MWSUG 2016 - Paper RF07

An Application of CALL VNEXT

John Henry King, Ouachita Clinical Data Services Inc., Caddo Gap, AR

ABSTRACT

VNEXT is a SAS® call routine that allows the examination of the program data vector (PDV), each call to the routine alters the values of its arguments to return information about each variable in the PDV. The data returned is variable name and optionally variable type and length. This Rapid Fire talk presents a short program that uses VNEXT to write a CSV file that includes the name row followed by the data rows all in one very short DATA step.

INTRODUCTION

A SAS® CALL returned is similar to a function but the way the output is returned it difference. A function returns a value that may be assigned to a variable or used directly in an expression while a CALL routine alters the values of its variable name arguments. For example CALL SORTN accepts a list of variables as arguments and alters the values of the variables such that the values are sorted in ascending order replacing the values of the argument variables. CALL SCAN has two arguments that are altered POSITION and LENGTH, which give the starting position and length of the word, contrasted with the SCAN function which would return the word. CALL VNEXT allows us to examine the PDV (program data vector) it has two optional arguments TYPE and LENGTH and one required argument NAME. When the value of the variable specified for the name argument is blank or missing CALL VNEXT returns the name of the left most variable from the PDV. Subsequent calls return the next variable until the last variable in the PDV is returned to the name argument, and by calling VNEXT multiple times we can traverse the entire PDV.

CSV WRITER VERSION 1

This example of VNEXT is an application that creates a CSV file; it is a simple data _NULL_ that writes the variable names as the first row of the output file followed by the data rows.

```
data _null_;
   file log ls=256 dsd; ❶
   set sashelp.heart; ❷
   if _n_ eq 1 then link names; ❸
   put (_all_)(:); ❹
   return; ❺
names: ❻
   length _name_ $32; ❼
   do while(1); ❽
      call vnext(_name_); ❾
      if _name_ eq '_name_' then leave; ❿
      put _name_ @; ⓫
      end;
   put; ⓬
   return; ⓭
   run;
```

❶ Define the output file in this example we will just use LOG but it would normally be a file 'your-data-file.csv'

❷ Read the SAS data set of interest.

❸ On the first iteration of the data step _N_=1 link to the statement label NAMES❻. Writing the name row is done using LINK instead of the more common "IF THEN DO" to isolate the declaration of the variable that will be altered by CALL VNEXT.

❹ put statement using the "special name variable list" _ALL_ to write the values of the variables from the input data set SASHELP.HEART❷. The parenthesis surrounding the variable list _ALL_ are needed to tell SAS that we want _ALL_ to mean a variable list NOT the PUT statement directive _ALL_. Since we have a parenthesized variable list we also need a parenthesized format list to satisfy the requirements of the PUT statement syntax. The format modifier colon is all that is needed as SAS will use a default format or the format associated with the variable to write each value.

❺ RETURN to start another iteration of the data step loop. Without this statement here the program would execute the LINK routine for each iteration of the data step.

❻ Statement label NAMES, this is the statement linked to by the LINK statement.

❼ Define a new variable to accept the output from VNEXT, it important that this variable be defined here in the LINK routine so that the variable list _ALL_ does not include it.

❽ Start an infinite loop; make sure there is a way to break out ❿ when using this construct.

❾ VNEXT is called to obtain the a variable name from the PDV. The initial value of _NAME_ is blank so the routine returns with the leftmost variable in the PDV.

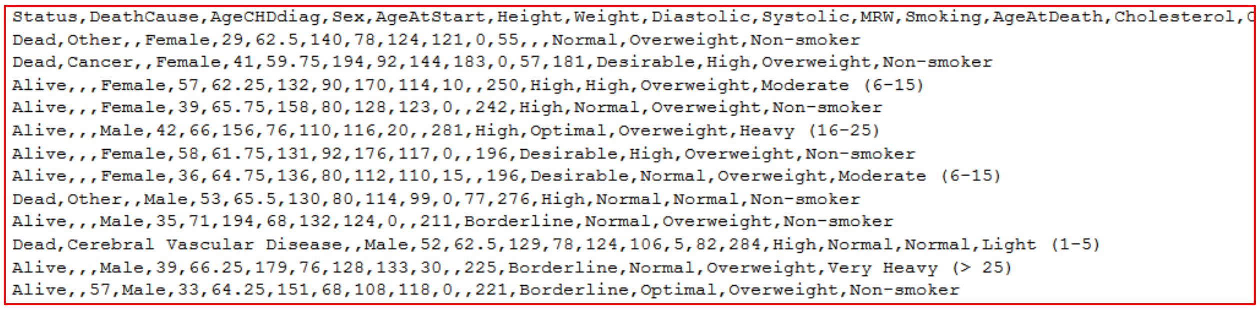
❿ To stop the infinite loop and stop writing the name row test the value of _NAME_ equal "_NAME_" it marks the end of the variables from SASHELP.HEART.

⓫ Write the value of _NAME_, a variable name from SASHELP.HEART, to the first row of the output CSV. Note the use of trailing @ so each execution of the put statement writes each name on to the same line.

⓬ In order to release the tightly held data line execute a PUT statement that does not use the trailing @.

⓭ Return to the line following the link statement.

