SESUG Paper 190-2019

Conditionally Executing Data Steps and Statements Based on the Presence of Variables in a SAS® Dataset

Charles D. Coleman, US Census Bureau

DISCLAIMER

Any views expressed are those of the author and not necessarily those of the Census Bureau. The Census Bureau has reviewed this product for unauthorized disclosure of confidential information. (Approval ID: CBDRB-FY19-ESMD-B00019.)

ABSTRACT

Sometimes a data step should only be executed if one or more variables are present in the input dataset. The same is true of statements within data steps. This paper provides several macros to detect the presence of variables and counts of the present variables. Examples show how these macros can be used to accomplish conditional execution for many scenarios. These macros have the advantage of only using macro statements, thus avoiding the costs of opening datasets.

INTRODUCTION

Conditional execution is familiar to all programmers. The classic method is the if/then/else construct, which is limited to the context of a program environment. Programs in Base SAS® and the SAS® Macro Language are two examples of these environments. Conditional execution using information external to a particular environment is generally not supported. For example, Base SAS® has no means for the programmer to determine what variables are in a dataset once it is loaded.

This paper uses information external to a data step to support conditionally running that data step and selected statements within it. The external information consists of input dataset variables. This paper presents macros to create macro variables that flag the existence of variables in a dataset, to count the number of such variables and a macro to do both. In addition, a required utility macro is presented. This utility macro creates a macro variable containing a list of all of the variables in the target data set. The other macros essentially sequentially test each of the test variables for its presence in the data set and perform actions based on the results.

After the macros are presented, examples are shown of their use. The last macro demonstrates the power of this approach: it loops through a large number of datasets to ascertain the existence of a variable in each. The examples are by no means exhaustive: the reader is encouraged to customize the use of the macros to his problem.

EXTRACTING DATA SET VARIABLES: MACRO %GETVARS

Macro %GETVARS queries DICTIONARY.COLUMNS to create macro variable &vlist containing all of the variables in datasest &lib..&dsn, as specified by its arguments. Library DICTIONARY consists of in-memory views that assemble run-time snapshots of the session properties when queried (SAS 2016, 149-155). This library can only be accessed using PROC SQL.

The code for getvars.sas is:

```
   %macro getvars(lib,dsn);
      /*  Based on
      https://communities.sas.com/t5/SAS-Tips-from-the-Community/SAS-Tip-
Easy-Way-to-Get-All-Variable-Names-From-a-Dataset/td-p/475815
      */
      %global vlist;
      proc sql noprint;
         select upcase(name) into :vlist separated by ' '
```

```
         from dictionary.columns
         where memname = upcase("&dsn") and libname = upcase("&lib");
      quit;
   %mend getvars;
```

Arguments &LIB and &DSN specify the library and dataset, respectively. The use of the colon in the select clause indicates that VLIST is a macro name rather than a dataset. The "separated by" qualification assures that each variable name is separated by a blank.

COUNTING SELECTED VARIABLES IN A DATASET: MACRO %COUNTDSVARS

Macro %COUNTDSVARS counts the number of variables. It puts the count of the variables listed in argument &var in dataset &lib.&dsn into global macro variable &ndsvars.

