

Using CALL EXECUTE to Generate Different Report Forms

Danna Chen, PPD Inc., Beijing

ABSTRACT

In the pharmaceutical industry, according to the actual data, we often need to output different report forms by using proc steps. For example, the output of blinded data and unblinded data are different. Call Execute statement can be used to execute procedure conditionally. This paper will outline this approach to accomplish these tasks and discuss the usage of Call Execute where this subroutine come up with a better alternative to the usual way.

INTRODUCTION:

Call Execute is very useful function which can be used in many different situations for many different purposes. It enables a program to Interact with the macro facility during the data step execution. It is used within a data step, and unlike most macro conditionals, can be used in open code. Create a data set which contains the values you want to put into your basic statement.

THE CALL EXECUTE SYNTAX

It has the following syntax:

Call execute(argument);

where argument can be one of three elements:

1) a character string enclosed in quotation marks:

Call execute('Proc print data=adb.adcm; run;');

2) the name of a variable contained within the dataset processed within the data step:

Call execute('Proc print data=work.||param||'; run;');

3) a character expression that resolves to a SAS statement or macro text expression:

Call execute('%report(file=||left(filename)||)'; run;');

Now let's get into the details of how the CALL EXECUTE routine can be used for, take a look at some simple example.

EXAMPLE 1-A PARAMETERS LIST.

In many situations we needed to use the same procedures on different parameters to output some data management listing, Usually we do that using many different data steps or by creating a macro for the procedure and calling that as many times as we need. But it's tedious and easy to make a mistake. Call Execute enable us to do this within a single data step as below.

Now lets consider a case in which we want to print the item that Laboratory results beyond the normal range, but Clinical significance fill in as normal. We will output only those subjects that have at least one abnormal data.

Sign with CoSign								
I3								
	A	B	C	D	E	F	G	H
1	check_no	detail_no	lower	upper	paramcd	Clinical_sign	param	
2	8000050	11	LLT10	ULT10	ALT	SALT	param1	
3	8000050	12	LLT9	ULT9	AST	SAST	param2	
4	8000050	13	LLT8	ULT8	ALP	SALP	param3	
5	8000050	14	LLT14	ULT14	TP	STP	param4	
6	8000050	15	LLT15	ULT15	ALB	SALB	param5	
7	8000050	16	LLT16	ULT16	TB	STB	param6	
8	8000050	17	LLT11	ULT11	CR	SCR	param7	
9	8000050	18	LLT20	ULT20	CA	SCA	param8	
10	8000050	19	LLT19	ULT19	NA	SNA	param9	
11	8000050	20	LLT17	ULT17	K	SK	param10	
12	8000050	21	LLT18	ULT18	MG	SMG	param11	
13								
14								
15								

We have an edit check specification like above, each parameter has the different upper limit and lower limit. In the usual way, we need to judge the results for each parameter. For many parameters, it would take a lot of code to generate each individual dataset field. Using Call Execute, the data step operates as a loop, we can keep the code much simpler.

	SUBJID	SALT	SAST	LLT9	ULT9	LLT10	ULT10	ALT	AST
1	01080	1	1	0	42	0	40	11	16
2	01080	1	1	0	42	0	40	11	16
3	01084	1	1	0	42	0	40	40	23
4	01084	1	1	0	42	0	40	29	17
5	03024	1	1	0	40	0	40	12.7	15.3
6	03024	1	1	0	40	0	40	16.1	16.4
7	03024	1	1	0	40	0	40	16.1	16.4
8	03024	1	1	0	40	0	40	12.7	15.3
9	03024	1	1	0	40	0	40	15.4	18.9
10	04025	1	1	0	60	0	60	24	34
11	04025	1	1	0	60	0	60	36	44
12	04025	1	1	0	60	0	60	28	34
13	04025	1	1	0	60	0	60	29	37
14	04025	1	1	0	60	0	60	23	33
15	04025	1	1	0	60	0	60	23	33
16	04038	1	1	0	60	0	60	44	23
17	04038	1	1	0	60	0	60	18	16
18	04038	1	1	0	60	0	60	18	16
19	04047	1	1	0	60	0	60	21	16
20	04047	1	1	0	60	0	60	30	19
21	04047	1	1	0	60	0	60	20	16
22	04047	1	1	0	60	0	60	14	13
23	04047	1	1	0	60	0	60	12	13
24	04047	1	1	0	60	0	60	12	13
25	04069	1	1	0	60	0	60	22	23
26	04069	1	1	0	60	0	60	14	17
27	04069	1	1	0	60	0	60	9	16
28	04069	1	1	0	60	0	60	9	16

```
%let xlsfile=C:\pharmasug\edit check.xls;
%xls2sas(edit,sheet1);/*Import Excel file into SAS dataset*/
```

