

There and Back Again: Navigating between a SASHELP View and the Real World

Anita Rocha, Center for Studies in Demography and Ecology

University of Washington, Seattle, WA

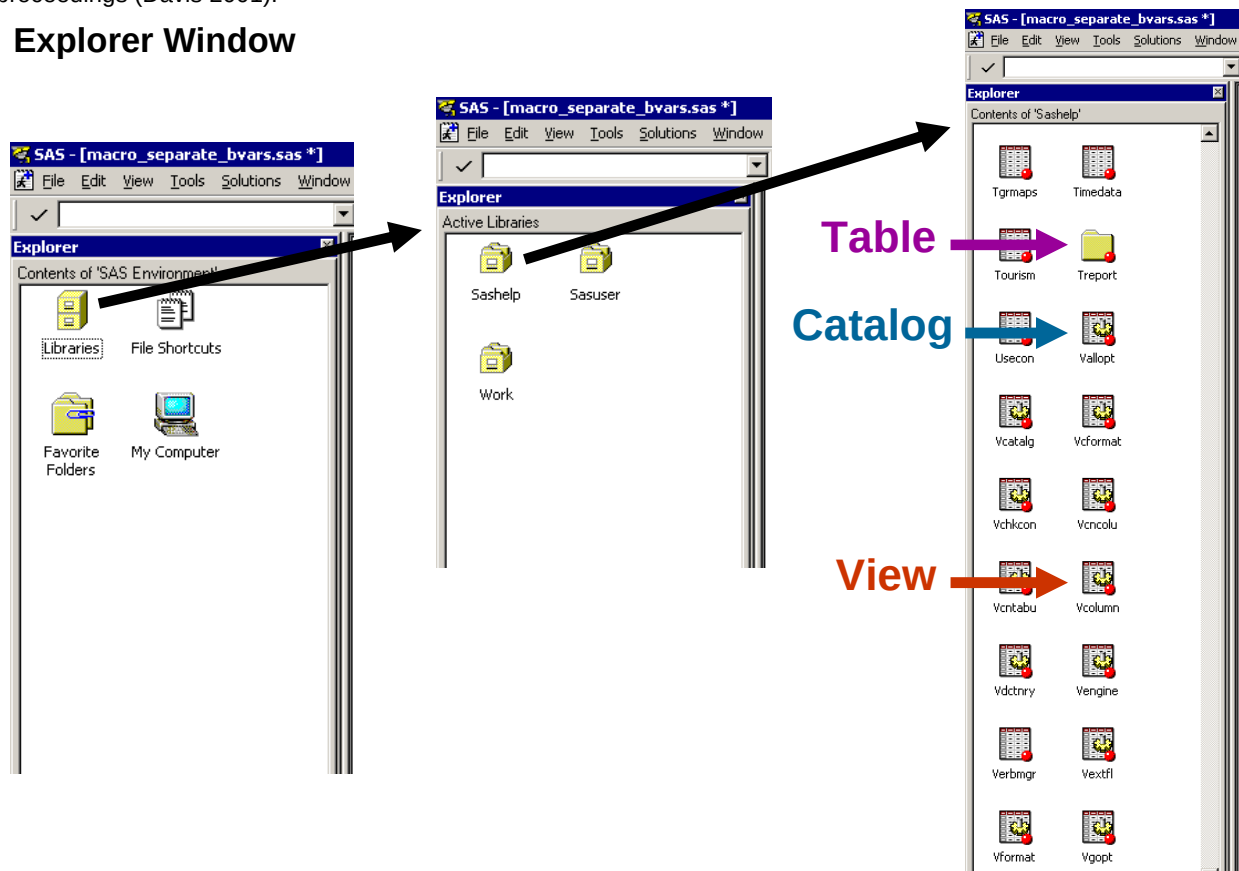
ABSTRACT

A real strength of SAS® compared to other statistical tools is its ability to handle huge tables. The built-in efficiencies, however, limit its flexibility for simple table manipulation using information at the meta-data level. For example, how can one reduce the table size by invoking, say, a selection criteria involving the variable names rather than the values of a variable? In this paper we explore the use of the SASHELP views, particularly the view called VCOLUMN. Then we use a macro to produce a simple dynamic SAS program to limit a large table based on a variable name criterion. We will go from meta-data to the real world of tables, with a little automation thrown in. This example is easily extendible to manipulation of formats, labels and other components of the meta-data. The examples herein were run on SAS 9.1 for Windows requiring only Base SAS.