The code for countdsvars.sas is:

```
   %macro countDSVARS(LIB,DSN,VARS);
      /* Input variables:
         LIB:   Input library
         DSN:   Input dataset name
         VARS:  Variables whose presence is to be counted with blank
separations

         Output variable:
         &NDSVARS:  Number of &VARS present in &LIB.&DSN

         Programmed by:

         Chuck Coleman
         Economic Statistical Methods Division
         U.S. Census Bureau
         July 12, 2019
     */

      %global NDSVARS;       *  Export &NDSVARS;
      %local I J NVARS NVLIST;
      %getvars(&lib,&dsn);   *  Put list of variables in &lib..&dsn into
&VLIST;
      %let VARS = %upcase(&VARS);  *  Make &VARS uppercase like the &VLIST&J;
      %let NDSVARS = 0;       *  Initialize &NDSVARS to 0.  If no matches,
will remain 0.;
      %let NVARS = %sysfunc(countw(&VARS,%str( )));  *  Count elements of
&VARS;
      %let NVLIST = %sysfunc(countw(&VLIST,%str( )));  *  Count elements of
&VLIST;

      /*  Create macro variables VLIST&J to hold elements of &VLIST for ease
and efficiency  */

      %do J = 1 %to &NVLIST;
         %let VLIST&J = %scan(&VLIST,&J);
      %end;

      /*  Test for presence of &VARS in &lib..&dsn via &VLIST.  Increment
&NDSVARS when one is found  */

      %do I = 1 %to &NVARs;
         /*  Select test variable &I  */
```

```
         %let CURRENTVAR = %upcase(%scan(&VARS,&I));
         /*  Loop through &VLIST via the &&VLIST&J  */
         %do J = 1 %to &NVLIST;
            %if &CURRENTVAR = &&VLIST&J %then %let NDSVARS = %eval(&NDSVARS +
1);    *  If match, increment &NDSNVARS;
         %end;
      %end;

   %mend COUNTDSVARS;
```

Macro %COUNTDSVARS is straightforward. It creates global macro variable &DSNVARS to hold the number of variables in argument &VARS found. After a call to %getvars and some initializations, it creates macro variable &VLIST&J to hold the &J'th variable in &VLIST for each variable in &lib..&dsn. It then loops through the variables contained in &VARS and the variables &VLIST&J to find matches. The syntax &&VLIST&J forces the macro processor to return the value of &VLIST&J. When a match is found, &NDSVARS is incremented. If no variable is found, &NDSVARS remains 0.

FLAGGING SELECTED VARIABLES' EXISTENCE: MACRO %EXISTDSVARS

Macro %EXISTDSVARS creates a variable &exist&VAR for each variable &VAR in argument &VARS. These variables take the value 1 if the corresponding variable is present, 0 otherwise.

The code for existdsvars.sas is:

```
   %macro existDSVARS(LIB,DSN,VARS);
      /* Input variables:
         LIB:     Input library
         DSN:     Input dataset name
         VARS:    Variables whose presence is to be counted

         Output variables:
         &NDSVARS:  Number of &VARS present in &DSN
         &exist&VAR:  Existence flags for each &VAR in &VARS: 1 if if &VAR
exists on &lib.&dsn.  0 otherwise.

         Programmed by:

         Chuck Coleman
         Economic Statistical Methods Division
         U.S. Census Bureau
         July 12, 2019
    */

      %local I J NVARS NVLIST;
      %getvars(&lib,&dsn);  *  Put list of variables in &lib..&dsn into
&VLIST;
      %let VARS = %upcase(&VARS);   *  Make &VARS uppercase like the
&VLIST&J;
      %let NVARS = %sysfunc(countw(&VARS,%str( )));   *  Count elements of
&VARS;
      %let NVLIST = %sysfunc(countw(&VLIST,%str( ))); *  Count elements of
&VLIST;

      /*  Create macro variables VLIST&J to hold elements of &VLIST for ease
and efficiency  */

      %do J = 1 %to &NVLIST;
            %let VLIST&J = %upcase(%scan(&VLIST,&J));
```

```
      %end;

      /*  Test for presence of &VARS in &lib..&dsn via &VLIST.  Increment
&NDSVARS when one is found  */

      %do I = 1 %to &NVARs;
            /*  Select test variable &I  */
            %let CURRENTVAR = %scan(&VARS,&I);
            /*  Initialize and make global &exist&CURRENTVAR    */
            %global exist&CURRENTVAR;
            %let exist&CURRENTVAR = 0;
            /*  Loop through &VLIST via the &&VLIST&J  */
            %do J = 1 %to &NVLIST;
                  %if &CURRENTVAR = &&VLIST&J %then %let exist&CURRENTVAR =
1;  *  Raise existence flag if &CURRENTVAR is found in &VLIST;
            %end;
      %end;

   %mend existDSVARS;
```

Macro %EXISTDSVARS is nearly identical to macro %COUNTDSVARS. The differences are that %EXISTVARS does not create &NDSVARS, instead creating one macro variable (&exist&CURRENTVAR) for each variable in &VARS in &lib..&dsn.

