Paper CC21

Touring the SASHelp Neighborhood

William Turner, ICON Clinical Research, North Wales, PA

ABSTRACT

The SASHelp data library can be of tremendous help to a user who is interested in programmatically accessing metadata that describes data sets and catalogs in data libraries defined by the user, as well as key operational information for the SAS Environment.

After a tour of some of the noteworthy landmarks in the SASHelp neighborhood, I will discuss ways in which that information can be employed to help you write analysis data set specifications and sanitize an interactive workspace in order to ensure you always submit your SAS programs with a clean slate.

The principles in this presentation should be applicable to SAS versions 6.12 and higher on any operating system, and the ideas and methods are suitable for SAS users of all experience levels.

INTRODUCTION

The Merriam-Webster OnLine Dictionary defines metadata as “data that provides data about other data.” The metadata in the SASHelp data library provide direct access to information about your data and your SAS session.

THE LANGUAGE OF SAS FILES

SAS files are comprised of a variety of objects that are recognizable to SAS and which reside in a SAS data library. In addition to the common data set and catalog member types, SAS files also include more exotic items such as index files, stored programs, and multidimensional databases (MDDBs).

In the SAS documentation, the term “SAS data set” is broadly defined as a collection of data values that has the familiar tabular structure of rows (observations) and columns (variables), and which also contains descriptive information about the data set. This descriptive information includes characteristics of the variables (e.g., name, type, length, format, label), as well as the characteristics of the file, such the data set label, date/time of the last modification, security features, and counts of the variables and observations.

What many users in the clinical arena commonly refer to as “SAS data sets” are, more specifically, SAS data files. SAS data files, identified as MEMTYPE=DATA in procedures such as PROC DATASETS, are a distinct type of SAS data set in that they physically contain data values.

SAS DATA VIEWS

In contrast, a SAS data view (MEMTYPE=VIEW) is not a physical store of data. It is actually a collection of references to other data sources that can be represented as a virtual SAS data file. Since SAS data views are just instructions to retrieve data, one cannot modify the source data object by way of the view. Although the two differ somewhat with regard to their respective purposes and methods of execution, it is helpful to think of SAS data views in the same category as stored compiled data step programs (i.e., as code or instructions rather than data).

While users can define their own SAS data views, the native data views that populate the SASHelp data library are the focus of this discussion. The term “native” denotes that these views are automatically created by SAS with no user instruction required. The data which are the source of the SASHelp data views are called DICTIONARY tables, and these are accessible to users only through PROC SQL. Alternatively, SASHelp data views are accessible through either data steps or procedures. This flexibility makes data views a better choice for use in everyday programming tasks.

THE SASHELP NEIGHBORHOOD

Inside the SAS Environment, also referred to as interactive SAS, the easiest method to view the contents of the SASHelp data library is with the SAS Explorer (Libraries display in SAS version 6.12). A screen shot of the library is shown below in Figure 1.

