PhUSE 2017

Paper CT05

Comparing 6 Techniques To Do Data Driven Programming

Joris Derks, OCS Life Sciences, 's-Hertogenbosch, the Netherlands

ABSTRACT

Data driven programming is essential for the development of strong generic and reusable code. In this paper six techniques to execute SAS code based on values from a dataset – data driven programming – are compared. The following techniques will be compared: fetchobs, proc sql with an into clause, call symputx, call execute, proc fcmp in combination with run_macro, and dosubl. Based on the situation one method might be more appropriate than another, for example when a return value is needed. Aspects of interest are: complexity, maintainability, features, flexibility, and availability. These aspects will be presented in an overview to allow for easy review. Syntax and examples will be provided for each technique.

INTRODUCTION

Data driven programming is a technique where the data controls the flow of the program. The program contains pieces of code that are triggered to be run when the data meets the criteria. Main advantages of working with data driven programming techniques are ease of program maintenance and a more modular development approach. The flow of the program becomes dynamic for every data situation.

These data driven techniques can make use of SAS® data steps, SQL procedures, or SAS macro coding. The main concept is to get values from a dataset and parametrize a macro call using these values. Simply writing a macro call directly in a data step is not supported by SAS, as without using these techniques, SAS doesn't support running multiple data steps or procedures in parallel. All techniques discussed in this paper should be able to call a macro based on values in a dataset.

One thing that sets these techniques in this paper apart is the moment at which the macro will be executed. The fetchobs technique, the call symputx, and proc sql with an into clause technique work completely outside of the step where the dataset values are obtained. The call execute technique is a bit of a mix, as macro code is executed within the data step and code from within the macro is executed when the data step has completed. These techniques require an extra step in the called macro to return the values to the dataset. Both the proc fcmp in combination with run_macro and the dosubl technique work within the step where the dataset values are obtained and support return values, so the result is directly available.

The main focus of this paper is to show six techniques that can be used to let the data control the flow of the program. All of these techniques have their own advantages and disadvantages. The final decision to choose for a particular technique is programmers preference. Some of these techniques might be more appropriate in a certain situation, however most of them will be usable in any situation. This paper scores these techniques on complexity, maintainability, features, flexibility, and availability and will show examples on how to use these techniques.

FETCHOBS

The fetchobs technique uses the fetchobs() function in combination with %syscall to put variable values from a SAS dataset into macro variables. When called, the function `fetchobs(<dsid>, <rownum>)` creates macro variables for every variable in dataset <dsid> with values from the observation number provided in <rownum>.Steps to use the fetchobs technique are to open the dataset for read access, the dataset ID and the total number of observations have to be obtained, use the syscall macro function to link the values from the dataset into macro variables, and then use fetchobs to obtain the values from the dataset into macro variables. The values in the SAS dataset are now available in macro variables and can be further used in macro coding to perform additional processing.

Advantages

Advantages of this technique are that the code to obtain the values from the dataset into macro parameters can be re-used and this code is not really complex.

Disadvantages

Disadvantage of this technique is that the fetchobs function puts all variables in the dataset into macro variables, meaning if only one is needed, many others are created. The flexibility of using this technique is limited as only macro coding is available to use, if data needs to be transformed in a way or has to be filtered, it has to be done before using the technique. A programmer is obliged to encapsulate this code into a macro and extra statements are needed to obtain the data, for example to get the dataset ID and the number of observations, a syscall, the fetchobs calls, and the %do-loop. A programmer should be careful when using this technique, as previous defined macro variable might then be overwritten by a variable value from the dataset. To maintain this code is not so easy as debugging is more difficult as a counter variable is needed to loop over the values and all coding is done in macro coding. If values need to be returned then an extra data step is needed in the called macro.

When to use

The fetchobs technique should mainly be used when no additional pre-processing of data is required, less flexibility is needed, and macro coding has preference.

