# Passing Data Set Values into Application Parameters

Lingqun Liu
The University of Michigan, Ann Arbor

## ABSTRACT

Passing data set values into different applications may require different techniques depending on the designs of the applications. This paper covers three of these techniques: 1. using the DATA _NULL_ and PUT statements to generate a temporary external code file; 2. using PROC SQL to store generated code into a macro variable; 3. using CALL EXECUTE routine within a DATA _NULL_ step to put generated code in the input stack. All of them will utilize the SAS Macro. Examples are used to illustrate these three methods.

## KEYWORDS

Data Driven SAS Application, Application Parameters, DATA _NULL_, PROC SQL, CALL EXECUTE, %SUPERQ, %NRSTR, SAS Input Stack, SAS Macro Processor

## INTRODUCTION

This paper contains four sections: I. Two types of “data driven” application designs; II. Using DATA _NULL_ and PUT statements; III. Using PROC SQL; IV. Using CALL EXECUTE routine. The first section discusses the two types of data driven SAS application design. The following three sections discuss three techniques that can be used to pass data set values into application parameters.

## I. TWO TYPES OF “DATA DRIVEN” APPLICATION DESIGNS

To generate data specific analytical or presentation results requires dynamic, “data driven” SAS applications. Suppose you have a data set containing information for 5000 dialysis facilities in the United States. You are asked to develop a SAS application to produce 5000 facility-specific data reports, one for each facility, from the data set. To accomplish this task you need to develop a SAS application and then execute it for each record in the data set. This application is a typical example of “data driven” applications.

To pass the input data set values into the application parameters, you need to develop a “driver” program so that each record will “drive” the application running once. “Data driven” applications can be developed as either macro applications or non-macro applications.

## 1. Macro Style Design

Data driven applications can be defined as *SAS macros*. Data set values will be passed to applications via *macro parameters*. Within macro definitions, sub-macros and regular SAS programs may be called. Here is the structure and examples:

%macro *application_name* (*parameter list*);
… Macro statements …
….SAS data step statements …
….SAS Procedure statements …
%mend;

**Example 1 (macro application without %include):**

```
*\example1.sas;
%macro m2(provfs);
    %put inside macro m2
&provfs;
    data m2_&provfs;
         m2="&provfs";
    run;
%mend;

%macro wo_inc(provfs);
    %m2(&provfs);
    data m1_&provfs;
          m1=&provfs;
    run;
%mend;
```

**Example 2 (macro application with %include):**

```
* \example2_inc.sas;
data inc_&provfs;
   inc="&provfs";
run;

* \example2.sas;
%macro m2(provfs);
  data m2_&provfs;
     m2="&provfs";
   run;
%mend;
%macro w_inc(provfs);
  %inc "example2_inc.sas";
  %m2(&provfs);
  data m1_&provfs;
   win=&provfs;
  run;
%mend;
```

**Example 3 (non-macro application):**

```
* \example3_1inc.sas;
proc print data=&dsn;
       where(date="&dat");
       var &var;
run;

* \example3_2inc.sas;
proc chart data=&dsn;
       where(date="&dat");
       vbar &var;
run;

* \example3.sas;
%inc "example3_1inc.sas";
%inc "example3_2inc.sas";
```

*Note: You can %includes regular SAS program inside %macro as seen in example 2, although I do not recommend this programming style.*

## 2. Non-macro Style Design

Data driven SAS applications can be developed as a set of well-organized modules and a main program. The modules are just regular SAS programs. The main program needs to call or invoke the modules with *%include*. Data step values are passed to the application's main program via macro variables. Large and complex applications will benefit from this structure. This modularity design allows multiple programmers to work on different modules simultaneously. This design will also make the application easier to understand and thus improve the maintainability of the applications. (Modularity design can also be applied to macro style applications described above. However, because complex applications usually have a large set of modules; it is not desirable to convert all modules to macros which will make developing and debugging processes more difficult and the application less efficient.) Here is the application structure:

Main.sas
  %let *macrvar* =*data step value*;
  %inc '*xx_inc*.sas';
  %inc '*xx_inc*.sas';

### 3. Passing data set values into data driven applications

The mechanism of passing data set values into data driven applications is the SAS macro facility. The macro style application will use macro parameters (local macro variables) as application parameters. The non-macro application will use global macro variables as application parameters. Basically, passing data set values into data driven applications is utilizing the SAS macro facility to store the data set values into these macro variables. This paper covers **three types of "driver" programs** with three techniques. All of the three techniques will generate SAS code to accomplish the task, although the generated code may be external/visible or internal/invisible. I will use examples to illustrate the techniques in the following sections.

## II. USING DATA _NULL_ AND PUT STATEMENTS

The first way to pass DATA set values to data driven applications is to automatically generate SAS code with DATA _NULL_ and PUT statements. An external SAS code file will be created and called explicitly. Let's say the sample data set has the following records, and the applications are defined in examples shown in section I above.

```
data facrep;
   provfs='111111';output;
   provfs='222222';output;
   provfs='333333';output;
run;
```

In the examples shown on next page, the DATA _NULL_ step creates an external program file which contains three statements invoking the macro application, one for each of the three values of variable PROVFS in the FACREP data set. After the DATA _NULL_ step finishes, the temporary program is executed with a *%include* statement in the same session. You can also use either the FILENAME statement or the %LET statement to store the external program file name. This technique is very easy to understand and works for all types of applications, macro or non-macro. But it requires setting up temporary external files. The programmer may need to clean them out afterwards.

## III. USING PROC SQL

The PROC SQL can be used to store the generated SAS statements into a macro variable instead of an external program file. The macro variable will be resolved as SAS statements and these statements are executed. For the examples shown on next page, you can use the `%put %SUPERQ(run)`statement to see the unresolved macro variable values. Please note that there is no "&" before the macro variable name when it is passed to %SUPER function.

*Pitfall: If you try to see the content of the macro variable &run without %SUPERQ, the resolved macro statements will be executed.*

**EXAMPLE II.1 (example1_driver_1.sas):**

```
data _null_;
set facrep;
file "&codepath\run.sas";
put '%wo_inc(' provfs ');';
run;
%inc “&codepath\run.sas”;
```

**EXAMPLE II.2 (example2_driver_1.sas):**

```
* use %let = ;
%let temp="&codepath\run.sas";
data _null_;
set facrep;
file &temp;
put '%w_inc(' provfs ');';
run;
%inc &temp;
```

**EXAMPLE II.3 (example3_driver_1.sas):**

```
 * use filename ;
filename temp "&codepath\run.sas";
data _null_;
set dates;
file temp;
put '%let dat=' date ';';
put '%let dsn=reptdata;’;
put '%let var=var1; ’;
put '%inc "&codepath\example3.sas";';
run;
%inc temp;
```

**DATA _NULL_ EXAMPLES**

**EXAMPLE III.1 (example1_driver_2.sas):**

```
proc sql noprint;
select '%wo_inc('||provfs||');'
  into: run separated by ' '
from facrep;
quit;

&run;
```

**EXAMPLE III.2 (example2_driver_2.sas):**

```
proc sql noprint;
select '%w_inc('||provfs||');'
   into: run separated by ' '
from facrep;
quit;

&run;
```

**EXAMPLE III.3 (example3_driver_2.sas):**

```
proc sql noprint;
select '%let dat='||date||
       '; %let dsn=reptdata;
       %let var=var1;
       %inc "&codepath\example3.sas";'
  into: run separated by ' '
from dates;
quit;

&run;
```