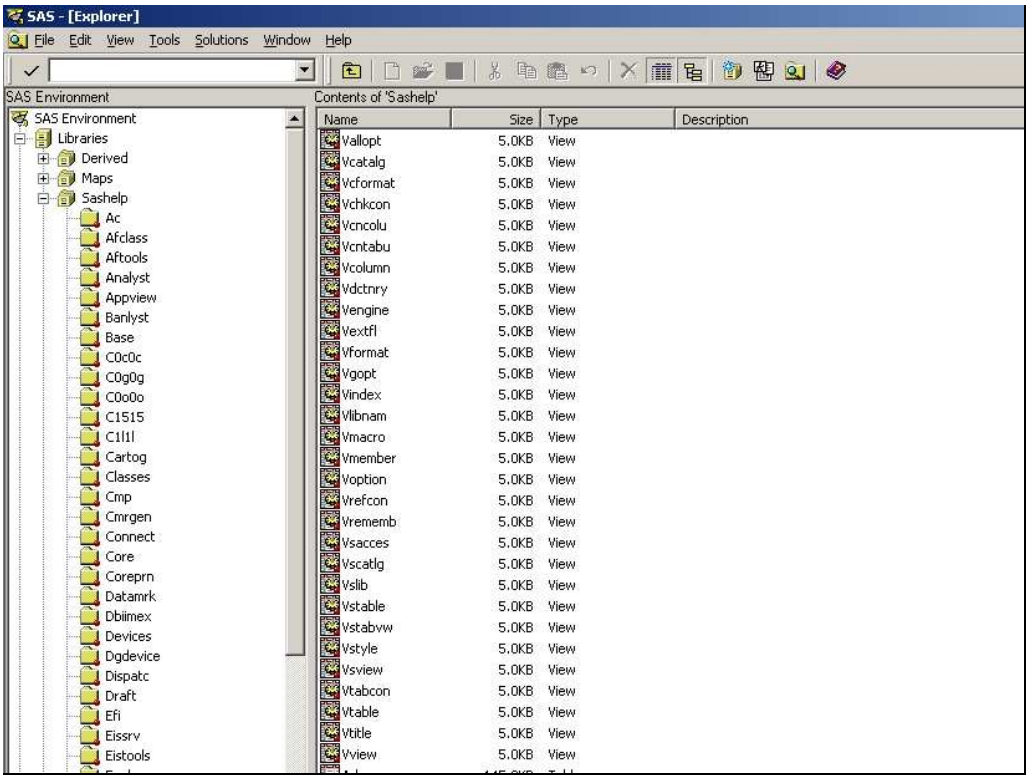


Figure 1. A list of native views in the SASHelp data library

TAKING A TOUR OF THE NEIGHBORHOOD

Of approximately thirty native data views in the SASHelp library, there are eight views which are of particular interest. Figure 2 lists those eight views and the key fields they contain, as well as some possible applications of that data. Note that the term "member" below refers to a SAS file (i.e., a data set, catalog, etc.).

Data View Name	Key Information	Applications
VMEMBER	SAS files names by library name and member type, including SAS engine name, use of index, and physical path	Verify existence of a particular data set
VTABLE	SAS files names by library name and member type, including data set label, file creation and modification datetime, counts of observations and variables, length of longest variable name and label, and other descriptive values	Check for data sets with zero observations, scan entire data libraries for files with variable names that exceed 8 characters or labels that exceed 40 characters, or verify the integrity of dependencies between data sets by datetime stamp
VCOLUMN	SAS data set variables by library, member name, and member type, including variable type, length, order in table, label, format, informat, and other descriptive values	Check for the existence of a variable, build a list of variables as the basis for a rough draft of analysis data set specifications
VMACRO	Macro variable names and values by scope	Clear values of global macro variables
VTITLE	Text of titles and footnotes by number	Access title/footnote values programmatically
VEXTFL	Names of assigned external file references, including physical path name	Check for assignment status of external file references and verify associated physical paths
VLIBNAM	Names of assigned data libraries by engine name, including physical path name	Capture the physical path of an assigned data library
VOPTION	Names, descriptions, and settings of SAS system options by option type and group	Verify option settings

Figure 2. Useful views in the SASHelp data library

If you would prefer to access descriptive information about the SASHelp library outside of the graphical interface, a PROC CONTENTS of the target data view is the most direct method. The code used to create the view from the source Dictionary table can be displayed using a DESCRIBE VIEW statement in PROC SQL.

```
178  proc sql;
179    describe view sashelp.vcolumn;
NOTE: SQL view SASHELP.VCOLUMN is defined as:

        select *
          from DICTIONARY.COLUMNS;

180
181    describe table dictionary.macros;
NOTE: SQL table DICTIONARY.MACROS was created like:

create table DICTIONARY.MACROS
  (
   scope char(9) label='Macro Scope',
   name char(32) label='Macro Variable Name',
   offset num label='Offset into Macro Variable',
   value char(200) label='Macro Variable Value'
  );

182  quit;
NOTE: PROCEDURE SQL used:
      real time           0.00 seconds
      cpu time            0.00 seconds

183
184  run;
```

It's worth mentioning the structure of the VMACRO view, as it is slightly more complicated than it first presents. The width of the field which holds the macro variable values (VMACRO.VALUE) is only 200 characters. Obviously macro variables can hold considerably more data than 200 characters (in SAS version 9.1.3, a macro variable can hold up to 65,534 characters).