Let's import the specification to a dataset EDIT which contains the names of all parameters, along with upper limit, lower limit and some other variables.

```

data _null_;
  set edit;
  call execute ('data ' || trim(paramcd) || ';' );
  call execute ('set output.lbc;');
  call execute (' where (' ||trim(paramcd)|| '<' ||trim(lower) ||' or
  '||trim(paramcd)|| '>' ||trim(upper)||') and '|| trim(clinical__sign) ||'=1;' );
  call execute ('discrepancy="blood biochemical item:the value of
  "||left(trim('||paramcd||'))||" is ("||trim(left('||paramcd
  ||'))||"),normal range is ("||trim(left(');
  call execute (trim(lower) ||')||" , "||trim(left('||trim(upper)||'))||
  "),the outcome is not in the normal range,but the clinical significance is
  normal,please check it again.";');
  call execute ('keep subjid discrepancy;');
  call execute ('run;');

  call execute ('proc print data=' || trim(paramcd) || ';' );
  call execute ('run;');
run;

```

Using dataset EDIT, we can obtain the desired result of all dataset to satisfy the original request entirely without manual intervention using Call Execute.

In this procedure we can see:

- 1) CALL EXECUTE ('data ' || TRIM(paramcd) || ';')

In this step "paramcd" will execute a parameter List –ALT,AST,ALP,TP,ALB,TB,CR,CA,NA,K,MG;

- 2) CALL EXECUTE (' WHERE (' ||TRIM(paramcd)|| '<' ||TRIM(lower) ||' OR '||TRIM(paramcd)|| '>' ||TRIM(upper)||') and '|| TRIM(Clinical__sign) ||'=1;')

Any variable names not enclosed in quotes get resolved to their values. such as paramcd, lower, upper must be included in EDIT dataset ,if not, it will be a missing value. when you need to use the variable in lbc dataset but not list in EDIT,you can set it on inside quotation marks.for example:

```

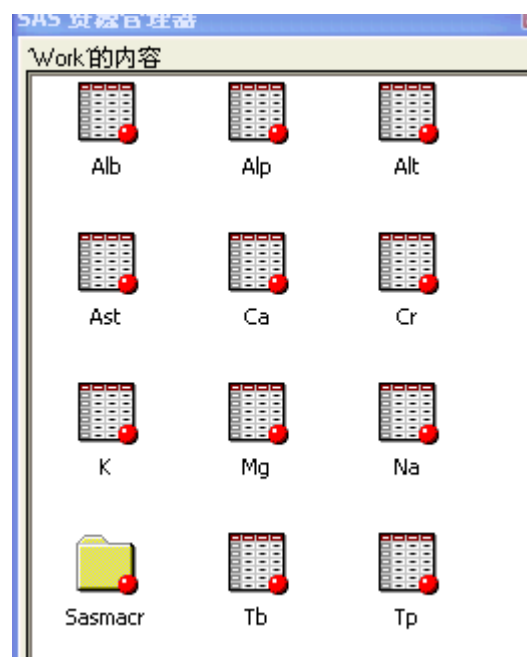
CALL EXECUTE (' WHERE VNUM=2 AND(' ||TRIM(paramcd)|| '<' ||TRIM(lower) ||' OR
  '||TRIM(paramcd)|| '>' ||TRIM(upper)||') and '|| TRIM(Clinical__sign) ||'=1;' )

```

The variable vnum is in the lbc dataset. If you set it out of quotation marks, it also execute as a missing value.

- 3) Note of caution: The argument of each Call Execute should not exceed 256 characters. In addition, if the length of successive Call Execute invocations sums to 32,767 characters or greater, errors may also occur. So in here we have to separate the statement "discrepancy=" into two call execute statement. It will make some errors if set it together.
- 4) You can use many Call Execute in a single data step. The code they generate just continues to stack up to be run once the current data step completes. Call Execute helps us combine data steps and proc steps together and no need to type same piece of code again and again.

At last, we can obtain 11 datasets which include the abnormal data.



Each dataset contains all subjects which meet selection criteria.

	SUBJID	discrepancy
1	06023	Blood biochemical item: The value of ALB is (4.51), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
2	06048	Blood biochemical item: The value of ALB is (4.36), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
3	06048	Blood biochemical item: The value of ALB is (4.5), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
4	06048	Blood biochemical item: The value of ALB is (4.65), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
5	06048	Blood biochemical item: The value of ALB is (4.65), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
6	06061	Blood biochemical item: The value of ALB is (4.13), Normal range is (35,50). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
7	07007	Blood biochemical item: The value of ALB is (40.5), Normal range is (...). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
8	07011	Blood biochemical item: The value of ALB is (33.6), Normal range is (35,55). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
9	07055	Blood biochemical item: The value of ALB is (65.8), Normal range is (35,55). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
10	08027	Blood biochemical item: The value of ALB is (39.1), Normal range is (60,80). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
11	08027	Blood biochemical item: The value of ALB is (37.8), Normal range is (60,80). The outcome is not in the normal range, But the clinical significance is normal, please check it again.
12	08027	Blood biochemical item: The value of ALB is (36), Normal range is (60,80). The outcome is not in the normal range, But the clinical significance is normal, please check it again.

EXAMPLE 2- EXECUTE MACRO CONDITIONALLY

Suppose we need to generate different outputs for blinded and unblinded data. There are many ways to cover this task. You will find Call Execute is efficient and makes code be compactness.

