

Divide & Conquer: Simple Sub-Data Sets Creation with Call Execute

Dylan L. Holt, Westat; Wilhelmina Ross, Westat

ABSTRACT

When working with larger data sets, one often needs to isolate certain data based on a range of criteria to disseminate sub-data sets, perform analyses, and create reports. This paper demonstrates the use of the *Call Execute* routine to systematically create sub-data sets efficiently from a larger data set while maintaining consistency in format and naming conventions. In addition, the paper presents the flexibility of this method when dividing the entire data set into logical sub-data sets or selecting only certain criteria to create sub-data sets. Lastly, the paper demonstrates this method as a clear example of the powerful extension of the Macro facility within the DATA Step as mediated by the *Call Execute* routine.

INTRODUCTION

As data collectors, managers, and researchers, we interact on a daily basis with data sets containing tens, if not hundreds of thousands of records, and more often than not, our first task is to isolate a certain subset of that data set for specific analyses, report development, or dissemination to another group. This task is simple when isolating just one or even a small handful of sub-data sets from a larger one and may require substituting a few values for the parameters each time one is creating a sub-data set. However, such a task can become cumbersome and redundant at best and error-ridden at worst when attempting to create a large number of individual sub-data sets manually. Moreover, consistent formatting and naming conventions are harder to maintain when attempting this task one-by-one, and could lead to confusion when visiting and using these sub-data sets at a later time.

A clear, real world example emerges when collecting and analyzing data from hospitals at the state level. Within a state, there may be many hospitals and within each hospital, there can be tens, hundreds, or even thousands of new patients within a given year. The state will often collect and aggregate all of these data into a single data set in order to conduct larger population health studies and to determine overall health outcomes at the state level. The challenge, however, arises when the state may be responsible for reporting back to an individual hospital regarding specific health outcomes for the service area of that hospital or even communicating to the hospital potential errors in the quality and management of their data. This means that the state will need to deconstruct that data set into at very least as many sub-data sets as there are hospitals but may even need to separate out certain diseases resulting in many more sub-data sets per hospital.

To accomplish this task without the need to individually select each set of criteria for outputting a sub-data set from a main data set, we present a simple yet powerful solution incorporating SAS®'s Macro language and the *Call Execute* routine within a basic DATA Step. The syntax for the *Call Execute* routine is straight-forward:

```
call execute(argument)
```

The *Call Execute* routine resolves the argument and issues the resolved value for execution at the next step boundary. The power and flexibility of the *Call Execute* routine emerges as the passed argument in *Call Execute* routine can be a character expression that is resolved by the DATA step to macro text or to a SAS statement. In summary, this capacity enables us to generate SAS code within the DATA step using values from the set data. For example, consider the following program:

```
data _null_;
  input name $ age $;
  datalines;
    Dan 25
    Steve 37
    Bob 45
  ;
  call execute('%some_macro('||name||','||age||')');
run;
```

When submitted, the following code is generated and executed:

```
%some_macro(Dan, 25)
%some_macro(Steve, 37)
%some_macro(Bob, 45)
```

Output from a CALL EXECUTE Statement

From within the DATA step, we have generated macro text using the *Call Execute* routine that will take values from the set data as arguments and passed them into a macro for further processing.

In this paper, we demonstrate the use of the *Call Execute* routine to systematically create sub-data sets efficiently from a larger data set while maintaining consistency in format and naming conventions. In addition, we present the flexibility of this method when dividing the entire data set in logical sub-data sets, selecting only certain criteria to create sub-data sets, and extending the program to report development though additions on the DATA Step and Macro side of the existing program.

To demonstrate this method, we create a mock data set using random number generators to generate values for three fields – *FacilityID*, *PrimarySite*, and *TumorSize* – and then we assign meaning to values in fields *FacilityID* and *PrimarySite*. The resulting data set, named “mock_hosp” contains 20,000 records with 26 unique values in the *FacilityID* field and 8 unique values in the *PrimarySite* field. This mock data set is representative of our real world example as different *FacilityID* values represent different hospitals, different *PrimarySite* values represent different diseases, and *TumorSize* represents a specific characteristic of the disease for a given patient. The program used to create this mock data set is provided along with the main program in the Appendix. The goal now is to create a sub-data set containing records for each unique *FacilityID* by each unique *PrimarySite* using the Divide and Conquer method. In summary, we will create 208 sub-data sets (26 Facilities multiplied by 8 Primary Sites) from the main data set of 20,000 records.