The view accommodates values greater than 200 characters by adding additional observations to the file and incrementing the value of the variable VMACRO.OFFSET. The default value of OFFSET is zero, and its value increases by 200 for each observation added for a particular macro variable. Figure 3 shows how VMACRO stores a macro variable value of extended length.

VIEWTABLE: Sashelp.Vmacro

	Macro Scope	Macro Variable Name	Offset into Macro Variable	Macro Variable Value
3	GLOBAL	LINCOLN	0	▯Four score and seven years ago our fathers brought forth on this continent▯ a new nation▯ conceived in Liberty▯ and dedicated to the proposition that all men are created equal. Now we are engaged in
4	GLOBAL	LINCOLN	200	a great civil war▯ testing whether that nation▯ or any nation so conceived and so dedicated▯ can long endure. We are met on a great battle▯field of that war. We have come to dedicate a portion of tha
5	GLOBAL	LINCOLN	400	t field▯ as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But▯ in a larger sense▯ we can not dedi
6	GLOBAL	LINCOLN	600	cate ▯▯ we can not consecrate ▯▯ we can not hallow ▯▯ this ground. The brave men▯ living and dead▯ who struggled here▯ have consecrated it▯ far above our poor power to add or detract. The world will
7	GLOBAL	LINCOLN	800	little note▯ nor long remember what we say here▯ but it can never forget what they did here. It is for us the living▯ rather▯ to be dedicated here to the unfinished work which they who fought here hav
8	GLOBAL	LINCOLN	1000	e thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us ▯▯ that from these honored dead we take increased devotion to that cause for which they ga
9	GLOBAL	LINCOLN	1200	ve the last full measure of devotion ▯▯ that we here highly resolve that these dead shall not have died in vain ▯▯ that this nation▯ under God▯ shall have a new birth of freedom ▯▯ and that government
10	GLOBAL	LINCOLN	1400	of the people▯ by the people▯ for the people▯ shall not perish from the earth.▯

Figure 3. Example of an extended-length macro variable value

ACCESSING DATA VIEWS PROGRAMMATICALLY

As noted above, native views in the SASHelp library can be accessed through a data step, PROC SQL, or other procedure in the same manner as a SAS data file. Views can take a little longer to process than one might expect, since SAS is not simply pulling in a single file, but rather executing a block of code which must look across multiple data sources in order to construct your view. This performance will vary depending upon the number of libraries assigned, number of data sets they each contain, the level of computing power you have available, and the speed of your network connections (particularly across wide-area networks).

USEFUL APPLICATIONS OF THE SASHELP DATA VIEWS

Now that we know what's out there, it's time to discuss some ways in which this data can help to improve our productivity and effectiveness. Given the large volume of data available and the many different ways in which SAS is used in pharmaceutical programming and beyond, a creative mind could devise myriad applications for this information. I will focus on a few applications that I use on a daily basis.

