

Let SASHELP Help You – Easy Ways SAS Can Help You with Your Dynamic Programming and Validation

Stacey D. Phillips and Gary Klein, STATPROBE, INC, San Diego, CA

ABSTRACT

SASHELP tables are data views, contained in the SASHELP library, that automatically collect and store information about the current SAS session on a constant basis. These views are a wealth of information about your session and can be accessed via standard data step procedures to improve and automate your code. Through the combined use of SASHELP views and macros, programming can be easily transported between different programs and projects as your code becomes more dynamic – that is, less data-driven. This paper will explore some simple techniques using the information contained in the SASHELP library that can be utilized in general programming and the validation process to automate your programming procedures as well as make them consistent across programs and projects.

INTRODUCTION

As programmers, we are constantly striving to make our code more useable, more adaptable and generally more dynamic. Dynamic code allows us to write programs before we see the data, as the code does not depend on the specific parameters of the data, but rather general parameters that are consistent across all SAS data. Further, developing dynamic code allows us to move the code between programs with little to no alteration. The ability to write dynamic code makes us better, more efficient and less data dependent programmers in the long-run.

In the pharmaceutical industry, statistical programmers often find themselves doing the same tasks over and over again. Because of FDA submission requirements, we find that there is much similarity and uniformity in the tasks we do. It quickly becomes clear that developing processes to automate these tasks is an excellent investment in our programming time. This paper will demonstrate 6 specific examples where SASHELP data views can be utilized to make these processes more automated and dynamically written.

SASHELP tables are data views that collect and retain information about the current SAS session. This information is collected on a wide range of topics including active datasets, current system options and parameters of the macro symbol table. SASHELP views are a wealth of information about what's going on with SAS at the time of usage and can be tapped for specific information that is difficult, if not impossible to get elsewhere. The ability to access and utilize this information can greatly expand the flexibility and adaptability of the programmer's code.

THE BASICS

SASHELP views are read-only SAS data views that contain a multitude of information about the current SAS session. This includes information about datasets, macros, current SAS system options, libraries and numerous other bits of information. The information contained in these views can be accessed via the SASHELP library using regular library referencing functionality (eg. sashelp.vcolumn). SASHELP views are convenient because they behave in the same way as SAS datasets and can be manipulated similarly using the data and procedure steps.

Another type of view containing similar information in SAS is called a dictionary table. While you can retrieve the same information from dictionary tables as you can SASHELP views, dictionary tables are only available within PROC SQL and cannot be manipulated via the data step. The major difference between the two comes down to preference. If you prefer working in the data step, then SASHELP views are for you. If you prefer PROC SQL then you will likely prefer using dictionary tables. We will focus on SASHELP views in this paper, though we want you to be aware of the alternative.

There are currently 16 SASHELP views available in version 8.2 of the SAS System and the table below itemizes each of them by name and functionality. If you are interested in knowing the names of the specific parameters within an individual SASHELP view, run a PROC CONTENTS referencing the particular view you're interested in. Some of the SASHELP views are very large and contain so much information that it is necessary to use subsetting WHERE statements to limit the amount of generated output and computer usage time. For example, the VCOLUMN view provides information about name, type, length, member and library name for all variables in all datasets in the current session. If you haven't limited your request to a particular library, VCOLUMN will return one observation for every variable in every dataset in all of the available libraries which can get cumbersome very quickly. Fortunately, it is simple to select out which records are needed. The following example demonstrates several ways to limit the information in SASHELP.VCOLUMN using various data and procedure step techniques.

A. PROC PRINT of all variable information in all datasets in a given library named DATA.

```
proc print data=sashelp.vcolumn (where=(library='DATA'));
run;
```

B. Create a dataset named VARS that contains all of the variable information in the DEMO dataset located in the DATA library.

```
data vars;
  set sashelp.vcolumn (where=(library='DATA' and memname='DEMO'));
run;
```

C. Or, create a dataset named RACEINFO by sorting SASHELP.VCOLUMN and creating an output dataset that contains information on variables named RACE located in the DATA library.

```
proc sort data=sashelp.vcolumn (where=(library='DATA' and name='RACE'))
  out=raceinfo;
  by memname;
run;
```

The above examples are just a few of the very simple uses of SASHELP.VCOLUMN. There are many, many more ways to use this view and the 15 others. In this paper, we provide 6 easy ways that you can make SASHELP views work for you. The examples and code we provide are relatively basic, however the idea is to start from something simple and build upon it as your understanding of the concepts grow. The examples we provide can easily be combined with macro language to make them more general and dynamic. It is important to note that there are many other ways to accomplish the tasks that we outline below; however we provide this information simply as another tool in your SAS toolkit.