WHAT ARE SASHELP VIEWS?

Lots of information is available to a user just by opening a SAS session. PROC SQL can be used to create views onto DICTIONARY tables, the read-only storage area for information about the current SAS session. (See the Concepts: SQL Procedure, The SQL Procedure, *SAS OnlineDoc® 9.1.*) What's a view? It's essentially a SQL query, typically stored in a catalog. But you don't need to run an SQL procedure or a stored query to gain access to key information. There are views ready and available in a library called SASHELP. In particular, SASHELP contains a view called VCOLUMN with elements representing table contents, like variable names and labels, that can be used immediately. For a more detailed description of dictionary views, see the Michael Davis paper in the SUGI 26 proceedings (Davis 2001).

Explorer Window



To see what views are available at the start of a SAS session, first open SAS, go to the Explorer window, click on

LIBRARIES, and then click on SASHELP. There you'll see the default views.

CONTENTS OF A VIEW

By running PROC CONTENTS on the view VCOLUMN in the library SASHELP, one will note there are no observations. Surprise! Since this is not a table there are no observations associated with it. It is a view onto a DICTIONARY table containing meta-data about variables. To find out from which table the view is drawing its information, you can use the following SQL procedure.

```
proc sql;
describe view sashelp.vcolumn;
```

The results will display in the LOG as follows.

```
select *
from DICTIONARY.COLUMNS;
```

So VCOLUMN is a view onto the read-only dictionary table COLUMNS. The only way to access the contents of the table COLUMNS is via the view VCOLUMN or the manual construction of a different view using PROC SQL. Still, this meta-data will prove useful in a certain class of problems.

THE PROBLEM: REDUCING THE SIZE OF A TABLE

We were approached with the problem of reducing the size of a large table with many variables. In particular, the variable naming convention comprised of each variable associated with a particular hierarchy started with a different letter. For example, BBIRTH_ORDER, BSEX, and BGPA were associated with individual student information, while variables ANUMBER_GRADS, AVERAGE_INCOME, and AVERAGE_GPA were associated with high school level information, and so on. The user wanted to create a new table retaining only the variables from a single hierarchical level, namely variables that began with the letter 'B', the individual student information.

SOLUTION 1: KEEP STATEMENT WITH VARIABLES MANUALLY LISTED

One solution was to use the KEEP statement in a DATA STEP while listing the target variables manually. As an illustration we create here a small table and show the relevant data step, keeping only variables that begin with the letter 'B'.

```
data sample;
  input anumber_grads bbirth_order aaverage_income bsex aaverage_gpa bgpa;
  datalines;
160 2 30000 1 2.7 3.5
321 1 75000 2 3.1 2.1
114 3 72000 2 2.9 3.8
368 1 28000 1 3.3 1.5
;

data sample_b1;
  set sample;
  keep bbirth_order bsex bgpa;
run;
```

This works well for this simple table. In reality, however, the table presented had several thousand variables, a good portion starting with the letter 'B', and many observations. If by chance the target variables had consecutively numbered variable names or consecutive VARNUMs, those could be exploited to advantage with VAR1-VARN syntax in the KEEP statement. Unfortunately, that was not the case. Using the KEEP statement this way is now a lot less attractive for reducing the size of the table. It would involve extensive typing or cutting-and-pasting the target variables to the keep statement manually, risking errors. Moreover, if there is a subsequent request to retain variables for a different hierarchical level from the table, say for a set of variables starting with the letter 'A', typing a new set of variables again becomes tedious, not to mention the problem of accidentally omitting one of the target variables from a cumbersome list.

USING META-DATA PROVIDED BY SASHELP VIEWS

Another, more tractable solution was to use the SASHELP view VCOLUMN that already has the meta-data ready for manipulation. Specifically, we want access to variable names in the table so that variables starting with 'B' can be isolated in a keep statement. First, let's display the contents of the view VCOLUMN.

```
proc contents data=sashelp.vcolumn;
run;
```

Note in the OUTPUT window that the variable MEMNAME labeled 'Member Name' and NAME labeled 'Column Name' represent the table names and the variable names in the tables, respectively. If these variables sound odd or non-descriptive, the SAS programmer can print out the values to confirm that, indeed, MEMNAME represents the table names and NAME represents the variable names.

Caution: This will only identify the target table and variables if that table is "seen" by SAS. In other words, table contents will be available via the view onto the Dictionary table COLUMNS only if the table is in a recognized library of SAS, either from the execution of a LIBNAME statement or by creating a table stored in the WORK library, as in the table SAMPLE above.