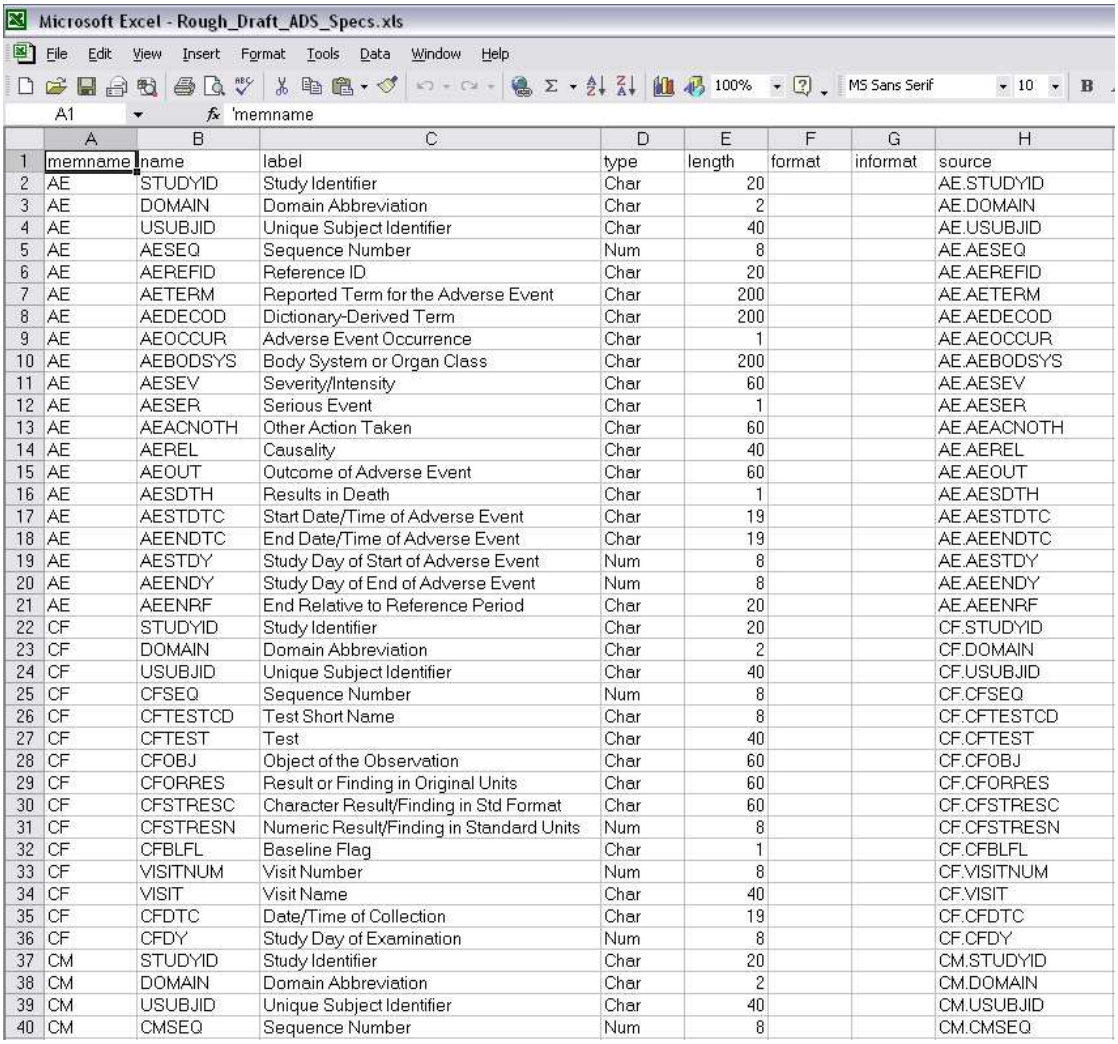
HELPING TO CREATE ANALYSIS DATA SET SPECIFICATIONS

One of the more time-consuming aspects of creating analysis data set specifications is documenting the variables which will be sourced from the raw data sets. Using the data view SASHELP.VCOLUMN, it's a simple process to output desired variable information for all of the data files in your raw data library. The resulting data file can be exported to any of the external file formats supported by the SAS products you have licensed (e.g., CSV, MS Excel, etc.).

```
/***
  Select all variables for all data sets in the raw data library.  Change the
  TYPE field to title case for display purposes and add a new variable to
  SPECS1 to reflect the variable(s) used to create the specification variable
  in question.  Since we're only dealing with the raw fields in SPECS1, this
  value will be self-referential for all of the variables in the specifications.
 ***/
data specs1 (keep=memname name type length varnum label format informat source);
  set sashelp.vcolumn
    (keep=libname memname memtype name type length varnum label format informat
     where=(libname eq 'RAW_DATA' and memtype eq 'DATA')
    )
  ;

  length source  $ 200;

  select (left(type));
    when ('num')  type = 'Num';
    when ('char') type = 'Char';
  end;

  source = trim(left(memname)) || '.' || trim(left(name));

  label source = 'Source Data Variable(s)';

run;

/***
  Use PROC SQL to reorder the variables as I would like to see them displayed in
  the specifications document, and then order the observations by the variable
  order in the raw data set (based on the VARNUM variable).
 ***/
proc sql;
  create table specs2 as
    select memname, name, label, type, length, format, informat, source
      from specs1
      order by memname, varnum
  ;

quit;
```

```
/***
  Export the resulting data file to my preferred analysis data specification
  document format, a MS Excel workbook.
 ***/
proc export data=work.specs2
  outfile="P:\Biometrics_Studies\X123_001\Documentation\Rough_Draft_ADS_Specs.xls"
  dbms=excel2000 replace
;

run;
```

Figure 4 is a screen shot of the resulting Excel workbook, to which I can easily add new derived variables or remove unwanted raw variables. It's only taken me a few minutes to complete, and I have significantly reduced the possibility of making an error in translating the information into the workbook.