The raw data is as below:

	PAT_NO	V_NAME_REP	grp	DOSSORT
1	01002	Week 12	1	1
2	01003	Week 12	1	2
3	01004	Week 12	0	1
4	01005	Week 12	1	2
5	01006	Week 12	1	1
6	01007	Week 12	1	2
7	01008	Week 12	1	2
8	01009	Week 12	1	1
9	01011	Week 12	1	1
10	01012	Week 12	1	1
11	01013	Week 12	1	1
12	01014	Week 12	1	2
13	01015	Week 12	1	2
14	01018	Week 12	1	2
15	01019	Week 12	1	2
16	01020	Week 12	1	1
17	01021	Week 12	1	1
18	01023	Week 12	1	2
19	01024	Week 12	1	1
20	01025	Week 12	0	1
21	01026	Week 12	1	1
22	01027	Week 12	0	1
23	01028	Week 12	0	1
24	01029	Week 12	1	2
25	01031	Week 12	1	2
26	01032	Week 12	0	1
27	02002	Week 12	1	1
28	02005	Week 12	1	2
29	02007	Week 12	1	2
30	02010	Week 12	0	1
31	02011	Week 12	1	1

```

proc format;
  value grpbody
    0="negative"
    1="positive"
  ;
run;
%macro unblind;
proc freq noprint data=effect;
  table dossort*grp/out=freq;
run;

data last;
  merge
    freq(where=(dossort=1) rename=(count=col1 percent=pct1))
    freq(where=(dossort=2) rename=(count=col2 percent=pct2));
  by grp;
  if col1=. then col1=0;if col2=. then col2=0;
  rename grp=name;

```

```

        keep grp col1 col2;
run;

proc report data=last split="|" nowindows headline ;
title 'hbv dna below 50 iu/ml by pcr through week 12-randomized cohort ';
    column name col1 col2 ;
    define name      /display format=grpbody.   style(column)=[cellwidth=8cm just=l]
    "characteristic";
    define col1 /display style(column)=[cellwidth=6cm just=c] "treatment group";
    define col2      /display style(column)=[cellwidth=6cm just=c] "placebo group";
run;
%mend unblind;

%macro blind;
proc freq noprint data=effect;
    table grp/out=freq;
run;

data last;
    set freq;
    rename grp=name count=col1;
    keep grp count;
run;

proc report data=last split="|" nowindows headline ;
title 'hbv dna below 50 iu/ml by pcr through week 12-randomized cohort ';
    column name col1 ;
    define name      /display format=grpbody.   style(column)=[cellwidth=8cm just=c]
    "characteristic";
    define col1      /display style(column)=[cellwidth=12cm just=c] "total";
run;
%mend blind;

```

According to the actual data, we can write two macros UNBLIND and BLIND to accomplish these work. In the unblinded data, we need to compute the frequency of dossort cross grp, But in the blinded data, we must combine the two different groups into total group.

```

%global blind;
%let blind=0;
Set a macro parameters and give an original value to blind.
data _null_;
    if &blind=1 then call execute('%blind');
    if &blind=0 then call execute('%unblind');
run;

```

In this procedure we can see:

1) Call Execute with macros is far more efficient and powerful than Call Execute standalone.

2) You can't use %IF %THEN %ELSE or %DO %END outside of a macro definition. Call Execute comes to the rescue.

3) If the macro have a parameter, you can't do this:

```
if &blind=1 then call execute('%blind(dat)');
```

This instruction can't work what you want, it will treat variable 'dat' as a character string, you can write it like this:

```
if &blind=1 then call execute(cats('%blind(', dat, ') '));
```

the blinded output:

HBV DNA Below 50 IU/mL by PCR through Week 12-Randomized Cohort

CHARACTERISTIC	Total
negative	9
positive	157

the unblinded output:

HBV DNA Below 50 IU/mL by PCR through Week 12-Randomized Cohort

CHARACTERISTIC	Treatment Group	Placebo Group
negative	9	0
positive	80	77

CONCLUSION:

The examples in this paper illustrate how to use Call Execute. To use Call Execute, two features must be in place.

Firstly, the macro processor must be turned on. Secondly, the data step must end with a run statement.

The Call Execute is a valuable subroutine. It is particularly powerful when used to invoke macros or to pass variables into a parameter list. This approach can significantly reduce programming development and maintenance costs. Once the basics of this method are understood, programming complex logic rules can be a simple task.

On the other hand, there are some limitations of it. Firstly, macro variables defined in the Call Execute argument cannot be referenced until the data step processing the call is completed. Secondly, the argument of each Call Execute should not exceed 256 characters.

One must really understand how SAS code and macro code are processed in order to avoid the pitfalls.

CONTACT INFORMATION

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

Name: Danna Chen

Enterprise: PPD Inc.

Address: 8/F, Tower B, Central Point Plaza No 11, Dongzhimen South Ave. Dongcheng District

City, State ZIP: Beijing, 100007

Work Phone: +86 10-57636250

Fax: +86 10-57636251

E-mail: Danna.chen@ppdi.com

Web: www.ppdi.com

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.