**PROC SQL EXAMPLES**

## IV. USING CALL EXECUTE ROUTINE

The CALL EXECUTE routine is another choice for passing data set values into data driven applications. It’s powerful and may be hard to understand. Here is the example from the official SAS documentation.

```
data dates;
   input date $;
datalines;
10nov97
11nov97
12nov97
;
data reptdata;
   input date $ var1 var2;
datalines;
10nov97 25 10
10nov97 50 11
11nov97 23 10
……
;
%macro rept(dat,a,dsn);
   proc chart data=&dsn;
       title "Chart for &dat";
       where(date="&dat");
       vbar &a;
   run;
%mend rept;

data _null_;
   set dates;
   call
execute('%rept('||date||','||'var1,reptdata)');
run;
```

“In this example, CALL EXECUTE passes the value of the DATE variable in the DATES data set to macro REPT for its DAT parameter, the value of the VAR1 variable in the REPTDATA data set for its A parameter, and REPTDATA as the value of its DSN parameter. After the DATA _NULL_ step finishes, three PROC GCHART statements are submitted, one for each of the three dates in the DATES data set.” (Excerpted from SAS documentation “*Macro Language Dictionary*”.)

Let’s make some changes to the macro REPT. We place the PROC statements into a subprogram c:\temp\chart_inc.sas and we add subprogram c:\temp\print_inc.sas. Organizing programs in this way is a very good practice when you have a large and complicated application. Modularity will make the application easier to maintain.

**Example 4 (macro application with %inc)**

```
* &codepath\example4_print_inc.sas;
 proc print data=&dsn;
 title "Print for &dat";
       where(date="&dat");
       var &var;
run;
```

```
* &codepath\example4_chart_inc.sas;
   proc chart data=&dsn;
       title "Chart for &dat";
       where(date="&dat");
       vbar &var;
   run;
```

```
* example4m.sas;
%macro rept(dat,var,dsn);
   %inc "&codepath\example4_print_inc.sas";
   %inc "&codepath\example4_chart_inc.sas";
%mend rept;
```

However, the following code with the CALL EXECUTE statement will fail. Why?

```
* example4_driver.sas - it will fail.;
data _null_;
   set dates;
   call execute('%rept('||date||','||'var1,reptdata)');
run;
```

***Pitfall**: %INCLUDE statement is not part of the SAS macro facility. So it's treated as plain text by the macro processor.*

When macro %rept is compiled, all the macro variables within these subprograms are not substituted with the values you expect. When the CALL EXECUTE routine executes the %rept with values passed from the DATES data set, the following SAS statements are generated and queued in the input stack waiting for compiling and execution. At the time these statements are executed the macro variables &dat, &var and &dsn will not exist.

```
proc print data=&dsn;
 title "Print for &dat";
       where(date="&dat");
       var &var;
run;
```

```
proc chart data=&dsn;
       title "Chart for &dat";
       where(date="&dat");
       vbar &var;
run;
```

There are two ways to work around this issue.

**SOLUTION 1:** The first one is to turn these regular subprograms into macros and invoke these submacros inside the main macro. Here are the revised programs.

**Example 4 (macro application with %inc macro)**

```
* &codepath\example4_print_macr.sas;
 %macro print_macr(dat,var,dsn);
 proc print data=&dsn;
 title "Print for &dat";
       where(date="&dat");
       var &var;
 run;
 %mend;
```

```
* &codepath\example4_chart_macr.sas;
   %macro chart_macr(dat,var,dsn);
   proc chart data=&dsn;
       title "Chart for &dat";
       where(date="&dat");
       vbar &var;
   run;
   %mend;
```

Once the subprograms are revised as macros, you can revise the application's main program as follows:

```
* &codepath\example4mm.sas ;
%inc "&codepath\temp\example4_print_macr.sas";
%inc "&codepath\temp\example4_chart_macr.sas";
%macro rept(dat,var,dsn);
   %print_macr(&dat,&var,&dsn);
   %chart_macr(&dat,&var,&dsn);
%mend rept;
```

The driver program will remain the same (example4_driver.sas) and it will work. Why? It will work because all the statements in those subprograms are macro statements which are visible to the macro facility during macro compiling phase. In the same way, the CALL EXECUTE routine can be applied to the application shown in example 1 in section I.