Assuming the above table SAMPLE has been created and thus in the WORK library, the variable names in the table can be viewed as follows. Note that the contents of this view has been subset by the WHERE statement.

```
proc print data=sashelp.vcolumn(where=(memname eq 'SAMPLE')) noobs;
  var memname name;
run;
```

The following is displayed in the OUTPUT Window.

memname	name
SAMPLE	anumber_grads
SAMPLE	bbirth_order
SAMPLE	aaverage_income
SAMPLE	bsex
SAMPLE	aaverage_gpa
SAMPLE	bgpa

Not surprisingly, the variable names in the table SAMPLE created above look a lot like values of the variable NAME in the view VCOLUMN. The variables MEMNAME and NAME can be treated much like one would treat any variables of a table, with the restriction that the original table and view can not be altered. So trying to modify the original Dictionary table COLUMNS or the view VCOLUMN will result in an error. No problem. This useful information can still be exploited and provide plenty of flexibility for meta-data manipulation.

SOLUTION 2: CREATE A MACRO VARIABLE CONTAINING A LIST OF TARGET VARIABLES

Back to our original problem, that is to reduce the size of the SAMPLE dataset by keeping only variables that start with the letter 'B', another solution would be to create a macro variable that can be used in the data step in place of the long list of manually entered variables. Below is the SAS code to create a macro variable B_VARS that will house a list of the target variables.

```
data _null_;
  set sashelp.vcolumn(where=(memname eq 'SAMPLE'));
  length vars $ 200;
  retain vars;
  if substr(upcase(name), 1, 1) eq 'B' then
    vars = trim(vars) || ' ' || name;
  call symput('b_vars', trim(left(vars)));
run;
```

Note that in the DATA STEP designated with _null_ we create a temporary character variable VARS of length 200 to house the list, identify the 'B' variables using a SUBSTR function, create the list by appending each 'B' variable to the VARS variable, and finally save the list as a macro variable B_VARS using CALL SYMPUT. The length 200 was chosen arbitrarily and can be adjusted to the estimated length of the resulting list. Caution: Be sure to overestimate the length of variable VARS so as not to cut-off the list prematurely.

To view the contents of the macro variable, run the following code. Note the prefix '&' identifies the macro variable.

```
%put b_vars = &b_vars;
```

As a result, the following should display in the LOG (not the OUTPUT) window.

```
b_vars = bbirth_order bsex bgpa
```

So creating the new table is simplified by using the KEEP statement plus the macro variable B_VARS.

```
data sample_b2;
  set sample;
  keep &b_vars;
run;
```

Here there is no exhaustive typing, no concern about mistyping a variable name, and no problem with omitting a variable name altogether from a lengthy list. Since the maximum length of any character value in SAS is 32,767 bytes, the length of the VARS variable can be very, very long. For the *real life* table where there were hundreds of 'B' variables, using a very long character variable VARS is adequate. But for even greater flexibility without the need to consider the length of the VARS variable, consider creating what may be called a *virtual* macro array.

SOLUTION 3: USE A SERIES OF MACRO VARIABLES, LIKE AN ARRAY

This *virtual* macro array example borrows generously from the 'REGION' example on page 71 in *Carpenter's Complete Guide to the SAS® Macro Language* by Art Carpenter (1998). Besides the solution developed below, this technique has broad application so it can be a useful addition to any SAS programmer's tool kit.

The basic idea is to assign one macro variable to the value of one variable name, and to do that for every target variable in the table. Let the macro variable names contain enumeration so that the macro variable series can be easily and exhaustively accessed later, much like the index for a traditional array within a DATA STEP.

First, create a series of macro variables with values of the names of the target 'B' variables.

```
%macro get_bvars();
  data _null_;
    set sashelp.vcolumn(where =(memname eq 'SAMPLE'));
    if upcase(substr(name, 1, 1)) eq 'B' then do;
      i +1;
      ii=put(i,3.);
      call symput('target' || left(ii), trim(name));
      call symput('total', ii);
    end;
  run;
%mend get_bvars;
```

So for all the 'B' variables, a series of macro variables called TARGET1, TARGET2, etc. is created. The enumeration of the TARGET# macro variables is automatic starting at 1 and counting up for as long as there are target variables in the table. Since there are three 'B' variables in the table SAMPLE, there should be three macro variables created. The total number of macro variables created is housed, naturally, in the macro variable TOTAL. To confirm the number of macro variables created in our example, run the macro and then the following PUT statement.

```
%get_bvars;
%put total = &total, target1 = &target1, target2 = &target2, target3=&target3;
```