COUNTING AND FLAGGING VARIABLES IN A DATASET: MACRO %EXISTANDCOUNTDSVARS

Macro %EXISTANDCOUNTDSVARS combines the functions of macros %EXISTDSVARS and %COUNTDSVARS. Its code is:

```
   %macro existandcountDSVARS (LIB,DSN,VARS);
      /* Input variables:
         LIB:     Input library
         DSN:     Input dataset name
         VARS:    Variables whose presence is to be counted

         Output variables:
         &NDSVARS:  Number of &VARS present in &DSN
         &exist&VAR:  Existence flags for each &VAR in &VARS: 1 if if &VAR
exists on &lib.&dsn.  0 otherwise.

         Programmed by:

         Chuck Coleman
         Economic Statistical Methods Division
         U.S. Census Bureau
         July 12, 2019
      */

      %global NDSVARS;    *  Export &NDSVARS;
      %local I J NVARS NVLIST;
      %getvars(&lib,&dsn);*  Put list of variables in &lib..&dsn into &VLIST;
      %let VARS = %upcase(&VARS);   *  Make &VARS uppercase like the
&VLIST&J;
      %let NDSVARS = 0;   *  Initialize &NDSVARS to 0.  If no matches, will
remain 0.;
```

```
      %let NVARS = %sysfunc(countw(&VARS,%str( )));    *  Count elements of
&VARS;
      %let NVLIST = %sysfunc(countw(&VLIST,%str( ))); *  Count elements of
&VLIST;

      /*  Create macro variables VLIST&J to hold elements of &VLIST for ease
and efficiency  */

      %do J = 1 %to &NVLIST;
         %let VLIST&J = %upcase(%scan(&VLIST,&J));
      %end;

      /*  Test for presence of &VARS in &lib.&dsn via &VLIST.  Increment
&NDSVARS when one is found  */

      %do I = 1 %to &NVARs;
         /*  Select tested variable &I  */
         %let CURRENTVAR = %scan(&VARS,&I);
         /*  Initialize and make global &exist&CURRENTVAR  */
         %global exist&CURRENTVAR;
         %let exist&CURRENTVAR = 0;
         /*  Loop through &VLIST via the &&VLIST&J  */
         %do J = 1 %to &NVLIST;
            %if &CURRENTVAR = &&VLIST&J %then %do;     *  Match found;
               %let NDSVARS = %eval(&NDSVARS + 1);     *  Increment &NDSNVARS;
               %let exist&CURRENTVAR = 1;              *  Raise existence
flag;
            %end;
         %end;
      %end;

   %mend existandcountDSVARS;
```

EXAMPLES

EXAMPLE 1: CONDITIONALLY RUNNING CODE BASED ON THE EXISTENCE OF ONE VARIABLE IN A DATASET

This is really two examples: Example 1a uses macro %EXISTDSVARS to look for variable X in dataset EXAMPLE1 and creates variable &existX. If X is found, it prints the contents of EXAMPLE1, which is simply the one observation variable X containing the string “Hello World!”. Example 1b uses macro %COUNTDSVARS to count the number of variables found, in this case 1, to do the same thing.

Example 1a

The code for Example 1a is:

```
   data work.EXAMPLE1;
      length x $12;
      infile datalines dlm=',';
      input x $;
      datalines;
      Hello World!
   ;

   %existDSVARS(work,example1,x);
```

```
%macro EXAMPLE1A;

   %if &existX = 1 %then %do;

      proc print data=example1;
      run;

   %end;

%mend EXAMPLE1A;

%EXAMPLE1A;
```

Example 1b

The code for Example 1b is:

```
data work.EXAMPLE1;
   length x $12;
   infile datalines dlm=',';
   input x $;
   datalines;
   Hello World!
;

%countDSVARS(work,example1,x);

%macro EXAMPLE1B;

   %if &NDSVARS = 1 %then %do;

      proc print data=example1;
      run;

   %end;

%mend EXAMPLE1B;

%EXAMPLE1B;
```

An important thing to note in Example 1b is that it only works when one variable is selected. Thus, Example 1a is to be preferred in this case.