AVAILABLE SASHELP VIEWS¹

SASHELP Name	SQL Name	Function
SASHELP.VCATALG	dictionary.catalogs	Contains a row of information for each SAS catalog: the name, location, and type of catalog (MACRO, FORMAT, and so on).
SASHELP.VCOLUMN	dictionary.columns	Includes one observation for every variable available in the session. Information includes name, type, length, and the library and member name of the data set in which the variable resides.
SASHELP.VEXTFL	dictionary.extfiles	Lists the file references assigned in the SAS session and the name of the external file they are assigned to.
SASHELP.VINDEX	dictionary.indexes	Lists the indexes available in the session as well as the data sets in which they reside and the variables they index.
SASHELP.VMACRO	dictionary.macros	A data set containing the names and values of all macro variables available to the session. Includes automatic and user-defined variables.
SASHELP.VMEMBERS	dictionary.members	Lists all data sets, catalogs, views, and multidimensional databases available in the session.
SASHELP.VOPTION	dictionary.options	Includes the name, description, current setting and scope for each option. Scope defines whether the option is operating system specific (level= 'Host') or available on all operating systems (level= 'Portable').
SASHELP.VTABLE	dictionary.tables	Contains the library reference, data set name, and other information for every data set available in the session. Does not include data views.

SASHELP.VTITLE	dictionary.titles	Gives the location (type), number and value of each title and footnote currently available. Type= 'F' indicates a footnote. Type= 'T' indicates a title.
SASHELP.VVIEW	dictionary.views	Lists the name and engine types of all views available in the session. Engine= 'SASDSV' indicates a DATA step view. Engine= 'SASESQL' indicates an SQL view. SQL views are portable across operating systems. DATA step views are not.
SASHELP.VSACCES	subset of dictionary.members where memtype='ACCESS'	Lists SAS/ACCESS views available in the session.
SASHELP.VSCATLG	subset of dictionary.members where memtype='CATALOG'	Quick list of catalogs and the libraries they are located in. VSCATLG is smaller than VCATLG and therefore runs faster.
SASHELP.VSLIB	subset of dictionary.members	Lists library names and the operating system locations they are assigned to.
SASHELP.VSTABLE	subset of dictionary.members where memtype='DATA'	Lists data set names and the libraries they are located in. VSTABLE is smaller than VTABLE and therefore runs faster.
SASHELP.VSTABVW	subset of dictionary.members where memtype='VIEW' or memtype='DATA'	VSTABVW includes both data sets and views and has fewer variables than VTABLE. This means it runs faster and uses fewer system resources.
SASHELP.VSVIEW	subset of dictionary.members where memtype='VIEW'	Includes the name and location of each view but not the type.

TECHNIQUES USING SASHELP DATA VIEWS

EXAMPLE 1 – SASHELP.VTABLE

Task: Comparing datasets between directories to make sure no modifications were made.

```
%macro compare(ds);
  proc compare base=a.&ds compare=b.&ds;
    run;
%mend;

data _null_;
  set sashelp.vtable (where=(upcase(libname)='A'));
  call execute('%compare(' || memname || ')');
run;
```

Discussion: It is common to have a “working” directory for program development and a “production” directory where the final programs, datasets, documentation, etc. are stored. Generally, the two directories are meant to be exact copies of each other; however the production directory is locked at a certain point in time once we are satisfied that everything is complete and correct. Just prior to locking the directory, we might want to compare the two directories to make sure no unexpected changes have been made.