Example

The following example shows how the fetchobs technique can be used. First the dataset has to be opened for read access, then the number of observations has to be returned, a link has to be created to the dataset, and a loop over all records in the dataset has to be used, values are put into macro variables and then a macro is called using these values. The macro DISPLAY_VALUES is used as an example, so no definition is provided. What should be observed is that the parameter values to call this macro (name, sex, weight) originate from the dataset.

```
/* Macro  fetchobs will obtain all observations in dataset sashelp.class and */
/*   print the values of variable name, sex, and weight.                    */
%macro fetchobs;
  /* Get the dataset ID */
  %let dsid = %sysfunc(open(sashelp.class));
  /* Get the number of observations */
  %let nobs = %sysfunc(attrn(&dsid,NOBS));
  /* Create a link between the dataset values and macro parameters. */
  %syscall set(dsid);

  %do i = 1 %to &nobs;
    %let rc = %sysfunc(fetchobs(&dsid,&i));
    /* Call macro with the values from the dataset */
    %DISPLAY_VALUES(name=&name.,sex=&sex.,weight=&weight.);
  %end;

  /* Close the dataset */
  %let id=sysfunc(close(&dsid));
%mend fetchobs;
%fetchobs;
```

Example 1: Using the fetchobs technique to get dataset values into macro variables.

PROC SQL WITH INTO CLAUSE

Structured Query Language (SQL) is a widely used language to manage data. With the proc sql procedure, SAS is able to execute SQL code. Using an into clause in the proc sql procedure, SAS puts values from a dataset into macro variables. There are two ways to utilize this function: a) the values can be put into macro variables and then a macro is needed to loop over all created macro variables, or b) macro statements can be generated based on the values in this dataset and these statements can be executed by calling the generated macro variable. The latter has an advantage as no extra macro with %do loop is needed, however the maximum length of macro variable might be reached, depending on the number of records in the dataset.

Advantages

This is a relatively simple programming technique. Executing the macro code is straightforward, and you can make use of SQL advantages (joins, order by, group by). The dataset that is used as input can be altered in the macro. Data set values can be altered before putting them into a macro parameter.

Disadvantage

Disadvantages of this technique are its flexibility, relating to the SQL language. If all values are put into separate macro variables then also first the total number of observations should be obtained. Another disadvantage is, is that there is no support for return values. If used when looping over values extra counter variable and consecutive ampersands might be needed which aid in complexity. Debugging is also less flexible as in one sql procedure it is not possible to create a variable containing the macro command ánd putting this command into a macro variable, as an into clause cannot be used in a create table statement. A step needs to be added in the called macro to return values to the source dataset.

When to use

This is one of my favorite methods, as one can quickly generate a macro statement from a dataset, using regular-used coding. I do not use it when values need to be returned to the input dataset or specific SAS coding is required to generate the statements. If a programmer is more a fan of using SAS data step coding, the SQL technique might not be a good choice. The macro DISPLAY_VALUES is used as an example, so no definition is provided.

```
/* The proc sql with into clause to call and execute the above macro. */
/* Command is a macro variable which will be filled by the proc sql procedure. */
%let command = ;
proc sql noprint;
  SELECT '%DISPLAY_VALUES('||CATX(',',sex,count(name))||')'
  INTO :command separated by '; '
  FROM sashelp.class
  WHERE age > 13
  GROUP BY sex
  ;
quit;

/* Display the actual generated macro calls. This is useful because: */
/* - What will be executed is displayed.                             */
/* - Programmer can copy any of the statements to perform debugging. */
/* SUPERQ is needed as otherwise the statement is directly executed. */
%put %superq(command);

/* EXECUTE the macros. */
&command
```

Example 2: Use the PROC SQL technique to execute a macro.

CALL SYMPUTX

Call symputx is a SAS routine to put dataset values into macro variables from within a data step. As an example, Call `symputx(name,age)` will create a macro variable for every name value in the dataset with the value of age. Both the name and value can be literals, but also any value in the dataset could be used. This technique has much overlap with the proc sql with an into clause technique. Also here this technique might be used using two methods: put all values in the variables of the input dataset into macro variables and a loop over them or generate macro calls in one macro variable and call the generated macro variable.