*Note: Beside %INCLUDE, SAS language functions SYMGET,SYMGETN, RESOLVE inside macro definitions will be viewed as plain text during the compiling step. Using CALL EXECUTE with macros containing any of these functions will produce unexpected results.*

**SOLUTION 2:** Another way goes in the reverse direction. We turn the application's main program into a regular SAS program as follows.

**Example 5 (non-macro application)**

```
* &codepath\example5.sas;
   %inc "&codepath\example4_print_inc.sas";
   %inc "&codepath\example4_chart_inc.sas";
   *… other statements…;
```

Then you might come up with the following lines to pass the data sets values via macro variables into the revised non-macro version application. These lines will run. However, they will produce frustrating results.

```
* &codepath\example5_driver_error.sas;
data _null_;
   set dates;
   call execute('%let dat='||date||';
                 %let var= var1;
                 %let dsn= reptdata;
                 %inc "&codepath\example5.sas";'
                );
run;
```

*Pitfall: the %LET statements will be executed immediately and the %INCLUDE statements will be executed after the hosting data step is finished. Therefore, the application will be repeatedly executed with the same set of values which is the last observation from the input data set.*

Why does this happen? Keep in mind that the CALL EXECUTE routine is actually a component of the **DATA step interfaces with the SAS macro facility** which "consist of eight tools that enable a program to interact with the macro facility **during** DATA step execution. … If CALL EXECUTE produces macro language elements, those elements execute immediately. If CALL EXECUTE produces SAS language statements, or if the macro language elements generate SAS language statements, those statements execute after the end of the DATA step's execution." (Excerpted from SAS documentation)

What should we do to prevent the %LET statement from being executed immediately? The answer is to use **macro quoting function,** %NRSTR, to mask the %LET statements so that the macro processor will treat %LET as plain text during compiling step. (Please refer to SAS documentation for the detailed information on macro quoting.) The correct driver program is as follows:

```
* &codepath\example5_driver.sas (using %nrstr);
data _null_;
   set dates;
   call execute('%nrstr(%let dat='||date||';
                 %let var=var1;
                 %let dsn=reptdata; ) ;
                 %inc "&codepath\example5.sas";'
               );
run;
```

## CONCLUSION

To pass data set values into application parameters, all of the three methods will generate SAS codes. The DATA _NULL_ and PUT statements method will store the generated code into temporary external files. The PROC SQL method will store the generated code into macro variables, while the CALL EXECUTE method will store them in SAS input stack. These methods can be applied to both types of designs. However, the third one may not be suitable to mixed style design. The design of your application will have a great impact on which technique to use. Developers' comprehension of the SAS macro will be the key factor to help them choose the appropriate application design and proper methods to pass data set values into application parameters.

## REFERENCES

SAS Institute Inc. 2004. “SAS(R) 9.1 Macro Language: Reference”, *SAS OnlineDoc® 9.1.3***.** Cary, NC: SAS Institute Inc.

H. Ian Whitlock, “CALL EXECUTE: How and Why”.
(http://www2.sas.com/proceedings/sugi22/CODERS/PAPER70.PDF)

## ACKNOWLEDEMENT

Many thanks go to Sandy Callard, Phil Tedeschi, Richard Eikstadt, Randy Webb, Cora Brunton for their comments and support during the writing of this paper. I would like to thank my great team at UM-KECC and everyone I have been learn from or worked with in my SAS adventure.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author may be contacted at:

Lingqun Liu
University of Michigan Kidney Epidemiology and Cost Center
315 W. Huron St. #240
Ann Arbor, MI 48103
Phone (734) 998-6609 Fax (734) 998-6620
**Email: lqliu@umich.edu; lingqun@gmail.com**