In this example, the user is comparing all of the datasets in LIBRARY A to all of the datasets in LIBRARY B to ensure that nothing has been changed. Using SASHELP.VTABLE we select all of the datasets in LIBRARY A and then call the %compare macro. The %compare macro executes for every existence of a MEMNAME (dataset) in VTABLE, thus comparing all datasets that exist in LIBRARY A to all datasets with the same names in LIBRARY B. Note that it is not necessary to know the names of the datasets you want to compare, as this is automatically generated from the VTABLE view.

EXAMPLE 2 – SASHELP.VCOLUMN

Task: Check all variables in a particular library for missing labels.

```
data _null_;
  set sashelp.vcolumn (where=(upcase(libname)='A'));

  if label='' then put "WAR" "NING: The variable " name "from "
    libname ". " memname " does not have a label.";
run;
```

Discussion: It is possible to run a PROC CONTENTS on all of the data sets in a particular directory, then output the information to a dataset to check for missing labels, formats, lengths, etc. Another option is to automate a step in our validation process by checking for these issues and having the results printed to the log using PUT statements. Using data _null_ we select all of the datasets in LIBRARY A as before. We check for the missing label (or format, length, etc.) and a warning is then printed to the log that names the problem variable, its complete LIBREF (libname.memname) and the fact that it is missing a label.

EXAMPLE 3 – SASHELP.VTABLE

Task: Compare the number of observations in selected datasets.

```
%macro checkds(lib, ds);
  proc sql noprint;
    select count(distinct subjid) into: subjcount from &lib..demog;
  quit;

  %local i;
  %let i=1;
  %do %while (%scan(%str(&ds), &i, %str( )) ^= );
    %let y=%scan( %str(&ds), &i, %str( ));

    data _null_;
      set sashelp.vtable (where=(upcase(libname)=upcase("&lib") and
        upcase(memname)=upcase("&y")));
      if nobs < &subjcount then put
        "WAR" "NING: The DATASET &Y HAS LESS THAN &subjcount SUBJECTS";
      else if nobs > &subjcount then put
        "WAR" "NING: The DATASET &Y HAS MORE THAN &subjcount SUBJECTS";
    run;

    %let i=%eval(&i+1);
  %end;

%mend;
%checkds(a, safety efficacy)
```

Discussion: When creating datasets, it is common to check a newly created dataset against a dataset known to have the correct and validated number of subjects in it. In the pharmaceutical industry, the dataset that is frequently used as the benchmark in this type of check is the DEMOG dataset where all of the demographic information for a particular study is stored. Assuming that we know the DEMOG dataset has been fully validated and contains one record per subject as expected, we can run this utility to check other datasets that are expected to have the same number of subjects.

In this case we want to check and make sure the number of subjects in the SAFETY and EFFICACY datasets are the same as the number of subjects in the DEMOG dataset. First, we utilize PROC SQL to count the number of subjects in DEMOG and put that into a macro variable named SUBJCOUNT. We then access SASHELP.VTABLE, select the datasets of interest and cross-check the number of observations in said dataset (using NOBS) against the number in the macro variable SUBJCOUNT. If the number of observations (subjects) is either more or less than our benchmark dataset, a warning is issued to the log.

EXAMPLE 4 – SASHELP.VCOLUMN

Task: Check to make sure a key variable is in all of the datasets of a library.

```
proc sort data=sashelp.vcolumn out=varcheck(keep=memname name );
  where upcase(libname)='A';
  by memname;
run;

data _null_;
  retain flag 0;
  set varcheck;
  by memname;

  if first.memname then pflag=0;
  if name='PROTOCOL' then pflag=1;
  if last.memname and pflag=0 then put
    "WAR" "NING:  The DATASET " memname "is missing the protocol variable.";
run;
```

Discussion: Sometimes we're interested in making sure a key variable has not inadvertently been dropped or deleted from a dataset or datasets. This example is checking to make sure that the very important variable PROTOCOL is in all of the datasets in a library. Using SASHELP.VCOLUMN, we create a dataset named VARCHHECK that contains all of the MEMNAMEs (or dataset names) and their respective NAMEs (variable names) in LIBRARY A. We then create a flag and use first./last. to process each dataset individually and check for the variable PROTOCOL. If the variable does not exist in a particular dataset, a warning is sent to the log.