Output from Examples 1a and 1b

Both examples produce the expected output, shown in Output 1.

```
Obs  x

 1   Hello World!
```

Output 1. Output from Examples 1a and 1b

EXAMPLE 2: CONDITIONALLY EXECUTING CODE WITHIN A DATA STEP.

Example 2 counts test variables in a dataset. In this case, variables X and Y are tested. If at least one of them is found, data step EXAMPLE2_CONDITIONAL is executed. This data step contains one conditionally executed statement for both X and Y. In each case, it is to create a new variable containing the original variable, rounded.

The code for Example 2 is:

```
data work.EXAMPLE2;
   input x y;
   datalines;
   1.4 2.4
   10.1 6.9
   8.9 3.1
;

%existandcountDSVARS(work,example2,x y);

%macro EXAMPLE2;

   %if &NDSVARS >= 1 %then %do;

      data EXAMPLE2_CONDITIONAL;
         set EXAMPLE2;
         %if &existX %then
            xr = round(x,1);;
         %if &existY %then
            yr = round(y,1);;
      run;

      proc print data=EXAMPLE2_CONDITIONAL;
      run;

   %end;

%mend EXAMPLE2;

%EXAMPLE2;
```

Example 2 uses macro %EXISTANDCOUNTDSVARS. It uses macro variable &NDSVARS to conditionally execute the data step and PROC PRINT. Within the data step, it uses macro variables &existX and &existY to conditionally execute data step statements. A slightly more verbose alternative is to call %EXISTDSVARS and use &existX and &existY in the %if-statement condition. The double semicolon at the end of each %if statement is required to end both the data step statement and the %if statement.

Example 2 produces the output shown in Output 2.

Obs	x	y	xr	yr
1	1.4	2.4	1	2
2	10.1	6.9	10	7
3	8.9	3.1	9	3

Output 2. Output from Example 2

As expected, variables xr and yr contain the rounded values of variables x and y, respectively.

EXAMPLE 3: USING AN AUXILIARY VARIABLE TO CONDITIONALLY RUN CODE ON OTHER VARIABLES

Example 3 shows how the presence or value of a third variable can control the execution of code on two other variables. It contains two identical %if loops that call %EXISTDSVARS to flag the existence of variables W, X and Z. Data step EXAMPLE3_CONDITIONAL contains an %if statement to compute variable ZI only if variable W is present. The only difference between the two %if loops is the input dataset EXAMPLE3: The first contains W and the second does not.

The code for Example 3 is:

```
data work.EXAMPLE3;
   input w x y;
   datalines;
   1 1 2
   1 10 5
   0 6 3
;

%macro EXAMPLE3;

   %existDSVARS(work,example3,w x y);

   %if &existX and &existY %then %do;

      data EXAMPLE3_CONDITIONAL;
         set EXAMPLE3;
            z = x/y;
            %if &existW %then
               zi = y/x;;
      run;

      proc print data=EXAMPLE3_CONDITIONAL;
         title 'Example 3 with Variable W Present';
      run;

   %end;

   data EXAMPLE3;
      set EXAMPLE3(drop = W);
   run;

   %existDSVARS(work,example3,w x y);

   %if &existX and &existY %then %do;

      data EXAMPLE3_CONDITIONAL;
         set EXAMPLE3;
         z = x/y;
         %if &existW %then
            zi = y/x;;
      run;

      proc print data=EXAMPLE3_CONDITIONAL;
         title 'Example 3 with Variable W Absent';
      run;
```

```
    %end;

    %mend EXAMPLE3;

%EXAMPLE3;
```

The macro call %EXISTDSVARS(work,example3,w x y) flags the existence of variables w, x and y in dataset EXAMPLE3. The subsequent %if statement only executes if variables X and Y are found. In Data step EXAMPLE3_CONDITIONAL, statement `z = x/y;` unconditionally executes. The next statement executes only if variable W is present, as indicated by the value of macro variable &existW. The %if statement is required to avoid creating variable ZI if W is absent. This statement clearly shows how data step statement execution can be made conditional on the existence of a dataset variable. Output 3 shows the output from this example. Note that variable ZI is only created when variable W exists.