Advantages

SAS data step functionality available to use which makes it flexible, for example first and last by group processing, if-then-else logic, and there is no need to obtain the total number of observations before the data step. It is easier to debug as statements can be put in the log. Also a macro statement per record can be added as character variable in the data step, to save the command for that record allowing easy review of generated macro calls. Not only the macro variable value is dynamic, but also the macro variable name can be defined dynamically.

Disadvantages

Extra statements are needed, like retain and a (large) variable when creating a command variable. When using this technique to loop over values a counter variable and consecutive ampersands are needed. A step needs to be added in the called macro to return values to the dataset.

When to use

Call symputx can be powerful in combination with by group processing or using the observation number to add another call to macro. For example for the first or last record. It is also very useful when the variables in the dataset require if-then-else logic, compared to proc sql technique. If different macros need to be called based on a parameter value this is easier than using a case statement in proc sql.

Example

The following example displays how the call symputx technique can be used to generate macro calls. Interesting points are the possibility to execute a macro at the beginning (%initialize), if the first record of a group is processed (%group_macro), based on a value in the dataset call a different macro (%run_for_males, %run_for_females), one and at the end (%last). The macro DISPLAY_VALUES is used as an example, so no definition is provided.

```
/* Create three types of macro variables: name_<observation number>, age_<name>,
and obs. */
data _NULL_;
  set sashelp.class end = last;

  call symputx('name_'||strip(put(_N_,best.)),name);
  call symputx('age_'||strip(name),age);

  if last then call symputx('obs',_N_);

run;

/* Loop over all observations, obtain the values from the dynamic parameters and
call the macro DISPLAY_VALUES. */
%macro LOOP_AGES;
  %do i = 1 %to &obs;
    %let name = &&&name_&i;
    %let age  = &&&&&age_&name.;
    %DISPLAY_VALUES(name = &name, age = &age);
  %end;
%mend LOOP_AGES;
%LOOP_AGES;
```

Example 3: The call symputx technique to obtain the values and then loop over them.

CALL EXECUTE

Call execute is an advanced SAS function that is able to execute any valid SAS code, which makes it a good candidate for data driven approaches. The call execute technique is very similar as the call symputx technique in combination with generating macro statements, however with call execute there is no need to create a macro variable and so removes the disadvantage of the maximum length of a macro variable. For example, `call execute('%mymacro(p1=one,p2=two);');` will run the macro %mymacro with parameters as provided, and will do so for every record in the dataset. When using this technique the values one and two, which are hardcoded into this example, will be replaced with values from the input dataset so every run of the macro will have different parameter values.
The timing of the function is different as call execute runs the macro code in the called macro within the data step. This is different compared to the three previous techniques. When a macro is called using a call execute function, SAS will resolve all values during the execution of the input data step. Any data step or proc steps are executed after the data step finished. This behavior can be changed by surrounding the command in the call execute statement with an %nrstr() function[1]. When this is applied the call execute function behaves equally as the call symputx technique: the macro is now actually called after the data step.

Advantages

For macro steps, the code that should be executed is executed during the data step where also the code is generated. This does not apply to data steps, these are still executed after the data step. SAS data step functionality is available to use. In one step a command variable can be created and stored, and the command can be executed.

Disadvantages

As indicated, the %nrstr fix (might be) needed for timing purposes which adds in complexity. Call execute does not provide a way to return a value from the macro to the input data set, without adding an extra step in the called macro. The call execute technique cannot be used in an SQL procedure.

When to use

Call execute is a powerful tool. It is possible to use any of the SAS data step advantages and is so flexible in its use. I like to use the call symputx technique over the call execute technique, as it does not require the %nrstr fix.