The resulting output of this program is shown in Figure 1. CSV file of SASHELP.HEART. To make this program into a macro you would include a FILE= parameter to specify the output file, and a DATA= parameter to specify the input data set, no other code would need to be modified as we use other SAS features to refer to the variables.



```
Status,DeathCause,AgeCHDdiag,Sex,AgeAtStart,Height,Weight,Diastolic,Systolic,MRW,Smoking,AgeAtDeath,Cholesterol,
Dead,Other,,Female,29,62.5,140,78,124,121,0,55,,,Normal,Overweight,Non-smoker
Dead,Cancer,,Female,41,59.75,194,92,144,183,0,57,181,Desirable,High,Overweight,Non-smoker
Alive,,,Female,57,62.25,132,90,170,114,10,,250,High,High,Overweight,Moderate (6-15)
Alive,,,Female,39,65.75,158,80,128,123,0,,242,High,Normal,Overweight,Non-smoker
Alive,,,Male,42,66,156,76,110,116,20,,281,High,Optimal,Overweight,Heavy (16-25)
Alive,,,Female,58,61.75,131,92,176,117,0,,196,Desirable,High,Overweight,Non-smoker
Alive,,,Female,36,64.75,136,80,112,110,15,,196,Desirable,Normal,Overweight,Moderate (6-15)
Dead,Other,,Male,53,65.5,130,80,114,99,0,77,276,High,Normal,Normal,Non-smoker
Alive,,,Male,35,71,194,68,132,124,0,,211,Borderline,Normal,Overweight,Non-smoker
Dead,Cerebral Vascular Disease,,Male,52,62.5,129,78,124,106,5,82,284,High,Normal,Normal,Light (1-5)
Alive,,,Male,39,66.25,179,76,128,133,30,,225,Borderline,Normal,Overweight,Very Heavy (> 25)
Alive,,57,Male,33,64.25,151,68,108,118,0,,221,Borderline,Optimal,Overweight,Non-smoker
```

Figure 1. CSV file of SASHELP.HEART

CSV WRITER VERSION 2

The following is an extension of the above program that will output a separate file for each level of a BY variable.

```
proc sort data=sashelp.class out=class;❶
   by age;
   run;
data _null_;
   set class; by age;❷
   length filevar $256;❸
   filevar=catx('\','\path',catx('.',age,'csv'));❹
   file dummy filevar=filevar dsd; ❺
   if first.age then link names;❻
   put (_all_)(:);❼
   return;❽
 names:❾
   length _name_ $32;❿
   do while(1);⓫
      call vnext(_name_);⓬
      if _name_ eq: 'FIRST.' then leave;⓭
      put _name_ @;⓮
     end;
   put; ⓯
  return;⓰
  run;
```

❶ Sort the data to by the desired BY groups, AGE.

❷ Set the data BY AGE. The BY statement creates the FIRST.AGE variable used to detect the beginning of each age group.

❸ Define the variable FILEVAR for the use in FILE statement option.

❹ Create a file name by concatenating the path, value of age for the current by group, and the file type.

❺ The FILE statement uses DUMMY for the file reference name the actual file path coming from the FILEVAR option. Each time the value of FILEVAR changes SAS closes the current output file and opens a new one using the value in FILEVAR. The DSD option is included to produce comma delimited output.

❻ Link to the label NAMES to write the name row at the start of each BY group.

❼ PUT all user variables defined on the SET statement. We might expect the variable FILEVAR to be included in the (_ALL_) variable list but because it is the variable associated with the FILEVAR option it would not be written to an output data set and by extension is not in the (_ALL_) variable list even though it was defined with LENGTH before the use of (_ALL_) in the put statement.

❽ RETURN to start another iteration of the data step loop. This is necessary to prevent the data step from executing the LINK routine. Without the RETURN here the data step would continue into the section we have defined as the LINK routine.

❾ Statement label NAMES this label is the target, the statement the program will GOTO, of the link statement.

❿ Define a new variable to accept the output from VNEXT. This variable must be defined AFTER the use of (_ALL_) in the put statement to prevent its inclusion in that variable list.

⓫ Start an infinite loop.

⓬ VNEXT it called to obtain the variable names in the PDV.

⓭ As VNEXT traverses the PDV check each name to determine the “exit flag” for the infinite loop, the FIRST variable FIRST.AGE. The code is somewhat more general by only testing for the string “FIRST.” using the colon modifier on the EQ operator.

⓮ Write a variable name to the first row of the output CSV. Note the use of trailing @ so each execution of the put statement writes each name on to the same line.

⓯ Release the tightly held data line.

⓰ Return to the line following the link statement.

This output from this program are CSV files one for each level of AGE; ‘11.csv’, ‘12.csv’, ’13.csv’, etc.

CONCLUSION

This has been a quick look at the SAS call routine VNEXT, the application while simple illustrates the basic function of the routine and how it might be applied more generally.

REFERENCES

CALL VNEXT, SAS(R) 9.4 Functions and CALL Routines: Reference, Fourth Edition

http://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#n0sd9cb58tfva5n1wdoj6my8xnh8.htm

CONTACT INFORMATION

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

John Henry King
Ouachita Clinical Data Services, Inc.
870-356-3033
iebupdte@gmail.com