PharmaSUG 2020 - Paper QT-183

A brief understanding of DOSUBL beyond CALL EXECUTE

Ajay Sinha, Novartis, India
M Chanukya Samrat, Novartis, India

ABSTRACT:

SAS® is one of the most widely used programming language in clinical space, the newer release of SAS® 9.3 R2 and above offers a function "DOSUBL" that can come very handy for programmers to shorten and make code more efficient. This paper intends to explain the use of DOSUBL function with help of some examples that can make code more robust and flash the best use of this function within the code.

Function DOSUBL enables immediate execution of SAS code after a text string is passed. It is somewhat similar to CALL EXECUTE routine however differs significantly, this paper intends to present the merits and pitfalls of DOSUBL function and where it can come handy to make most use of this function.

The DOSUBL function has a single argument, which is a string value. In a data step, the function submits code to SAS and waits for that code to complete submission. DOSUBL should be used in a DATA step. SAS documentation states that this function can also be used with %SYSFUNC outside a step boundary. DOSUBL executes code in a different SAS executive (known as a side session). This function is comparatively slower than CALL EXECUTE subroutine and uses more CPU resources however, the main advantage of using DOSUBL is immediate execution of the code in the side session.

This paper has four sections with examples in each section.

1) Introduction
2) Basic Difference between DOSUBL and CALL EXECUTE.
3) DOSUBL fails but when
4) Code enhancement with an example

INTRODUCTION:

The DOSUBL function has a single argument, which is a string value. In data step, the function submits code to SAS and waits for that code to complete submission. DOSUBL executes code in a different SAS executive (known as a side session).

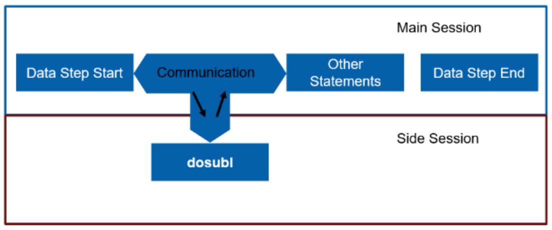
Example 1:

By looking at the example given below we can understand that we create a macro and then use the macro within a data step to perform the task we need for each observation within a data step.

```
%macro test1; %put this is test1; %mend;
%test1;
data _null_; rc = dosubl('%test1;'); run;
```

Result:

```
this is test1
this is test1
```



BASIC DIFFERENCE BETWEEN DOSUBL AND CALL EXECUTE:

CALL EXECUTE	DOSUBL
Is a Subroutine	Is a Function
Let you stack up statements, which will run once the current DATA step completes	Does not stack up statements, which will run once it encounters the function in the current DATA step
Comparatively Fast	Comparatively Slow

Example 1:

```
data _null_;
put "one";
call execute('data _null_; put "two";run;');
put "three";
run;
```

Result:

```
One
Three
Two
```

```
data _null_;
put "one";
rc = dosubl('data _null_; put "two";run;');
put "three";
run;
```

Result:

```
One
Two
Three
```

In the example of call execute we see that the data step is executed first and then the call execute is called and executed, however when we use dosubl then the statements are executed and the output is passed immediately

Example 2:

```
data BMW(keep=count);
call execute("
data _null_;
      set sashelp.cars;
      where make='BMW';
      totobs+1;
      call symputx('cnte',totobs);
run;");
count=symget("cnte");
run;
```

Result: Missing with _ERROR_=1

```
data BMW(keep=count);
rc=dosubl("
data _null_;
      set sashelp.cars;
      where make='BMW';
      totobs+1;
      call symputx('cntd',totobs);
run;");
count=symget("cntd");
run;
```

Result: 20

In this example when we use call execute we see that there is an error, as the symget is not able to find the macro variable, this is because data step is executed first and then the call execute is called, however when we use DOSUBL then the statements are executed immediately, also the macro variable is created immediately.

DOSUBL FAILS BUT WHEN

When we are trying to refer the calling environment (dataset name or variables) into the DOSUBL function then this function would return a RC (Return Code) not equal 0 so be cautious about when using this function.

Example1:

```
data a;
rc = dosubl('data a; set a; x=1; run;');
set a;
run;
```

log : ERROR: File WORK.A.DATA does not exist.

Another example in the SAS documentation where the DOSUBL fails is to try to assign the LIBNAME statement within the function, because LIBNAME in dosubl executed in the side session it will not be available for us in the calling environment.

Example2:

```
libname outside "C:\data";
data _null_;
   rc = DOSUBL('LIBNAME inside "C:\newData";');
   %put outside = %sysfunc(pathname(outside));
   %put inside = %sysfunc(pathname(inside));
run;
```

Result : outside = C:\data   inside =

CODE ENHANCEMENT WITH AN EXAMPLE

The code comparison shown below is based on the traditional method of using the back merge process, using DOSUBL and HASH concept and achieving the same result.

```
proc copy in=sashelp
out=work memtype=data;
   select class;
run;

proc summary
data=class nway;
class sex;
var age;
output
out=mage(drop=_:)
mean=agemean;
run;

proc sort data=mage;
by sex;
run;

proc sort data=class;
by sex;
run;

data class1;
merge class(in=a) mage(in=b);
by sex;
if a and b;
run;
```

```
proc copy in=sashelp
out=work memtype=data;
   select class;
run;

data class1;
if _N_ = 1 then do;

rc=dosubl
('proc summary data=class nway;
class sex;
var age;
output out=mage(drop=_:)
mean=agemean;
run;');

declare hash
myhs(dataset:"mage");
myhs.definekey("sex");
myhs.definedata("agemean");
myhs.definedone();
end;
set class;
if myhs.find() ne 0 then
call missing(agemean);
run;
```

The code on the left shows the traditional approach in which we first create the summary statistic mean of the age by sex then sort both the datasets and then merge them based on the key variable sex. In this case, we are using three additional steps. If asked which is the most time consuming and resource intensive PROC, then certainly it would be PROC SORT.

The code on the right shows that the same dataset can be achieved in a single data step, with the use of DOSUBL function and HASH objects. In a single data step, we create the statistic using the DOSUBL function and merge the data back with the parent domain using the HASH Objects.

There are many ways we can accomplish the same task in SAS. In the given examples both these methods run successfully on smaller datasets but as the data becomes huge then DOSUBL and HASH Objects can come to our rescue.

CONCLUSION

The DOSUBL function is a very powerful addition to repository of SAS functions. It is able to execute SAS code like a data step function. It also allows you to perform your task with minimal code, however used needs to be cautious when using it on large datasets.

CONTACT INFORMATION

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

Name: Ajay Sinha
Enterprise: Novartis
Address: India
City, State ZIP: Mumbai , Maharashtra 400051
Phone: +91 22 5024 3000
E-mail: ajay.sinha@novartis.com ; ajaysinhacr@gmail.com ;
Web: https://www.novartis.in

Name: M Chanukya Samrat
Enterprise: Novartis
Address: India
City, State ZIP: Hyderabad, Telangana 502032
Work Phone: +91 40 6758 1000
E-mail: chanukya_samrat.m@novartis.com
Web: https://www.novartis.in