METHODOLOGY

The Divide and Conquer method is comprised of three main components:

1. A Library to store the sub-data sets;
2. A Macro that will create the sub-data sets;
3. A DATA step with the Call Execute routine to pass arguments to the Macro.

CREATE A LIBRARY TO STORE THE SUB-DATA SETS

To create a library where the sub-data sets will be stored, we run a simple LIBNAME statement to point to a directory:

```
libname subs 'C:\SAS_Datasets\';
```

This statement creates a library with a libref of “subs” which we will use to store our sub-data sets in the folder with the file path “C:\SAS_Datasets”.

DEVELOP A MACRO FOR CREATING SUB-DATA SETS

The next step is to develop the macro which will later be called to create the sub-data sets. Knowing that we want to create a sub-data set containing records for each unique *FacilityID* by each unique *PrimarySite*, the macro will need minimally two capacities: 1) the capacity to pass in at least two parameters and 2) the capacity to query the main data set on those parameters and to output the results. For our purposes, the following macro, “create_subs”, will suffice:

```
%macro create_subs(var1, var2);
  proc sql;
    create table subs.&var1._&var2. as (1)
      select * from mock_hosp
      where FacilityID = "&var1." and PrimarySite = "&var2."; (2)
  quit;
%mend;
```

Using a PROC SQL statement, we create a data set that will be stored in the library “subs”. We use a combination of the same two arguments to name the data set (1) and to query the main data set “mock_hosp” (2), ensuring a consistent naming convention when we later will pass many sets of parameters as well as consistency between the name of the sub-data set and its contents.

WRITE DATA STEP WITH THE CALL EXECUTE ROUTINE

At this point, we have not interacted with any data. We have created a library where ultimately the sub-data sets will be stored and have developed a macro that will create said sub-data sets based on parameters it receives. The last, but certainly not least, step of the Divide and Conquer method is writing a basic DATA step which contains the *Call Execute* routine and will pass values for *FacilityID* and *PrimarySite* to the *create_subs* macro.

Before we do that, however, we need to group the main data set so that our last step will only pass each unique combination of *FacilityID* and *PrimarySite* once to the macro. To accomplish this task, we use the following PROC SQL statement to group the main data set by *FacilityID* and *PrimarySite*:

```
proc sql;
  create table hosp_mock_groups as
    select FacilityID, PrimarySite, Count(*)
    from mock_hosp
    group by FacilityID, PrimarySite;
quit;
```

This results in a summarized data set which contains one observation for each unique combination of *FacilityID* and *PrimarySite*. We set this summarized data set in our DATA step with the *Call Execute* routine:

```
data _null_;
  set hosp_mock_groups;
  call execute('%create_subs('||FacilityID||','||PrimarySite||')');
run;
```

Due to the implicit looping of any DATA step, the *Call Execute* routine runs for every record in our summarized data set and due to the grouping performed prior to running this DATA step this generates and executes the following for each unique combination of *FacilityID* and *PrimarySite*:

```
%create_subs(hospital_1, bladder)
%create_subs(hospital_1, breast)
%create_subs(hospital_1, colon)
%create_subs(hospital_1, lung)

...[Beginning and end shown for brevity]

%create_subs(hospital_26, prostate)
%create_subs(hospital_26, rectum)
%create_subs(hospital_26, skin)
%create_subs(hospital_26, stomach)
```

Output from a CALL EXECUTE Statement

As expected, these arguments are passed to our *create_subs* macro which in turn uses the arguments to create a sub-data set. The sub-data set is named in the consistent format “*FacilityID_PrimarySite*”, and contains only cases for the specified *FacilityID* and *PrimarySite* from the main data set. To check these results, we can use the Windows® command line to navigate to the folder we assigned earlier to store the sub-data sets and list its contents:

Command Line:

```
>> cd C:\SAS_Datasets
>> ls
```

```
hospital_01_bladder.sas7bdat
hospital_01_breast.sas7bdat
hospital_01_colon.sas7bdat
hospital_01_lung.sas7bdat
hospital_01_prostate.sas7bdat
hospital_01_rectum.sas7bdat
hospital_01_skin.sas7bdat
hospital_01_stomach.sas7bdat
hospital_02_bladder.sas7bdat
hospital_02_breast.sas7bdat
hospital_02_colon.sas7bdat
hospital_02_lung.sas7bdat
hospital_02_prostate.sas7bdat
hospital_02_rectum.sas7bdat

...[Beginning and end shown for brevity]

hospital_25_breast.sas7bdat
hospital_25_colon.sas7bdat
hospital_25_lung.sas7bdat
hospital_25_prostate.sas7bdat
hospital_25_rectum.sas7bdat
hospital_25_skin.sas7bdat
hospital_25_stomach.sas7bdat
```

```

hospital_26_bladder.sas7bdat
hospital_26_breast.sas7bdat
hospital_26_colon.sas7bdat
hospital_26_lung.sas7bdat
hospital_26_prostate.sas7bdat
hospital_26_rectum.sas7bdat
hospital_26_skin.sas7bdat
hospital_26_stomach.sas7bdat

```

Output from COMMAND LINE Execution

We have successfully created 208 sub-data sets from our main data set of 20,000 records, a sub-data set for each *PrimarySite* in each *FacilityID*.

EXTENSIONS OF THE CURRENT PROGRAM

In the above demonstration, we simply create a sub-data set for every unique combination of *FacilityID* and *PrimarySite*. However, following our real world example, we may receive a request from a hospital to provide them with data regarding a specific set of diseases or even only sub-data sets that contain a certain threshold of cases per disease. Additionally, we may also need to provide some basic descriptive statistics on each sub-data set along with the sub-data set itself. Our current program is easily extensible to such requests on both the DATA step and Macro sides.

EXAMPLE 1: SELECTIVE SUB-DATA SET CREATION

In terms of selecting only certain sub-data sets to create, we can simply add any number of conditions where the *Call Execute* routine will only generate macro text if the conditions are met. Below is an example of this selectivity:

```

data _null_;
set hosp_mock_groups;
if PrimarySite in ('Breast','Lung','Colon') and Count > 80;
call execute('%create_subs('||FacilityID||','||PrimarySite||')');
run;

```

For this example, we only want sub-data sets to be created from the main data set for *Breast*, *Lung*, or *Colon* cases (*PrimarySite*) in hospitals (*FacilityID*) that have more than 80 cases in these *PrimarySite* categories. As a note, we create the *Count* variable used in this example when we group the main data set by *FacilityID* and *PrimarySite*. With this program, we selectively created only 72 sub-data sets that meet the applied conditions:

```

Command Line:
>> cd C:\SAS_Datasets
>> ls

```

```

hospital_01_breast.sas7bdat
hospital_01_lung.sas7bdat
hospital_02_breast.sas7bdat
hospital_02_colon.sas7bdat
hospital_02_lung.sas7bdat

...[Beginning and end shown for brevity]

hospital_25_breast.sas7bdat
hospital_25_colon.sas7bdat
hospital_25_lung.sas7bdat
hospital_26_breast.sas7bdat
hospital_26_colon.sas7bdat
hospital_26_lung.sas7bdat

```

Output from COMMAND LINE Execution

Due to the consistent naming conventions controlled by the macro, we can easily separate, package, and send these sub-data sets to the correct hospitals. We can easily manage selectivity on the DATA step side of our program.

EXAMPLE 2: ADDITIONAL ANALYSES FOR EACH SUB-DATA SET

We can also easily extend the Macro “create_subs” to perform additional transformation or analyses on each sub-data set as it is created by simply adding additional statements. Consider the following extended macro:

```
%macro create_subs(var1, var2);

  proc sql;
    create table subs.&var1._&var2. as
      select * from mock_hosp
      where FacilityID = "&var1."
      and PrimarySite = "&var2.";
  quit;

  proc freq data=subs.&var1._&var2.; (1)
    table Tumorsize;
    where Tumorsize > '500';
    title "&var2. Cases with Large Tumor Sizes in &var1."; (2)
  run;

%mend;
```

This macro still creates a data set that will be stored in the library “subs”, which will be named based on the passed arguments as well as contain the results of the query on the main data set. However, we now also output frequencies for *TumorSize* values greater than 500. Moreover, using the arguments passed to the macro from the *Call Execute* routine allows us to 1) select that sub-data set just created and 2) establish consistent naming conventions of title of the frequency table.

Both of these examples depicting the simple yet powerful extension of the current program, provide a foundation for the types of extended capabilities available in our Divide and Conquer method. Additionally, we demonstrate clear separation of concerns by controlling selectivity on the DATA step side and writing any analyses or transformation on the Macro side. Separating out these two primary roles yields cleaner programs that can be updated and debugged more easily. Making our programs more modular also allows us to adapt or reuse our programs in different circumstances and settings.