Look again at the LOG window. It should correctly indicate the three target 'B' variables.

```
total = 3, target1 = bbirth_order, target2 = bsex, target3=bgpa
```

Next, use this series of macro variables in a KEEP statement in a DATA STEP. Note that the macro do-loop simply lists out the values of the enumerated macro variable series.

```
%macro keep_bvars();
  data sample_b3;
```

```

        set sample;
        keep
%do i = 1 %to &total;
    &&target&i
%end;
;
run;
%mend keep_bvars;

```

Now run the macro.

```
%keep_bvars;
```

The double-ampersand (&&) prefixed to the macro variable TARGET in the do-loop is no typo but essential. In the first pass of the macro processor, the double ampersand (&&) resolves to a single ampersand (&) while the &i resolves to a numeric as dictated by the do-loop. For example, when &i = 1 the first time through the do-loop, the macro processor will resolve the macro variable as follows.

Macro processor

```

&&target&i   $\xrightarrow{\text{pass 1}}$   &target1   $\xrightarrow{\text{pass 2}}$   bbirth_order

```

To learn more about the resolution of macro variables, see Carpenter (1998) pages 14-15.

SOLUTION 4: GENERALIZE SOLUTION 3 BY ALLOWING VARYING TABLE AND TARGET VARIABLE INPUTS

Will this algorithm for reducing the size of a table be used more than once? If there might be another table or another set of target variables of interest, it is efficient to broaden the use of the macro by allowing the table and target variable criteria to vary. We generalized the above two macros by 1) combining them into a single macro since they operate together, 2) allowing varying inputs for the table and target variables, and 3) displaying the results.

```

%macro subset_table(table=, targetvar=);
    data _null_;
        set sashelp.vcolumn(where =(memname eq "%upcase(&table)"));
        if upcase(substr(name, 1, 1)) eq "%upcase(&targetvar)" then do;
            i +1;
            ii=put(i,3.);
            call symput('target' || left(ii), name);
            call symput('total', ii);
        end;
    run;

    data &table._b3;
        set &table;
        keep
%do i = 1 %to &total;
    &&target&i
%end;
;
run;

    proc contents data=&table._b3;
run;
%mend get_bvars;

```

Finally, we run the macro with inputs SAMPLE and B for table and target variable criteria, respectively.

```
%subset_table(table=sample, targetvar=b);
```

CONCLUSION

There are at least two reasons why a SAS programmer might prefer the virtual macro array solution (corresponding to Solutions 3 and 4). First, it's robust to changes in magnitude such as a rapid growth in the number of target variables. Second and perhaps most important, the ideas behind the *virtual* macro array are flexible and scalable to a broad class of applications that involve meta-data and repetitive manipulation of lots of information. For example, a similar approach can be used across tables with long lists of variables to modify and re-assign labels, or to alter and assign formats. If there is systematic criteria for manipulating any information at the meta-data level, especially on a large scale, this is a flexible candidate technique.

REFERENCES

Carpenter, Art, 1998, *Carpenter's Complete Guide to the SAS Macro Language*, Cary, NC: SAS Institute Inc., pp14-15, 70-74.

Concepts: SQL Procedure, The SQL Procedure, *SAS OnlineDoc® 9.1.2*, ©2002-2004, Cary, NC: The SAS Institute Inc. (<http://support.sas.com/onlinedoc/912/docMainpage.jsp>).

ACKNOWLEDGMENTS

The author would like to acknowledge support from the Center for Studies in Demography and Ecology, University of Washington, Box 353412, Seattle, WA 98195-3412 (<http://csde.washington.edu>).

Also, the author would like to thank the graduate students and other researchers that study demography at the University of Washington many of whom consistently present with intriguing problems.

Finally, the author has deep appreciation for all she learned about macro programming from Art Carpenter and his comprehensive book.

RECOMMENDED READING

Davis, Michael, 2001, You Could Look It Up: An Introduction to SASHELP Dictionary Views, *SUGI 26 Proceedings*, Cary, NC: The SAS Institute Inc.

CONTACT INFORMATION

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

Anita Rocha
Center for Studies in Demography and Ecology
University of Washington
Box 353412
Seattle, WA 98195-3412
Work Phone: 206.616.6687
Fax: 206.616.8135
E-mail: alrocha@u.washington.edu
Web: <http://csde.washington.edu/>

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.