our SAS programs. This way we can eventually change the above code into:

```
CALL EXECUTE ('%INCLUDE "C:sascodes\"||[program name]||";');
```

Using a PIPE statement read the contents of a folder into SAS as a data set.

```
FILENAME file1 PIPE 'dir "C:sascodes\"';

DATA sasfiles;
LENGTH pgmname $200;
INFILE file1 TRUNCOVER;
INPUT sasfile $ 1-200;
pgmname=SCAN(sasfile,-1,');
IF SCAN(sasfile,-1,'.')='sas';
RUN;
```

Resulting in the data set:

```
SAS   [VIEWTABLE: WORK.SASFILIES]
```

Now take the variable containing the program names (pgmname) and plug it into our already constructed CALL EXECUTE statement. As the data step runs, each file will be included and run independently.

```
DATA _NULL_;
SET sasfiles;
CALL EXECUTE ('%INCLUDE "C:sascodes\"||pgmname||";');
RUN;
```
If a program name changes or a new program is added to the directory, no change to the program is necessary. When rerun the data set *sasfiles* will automatically contain the updated information.

2. Statistician - SAS/GRAPH Symbol Statement

Assume the task at hand is to produce a series of graphs of mean over time by treatment (mean*actevent=tpatt). This sounds very straight forward and it is likely that we will write some code like this.

```sas
SYMBOL1 I=J V=CIRCLE L=7 C=BLACK;
SYMBOL2 I=J V=STAR L=3 C=BLACK;
SYMBOL3 I=J V=DIAMOND L=5 C=BLACK;
SYMBOL4 I=J V=DOT L=1 C=BLACK;
TITLE H=1 F=SWISSB 'All treatment groups - normal method';
PROC G PLOT DATA=indata;
   PLOT mean*actevent=tpatt;
RUN;
QUIT;
TITLE;
```

This results in a perfectly fine graph.

The problem occurs when one of the treatments is left out.
In this example Treatment D has taken on the symbol of diamond whereas in the previous example Treatment D was a solid dot. This occurs because the symbol statement is assigned in sequential order and is not based directly on the treatment group. Because D comes up third, in the data set, SYMBOL3 (diamond) is assigned to it. This can cause confusion to the reader. To remedy this situation we need to connect the symbol statement to the value of the treatment.

Start by creating macro variables for the treatments. The value of the macro variable is the SYMBOL statement without the keyword SYMBOL. Here we need the name of the treatment (a,b,c,d) to be in the name of its macro variable.

```plaintext
%LET TRTa = I=J V=CIRCLE L=7 C=BLACK;
%LET TRTb = I=J V=STAR L=3 C=BLACK;
%LET TRTc = I=J V=DIAMOND L=5 C=BLACK;
%LET TRTd = I=J V=DOT L=1 C=BLACK;
```

Then create a data set which contains all of the values of TPATT which are going to be present in the graph.

```plaintext
PROC SORT DATA=alltrt OUT=_trts(KEEP=tpatt) NODUPKEY
   BY tpatt;
RUN;
```

Using CALL EXECUTE we assign the symbols to their treatments.

```plaintext
DATA _NULL_; 
  SET _trts; 
  CALL EXECUTE ('SYMBOL'||LEFT(PUT(_N_,BEST8.))||' &trt'||LEFT(tpatt)||';'); 
RUN;
```

The following lines of code are generated.
When one group is omitted and the same code is used, the following lines of code are written:

```
20731 DATA _NULL_;
20732 SET trts;
20733 CALL EXECUTE ('SYMBOL'||LEFT(PUT( _BESTB.))||'! &trt||LEFT(tpatt))'';
20734 ;
20734 RUN;
```

```
NOTE: There were 3 observations read from the data set WORK._TRTS.
NOTE: DATA statement used:
   real time  0.00 seconds
   cpu time   0.00 seconds
```

Notice that SYMBOL3 is now V=DOT L=1, instead of V=Diamond L=5 as it was in the previous example. This code then creates the correct graph, where Treatment D has the same symbol as in all other graphs.
3. Data Management - Searching a Library or Libraries

This is a great tool for data checking and data cleaning. We can use CALL EXECUTE in conjunction with the metadata views to search through SAS data sets. For example, from experience, we know that dates are often entered incorrectly. So let’s run a check to capture all dates which cannot be correct, namely those after the database lock date.