Example

In this example firstly a dataset is sorted to be able to use by group processing. Normal SAS data step functionality can be used to call a different macro depending on a value in the dataset using the call execute routine. The called macros (initialize, group_macro, run_for_males, run_for_females, last) are used as examples, so their definition is not provided.

```
/* Sort the input dataset based on the groups: sex */
proc sort data = sashelp.class
          out = work.class;
  by sex;
run;

data _null_;
  set work.class end = last;
  by sex;
  if _n_ = 1 then call execute('%initialize;');
  if first.sex then call execute('%group_macro('||strip(sex)||');');
  if sex = 'M' then call execute('%run_for_males('||strip(name)||');');
  if sex = 'F' then call execute('%run_for_females('||strip(name)||');');
  if last then call execute('%last;');
run;
```

Example 4: Example using the call execute technique.

PROC FCMP WITH RUN_MACRO

Proc fcmp is a SAS procedure to create custom functions. It can be seen as data step equivalent of a SAS macro. run_macro is a function only available for use in proc fcmp.
As example, `rc = run_macro('count_nr_subjects', study,nr_subjects)` will run the macro count_nr_subjects and two local macro variable will be created: study and nr_subjects. Study will be prefilled with a value from the dataset and nr_subject will be filled by the macro count_nr_subject. The logic of this function is that it will setup local macro variables, fills them with values from the custom function, then executes the macro, waits for it to complete, then obtains the values of the local macro variables created, and returns it to the input dataset[2]. This technique allows other data steps to be executed during the execution of another data step, an important difference compared to the previous techniques and it allows macro's to return a value directly into the dataset. Another difference is that this technique can be used both in a data step and in the sql procedure. As this technique creates local macro variables to be used in the macro, the macro is not able to have macro parameters, otherwise these will always be created empty.

Advantages

Two important features of this technique are that the referred code is actually executed during the data step or sql procedure, a value can be returned, and it is able to generate datasets, as previous techniques were lacking these features. The technique is flexible as it can be used both in a data step and in proc sql procedure.

Disadvantages

There is extra code needed to first define the function and then to execute a macro. Macro that is executed should not contain macro parameters, making the macro mainly usable only for this technique. The source dataset cannot be used in the macro. It is a complex technique and will require a learning curve to create custom functions. It is also more difficult to maintain. If there is an update to a parameter, the data step, macro, and custom function require modifications.

When to use

This is the first technique that is executed completely during a data step. If a programmer does not know the proc fcmp function already, the learning curve on how to use this is quite steep. However, the ability to return a value to the input data set is quite a useful feature, as for example status codes can be easily returned to the source dataset.

Example

In this example proc fcmp is used to create a custom function count_nr_subjects. This function sets the parameter values into (local) macro variables, create an empty macro variable nr_subjects which will be filled by the macro, it will call the macro and wait for it to end, and at the end return the value of nr_subjects. In the data step the function count_nr_subjects is called, the return value of this function goes into variable count_subjects. The macro count_nr_subjects is used as example, so no definition is provided.

```
/* In this fcmp procedure a custom function is create named 'count_nr_subjects',
which will set the value of the parameter study into a (local) macro variable
study, create empty macro variable nr_subjects, call the macro count_nr_subjects,
when the macro count_nr_subjects is finished the nr_subjects value is returned */
proc fcmp outlib = work.func.functions;
  function count_nr_subjects(study $);
    rc = run_macro('count_nr_subjects',study,nr_subjects);
    return(nr_subjects);
  endsub;
run;

/* Add this to the cmplib option, otherwise the function cannot be loaded. */
options cmplib=work.func;

/* Loop over all studies in work.overview and call the function count_nr_subjects
with study name as parameter. */
data work.overview_completed;
  length count_subjects 8;
  set work.overview;
  count_subjects = count_nr_subjects(study_name);
run;
```

Example 5: The proc fcmp to obtain variable values from a dataset and to return a value to a dataset.