CONCLUSION

As the sheer amount of data we collect, manage, or explore continues to grow, we need to have available tools that can parse and separate these data into meaningful chunks; however, we cannot lose track of where these meaningful chunks belong in the larger picture and need to embed their individual meaning in titles and names of the sub-data sets we create. We conclude that by using the *Call Execute* routine, we can write simple yet powerful programs that create sub-data sets and more from larger main data sets while maintaining consistent naming conventions. The Divide and Conquer method is a clear example of the powerful extension of the Macro facility within the DATA Step as mediated by *Call Execute* routine. Using these tools, we create and manage sub-data sets without creating a headache for our clients, colleagues, and ourselves.

REFERENCES

- Usov, A. (2014) Call Execute: Let Your Program Run Your Macro. Retrieved from <http://www.lexjansen.com/phuse/2014/cc/CC06.pdf>
- Michel, D. (2005) CALL EXECUTE: A Powerful Data Management Tool. Retrieved from <http://www2.sas.com/proceedings/sugi30/027-30.pdf>
- Whitlock, H. I. (1997) CALL EXECUTE: How and Why. Proceedings of the Twenty-Second Annual SAS Users Group International Conference.

CONTACT INFORMATION

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

Name: Dylan L. Holt
Enterprise: Westat
Address: 1600 Research Boulevard
City, State ZIP: Rockville, MD 20850-3129
Work Phone: 301-251-2245
Fax: 240-314-2377

E-mail: dylanholt@westat.com
Web: www.westat.com

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

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 trademarks of their respective companies.

APPENDIX

```

/**Simple Sub-Data Sets Creation with Call Execute**/

/*Create Data Set for Demonstration Purposes*/

%let NObs = 20000;
data rand_nums;
  call streaminit(123);
  xMax = 25;
  yMax = 7;
  zMax = 999;
  do i = 1 to &NObs.;
    xRan = floor((1+xMax)*rand("Uniform"));
    yRan = floor((1+yMax)*rand("Uniform"));
    zRan = floor((1+zMax)*rand("Uniform"));
    output;
  end;
run;

/*Code Random Numbers to Something Meaningful*/

data mock_hosp (keep= FacilityID PrimarySite TumorSize);
  length FacilityID PrimarySite $15;
  set rand_nums;
  FacilityID = 'Hospital_' || put(input(cats(xRan+1), 2.), z2.);
  if yRan = 0 then PrimarySite = 'Breast';if yRan = 1 then PrimarySite = 'Lung';
  if yRan = 2 then PrimarySite = 'Colon';if yRan = 3 then PrimarySite =
'Prostate';
  if yRan = 4 then PrimarySite = 'Bladder';if yRan = 5 then PrimarySite = 'Skin';
  if yRan = 6 then PrimarySite = 'Rectum';if yRan = 7 then PrimarySite =
'Stomach';
  TumorSize = put(input(cats(zRan), 3.), z3.);
run;

/*MAIN PROGRAM BELOW*/

/*Library to Store Sub-Data Sets*/

libname subs 'C:\SAS_Datasets\';

/*Macro to be executed by Call Execute to Create Sub-Data Sets*/

%macro create_subs(var1, var2);

  proc sql;
    create table subs.&var1._&var2. as
      select * from mock_hosp
      where FacilityID = "&var1." and PrimarySite = "&var2.";
  quit;

  /*Sub-Data Set has been created. Add more to macro to perform additional
transformations/analyses on this Sub-Data Set*/
  /*Un-Comment to run below statement*/
  /* proc freq data=subs.&var1._&var2.;
    table Tumorsize;
    where Tumorsize > '500';
    title "&var2. Cases with Large Tumor Sizes in &var1.";
  run;
*/

%mend;

/*Group Main Data set to create one observation by unique combination of FacilityID
& PrimarySite*/

```

```
proc sql;
  create table hosp_mock_groups as
    select FacilityID, PrimarySite, Count(*) as Count from mock_hosp
    group by FacilityID, PrimarySite;
quit;

/*DATA Step with Call Execute*/

data _null_;
  set hosp_mock_groups;
  /*Add Conditions for more Selective Runs*/
  /*Un-Comment to run below statement*/
  /*if PrimarySite in ('Breast','Lung','Colon') and Count > 80;*/
  call execute('%create_subs('||FacilityID||','||PrimarySite||')');
run;
```