EXAMPLE 5 – SASHELP.VLIB

Task: Look up the path of source data and print to the end of a table.

```
%macro getpath(ds);
  *Create macrovariable whose name is the libname *;
  * and whose value is its path *;

  data _null_;
    set sashelp.vslib;
    call symput(libname,compress(path));
  run;

  %local y;
  %let y=%scan(&ds, 1, .);
  %put y=&y;

  %let printds=%sysfunc(tranwrd(&ds, &y., &&y)).sas7bdat;
  %put printds=&printds.;
%mend;
%getpath(rawdata.demog)
```

Discussion: It could be the case where we inherit an older program that includes some initialization file to set up the libnames of our data. This initialization file may have restricted access or change frequently and the contents of it may be unknown. However, the information contained in this file, such as the path of the data, may be important to us if, for example, we want to print the path of our data in a footnote. We can access SASHELP.VSTABLE which contains all of our libnames and the paths with which they are associated. With a moderate amount of macro-programming, we can replace the library with the path, e.g., rawdata.demog can be replaced with /project/study/demog.sas7bdat.

EXAMPLE 6 – SASHELP.VSTABLE

Task: Check for the existence of a dataset and program conditionally based on this information.

```
%let labflag=0;
%let lablist=;

data datalist;
  length lablist $200;
  retain lablist;
  set sashelp.vstable(where=(upcase(libname)='RAWDATA')) end=eof;

  *Reassign macro variable labflag if ALLLAB dataset exists *;
  if upcase(memname)='ALLLAB' then call symput('labflag','1');
```

```

*Create a list of all lab datasets *;
if index(upcase(memname),'LAB') then
  lablist=trim(left(lablist))||"rawdata."||compress(memname);

*Set list of labs to a macro variable to be used later *;
if eof then call symput('lablist',trim(left(lablist)));
run;

%macro run_eoswk;
  %if &labflag eq 1 %then %do;
    %eoswk;          * Run EOSWK because dataset ALLLAB exists *;
  %end;
  %else %if &labflag ^= 1 %then %do;

    *Create ALLLAB dataset *;
    data rawdata.alllab;
      set &lablist;
    run;

    *Run EOSWK using created ALLLAB data *;
    %eoswk;

    *Delete ALLLAB when done with macro *;
    proc datasets library=rawdata;
      delete alllab;
    quit;
  %end;
%mend;

%run_eoswk;

```

Discussion: Many times we are forced to use validated macros that do not exactly fit our data. In this example, the client requires us to use the macro %eoswk. This macro runs off of a dataset ALLLAB which is the concatenation of various other lab datasets. As we may or may not have the ALLLAB dataset, we will check for its existence using SASHELP.VSTABLE. If it doesn't exist, we will build ALLLAB from other lab datasets, run the %eoswk macro, and then delete the ALLLAB dataset when finished. If ALLLAB existed in the first place, we could just call %eoswk.

CONCLUSION

We have demonstrated a few of the many, many uses of the SASHELP library in the above examples. Note that the example code can easily be transported to programs you may be currently working on with little alteration. From the above examples, it is clear that the information contained in the SASHELP library creates endless possibilities in the realm of automating and improving both general programming and validation techniques. SASHELP's data views are data-rich sources of information that can be exploited to automate processes. By learning to access and manipulate the information contained within the SASHELP library, you will be well on your way to being a more dynamic programmer.

REFERENCES

1. Thielbar, Melinda, "The SASHELP Library: It Really Does Help You Manage Data," *SAS Bits and Bytes*, 2002, <http://support.sas.com/sassamples/bitsandbytes/sashelp.html>.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Stacey D. Phillips
 STATPROBE, Inc.
 10052 Mesa Ridge Court
 Suite 200
 San Diego, CA 92121
 Email: stacey.phillips@sd.statprobe.com

Gary Klein
 STATPROBE, Inc.
 10052 Mesa Ridge Court
 Suite 200
 San Diego, CA 92121
 Work Phone: (858) 597-1000 ext. 2460
 Email: gary.klein@sd.statprobe.com

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.