Again, start with the code that needs to be looped through our DATA step.

```
PROC PRINT DATA=[dataset name] WIDTH=MIN N NOOBS LABEL;
  VAR subjid [date variables];
  WHERE [date variable] GT &lockdt. OR [date variable2] GT &lockdt. ... ;
RUN;
```

This example will require more DATA step manipulation to get our data in a form we can use.

For our code to work we need three variables:

1. `libdotmem` - for the DATA statement which contains the LIBNAME and MEMNAME of all the data sets which contain date values
2. `gtdate` – Contains the values for the WHERE statement
3. `name` – Contains the variable names for the VAR statement

For ease of adapting this program to new studies, create two macro variables. This makes the program macro-like without adding the confusion of a macro.

```
%LET lockdt='23SEP2005'D;
%LET lib=pharma;
```
PROC SQL;
CREATE TABLE names AS
SELECT DISTINCT COMPRESS(libname)|| '.' || COMPRESS(memname) AS libdotmem,
       TRIM(name)|| ' GT &lockdt. ' AS gtdate, name
FROM sashelp.vcolumn
WHERE format='DATE9.' AND UPCASE(libname)=UPCASE("&lib");
QUIT;

Resulting in the data set:

For the most efficient output we want to only run our PROC PRINT once per data set. For data sets with multiple dates, we need to compress the data to one observation per MEMNAME. In other words we need to change \textit{gtdate} and \textit{name} into horizontal lists that can be put directly (and syntactically) into our PROC PRINT statement.

DATA namelist(DROP=retclause retname);
LENGTH libdotmem name gtdate $200;
RETAIN retclause retname;
SET names;
BY libdotmem;
IF NOT(FIRST.libdotmem AND LAST.libdotmem) THEN DO;
  IF FIRST.libdotmem THEN DO;
    retclause=gtdate;
    retname=name;
  END;
  ELSE DO;
    retclause=TRIM(retclause)|| ' OR ' ||gtdate;
    retname=TRIM(retname)|| ' ' ||TRIM(name);
  END;
ENDIF;

IF LAST.libdotmem THEN DO;
  gtdate=retclause;
  name=retname;
  END;
END;
IF LAST.libdotmem;
RUN;

Resulting in the data set:
This data set is now ready to be simply plugged into our CALL EXECUTE.

```sas
DATA_NULL;
SET namelist;
CALL EXECUTE ('TITLE Dates after &lockdt. from ' || TRIM(libdotmem) || ' ;');
CALL EXECUTE ('PROC PRINT DATA=' || TRIM(libdotmem) || ' WIDTH=MIN N NOOBS LABEL;');
CALL EXECUTE ('  VAR subjid ' || TRIM(name) || ';');
CALL EXECUTE ('  WHERE ' || TRIM(gtdate) || ' ;');
CALL EXECUTE ('RUN;');
CALL EXECUTE ('TITLE;');
RUN;
```

This program can easily be used across projects and studies. The only necessary changes to the program are the values of the two macro variables at the top.

Conclusion:

CALL EXECUTE presents a solution to many problems that arise during the manipulation and reporting of clinical trial data. Using the techniques presented in this paper the members of the trial team can create programs that automatically adapt to changing data. This enables the program to be copied from study to study and project to project, without or with only few changes. This in turn saves time and leads to more consistent results.

References:
Contact Information:
You comments and questions are very much encouraged.

Daniel Boisvert  
Email: DJBoisvert@hotmail.com

Shafi Chowdhury  
Shafi Consultancy  
shafi@shaficonsultancy.com  
www.shaficonsultancy.com

Trademark Information:
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.