Example 3 with Variable W Present

Obs	w	x	y	z	zi
1	1	1	2	0.5	2.0
2	1	10	5	2.0	0.5
3	0	6	3	2.0	0.5

Example 3 with Variable W Absent

Obs	x	y	z
1	1	2	0.5
2	10	5	2.0
3	6	3	2.0

Output 3. Output from Example 3

Note that variable ZI is not created when variable W is absent.

EXAMPLE 4: LOOPING THROUGH DATASETS TO DETERMINE THE CODE TO EXECUTE.

Example 4 shows the true power of these macros. One-time use as in the previous examples only makes sense if a program is to be run multiple times on a changing dataset. Example 4, in contrast, runs on multiple datasets sequentially. In this case, it looks for datasets in the built-in library MAPS to find those that contain variable POP.

Example 4's code is:

```
proc datasets nolist nowarn;
   delete EXAMPLE4_OUTPUT;
run;

/*  Put list of dataset names in library MAPS into variable &DSLIST  */
```

```
%global DSLIST;

proc sql noprint;
   select unique memname into :DSLIST separated by ' '
   from dictionary.columns
   where libname = 'MAPS';
quit;

%macro EXAMPLE4;

   %let NDSLIST = %sysfunc(countw(&DSLIST,%str( )));  *  Count elements of
&DSLIST;
   %do K = 1 %to &NDSLIST;       *  Loop through datasets in library MAPS;

      %let DSN = %scan(&DSLIST,&K);
      options nonotes;
      %existDSVARS(MAPS,&DSN,pop);     * Look for POP in MAPS..&dsn;
      options notes;
      %if &existPOP %then %do;    *  POP found;

         data TEMP;
            length DSN $32;
            DSN = "&DSN";
         run;

         proc append base=EXAMPLE4_OUTPUT data=TEMP force;
         run;

      %end;

   %end;

%mend EXAMPLE4;

%EXAMPLE4;

proc print data=EXAMPLE4_OUTPUT;
run;
```

Example 4 uses macro %EXISTDSVARS to determine if a dataset contains a variable named POP. If so, it appends the dataset name as variable DSN to dataset EXAMPLE4_OUTPUT. It quickly examined 378 datasets to find the two that contain POP, as shown in Output 4.

Obs	DSN
1	USCITY
2	WAKEABG

Output 4. Output from Example 4

CONCLUSION

SAS® code can be conditionally executed based on the presence of dataset variables. This paper has presented 3 macros and an enabling utility macro to accomplish this. These macros can enable a

tremendous amount of conditional processing. The examples, while simple, illustrate powerful ways to conditionally execute code. Example 4, in particular, shows how these macros are most powerful when applied to multiple datasets.

REFERENCES

SAS Institute Inc. 2016. SAS® 9.4 SQL Procedure User's Guide, Fourth Edition. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to thank Ahmed Al-Attar and Jesus Lopez for programming assistance, and Suzanne Dorinski and Bonnie Kegan for review.

RECOMMENDED READING

- *SAS® DATA Step Statements: Reference*
- *SAS® Macro Language Reference*

CODE REPOSITORY

All macros will uploaded to https://sourceforge.net/projects/chuckcolemansas/files/SESUG%202019/.

CONTACT INFORMATION

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

Charles D. Coleman
Construction Survey Statistical Methods Branch
Economic Statistical Methods Division
U.S. Census Bureau
CENHQ 5H482C
Washington, DC 20233
301-763-6068
charles.d.coleman@census.gov