DOSUBL

The dosubl function is a new function, introduced in the second maintenance release of SAS9.3[3]. Dosubl can be seen as an evolution of call execute. Compared to call execute, dosubl is actually able to execute data step code while another data step is being executed. In this way it has the advantages of both call execute and the proc fcmp technique. It only misses the return value from proc fcmp, however using symget the possibility is still there, but requires an extra statement. When the following statement is executed `dosubl('%count_nr_subjects('||strip(study_name)||');')` SAS will run the macro count_nr_subjects and the value from the variable study_name is used as macro parameter. If a return value is needed then this can be obtained using a symget function.

Advantages

This technique has many advantages: It does not require extra steps, no custom functions are needed, it can be used in a data step and in an sql procedure, and it supports return values. Macro parameters of the called macro can be filled, so the called macro does not require any modifications.

Disadvantages

Only available since SAS9.3M2.

When to use

Dosubl seems to be a very promising function. It has advantages of both call execute and proc fcmp, like running data steps during another dataset and the ability to return values to a dataset. If using SAS9.3M2 and up, dosubl might the best choice for data driven techniques.

Example

The following example is similar as to what was used for proc fcmp including run_macro. What should be noticed compared to the proc fcmp technique is that there is no need to create an extra function. An extra symget function is needed to obtain the return value. The macro count_nr_subjects is used as example, so no definition is provided.

```
%LET nr_subjects = ;
/* Loop over all studies in work.overview and call the macro count_nr_subjects with
study name as parameter using the DOSUBL function.
** Then use symget to get the return value.
** If there is negative value in the count_subjects variable then put an er-ror. */
data work.overview_completed(drop = rc);
  length count_subjects 8;
  set work.overview;
  rc = dosubl('%count_nr_subjects('||strip(study_name)||');');
  count_subjects = symgetn('nr_subjects');
  if count_subjects < 0 then put 'ER' 'ROR: something went wrong when counting
subjects in study:' study_name;
run;
```

Example 6: An example of using the dosubl technique.

CONCLUSION

Six Techniques To Do Data Driven Programming in SAS are discussed and example code is presented. All techniques have their own advantages and disadvantages. The choice of technique depends on the situation and programmer's preference. If using SAS9.3M2 or later, the dosubl technique is the most promising technique as it has no disadvantages.

Below is a table that can be used as reference to easily review what the characteristics are of all six techniques.

Complexity is higher when extra statements are needed or timing is of importance. Maintainability is better when no counter variable is required and there is no learning curve. Features are positive when the technique allows return values directly in the source dataset. A technique useable in multiple languages (SAS, SAS Macro, SQL) has a better flexibility. Availability is scored negative when the technique is introduced more recently.

Technique	Complexity	Maintainability	Features	Flexibility	Availability
FETCHOBS	-	-	-	-	+
PROC SQL with INTO	+	- / +	-	-	+
CALL SYMPUTX	+	- / +	-	-	+
CALL EXECUTE	-	+	-	-	+
PROC FCMP + RUN_MACRO	-	-	+	+	+
DOSUBL	+	+	+	+	- / +

+ : positive

- : negative

REFERENCES

1. Call Execute: Let Your Program Run Your Macro - Artur Usov, OCS Consulting BV, 's-Hertogenbosch, Netherlands
2. RUN_MACRO Run! With PROC FCMP and the RUN_MACRO Function from SAS® 9.2, Your SAS® Programs Are All Grown Up - Dylan Ellis, Mathematica Policy Research, Washington, DC
3. Submitting SAS® Code On The Side - Rick Langston, SAS Institute Inc., Cary NC

RECOMMENDED READING

Changing Data Set Variables into Macro Variables - William C. Murphy Howard M. Proskin and Associates, Inc., Rochester, NY

Executing a PROC from a DATA Step - Jason Secosky, SAS Institute Inc., Cary, NC

CONTACT INFORMATION

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

Author Name	Joris Derks
Company	OCS Life Sciences
Address	Ruwekampweg 2G
City / Postcode	‘s-Hertogenbosch / 5222 AT
Work Phone	+31 (0)73 523 6000
Email	sasquestions@ocs-consulting.com
Web	www.ocs-lifesciences.com