Microsoft Excel - Rough_Draft_ADS_Specs.xls

File Edit View Insert Format Tools Data Window Help

A1 'memname

	A	B	C	D	E	F	G	H
1	memname	name	label	type	length	format	informat	source
2	AE	STUDYID	Study Identifier	Char	20			AE.STUDYID
3	AE	DOMAIN	Domain Abbreviation	Char	2			AE.DOMAIN
4	AE	USUBJID	Unique Subject Identifier	Char	40			AE.USUBJID
5	AE	AESEQ	Sequence Number	Num	8			AE.AESEQ
6	AE	AEREFID	Reference ID	Char	20			AE.AEREFID
7	AE	AETERM	Reported Term for the Adverse Event	Char	200			AE.AETERM
8	AE	AEDECOD	Dictionary-Derived Term	Char	200			AE.AEDECOD
9	AE	AEOCCUR	Adverse Event Occurrence	Char	1			AE.AEOCCUR
10	AE	AEBODSYS	Body System or Organ Class	Char	200			AE.AEBODSYS
11	AE	AESEV	Severity/Intensity	Char	60			AE.AESEV
12	AE	AESER	Serious Event	Char	1			AE.AESER
13	AE	AEACNOTH	Other Action Taken	Char	60			AE.AEACNOTH
14	AE	AEREL	Causality	Char	40			AE.AEREL
15	AE	AEOUT	Outcome of Adverse Event	Char	60			AE.AEOUT
16	AE	AESDTH	Results in Death	Char	1			AE.AESDTH
17	AE	AESTDTC	Start Date/Time of Adverse Event	Char	19			AE.AESTDTC
18	AE	AEENDTC	End Date/Time of Adverse Event	Char	19			AE.AEENDTC
19	AE	AESTDY	Study Day of Start of Adverse Event	Num	8			AE.AESTDY
20	AE	AEENDY	Study Day of End of Adverse Event	Num	8			AE.AEENDY
21	AE	AEENRF	End Relative to Reference Period	Char	20			AE.AEENRF
22	CF	STUDYID	Study Identifier	Char	20			CF.STUDYID
23	CF	DOMAIN	Domain Abbreviation	Char	2			CF.DOMAIN
24	CF	USUBJID	Unique Subject Identifier	Char	40			CF.USUBJID
25	CF	CFSEQ	Sequence Number	Num	8			CF.CFSEQ
26	CF	CFTESTCD	Test Short Name	Char	8			CF.CFTESTCD
27	CF	CFTEST	Test	Char	40			CF.CFTEST
28	CF	CFOBJ	Object of the Observation	Char	60			CF.CFOBJ
29	CF	CFORRES	Result or Finding in Original Units	Char	60			CF.CFORRES
30	CF	CFSTRESC	Character Result/Finding in Std Format	Char	60			CF.CFSTRESC
31	CF	CFSTRESN	Numeric Result/Finding in Standard Units	Num	8			CF.CFSTRESN
32	CF	CFBLFL	Baseline Flag	Char	1			CF.CFBLFL
33	CF	VISITNUM	Visit Number	Num	8			CF.VISITNUM
34	CF	VISIT	Visit Name	Char	40			CF.VISIT
35	CF	CFDTC	Date/Time of Collection	Char	19			CF.CFDTC
36	CF	CFDY	Study Day of Examination	Num	8			CF.CFDY
37	CM	STUDYID	Study Identifier	Char	20			CM.STUDYID
38	CM	DOMAIN	Domain Abbreviation	Char	2			CM.DOMAIN
39	CM	USUBJID	Unique Subject Identifier	Char	40			CM.USUBJID
40	CM	CMSEQ	Sequence Number	Num	8			CM.CMSEQ

Figure 4. Variable information in MS Excel after exporting data from SASHELP.VCOLUMN

CLEANING UP THE NEIGHBORHOOD

One of the great benefits of running SAS in batch mode (i.e., from the command line or OS file explorer) is that your program always executes in a clean and consistent SAS session. In contrast, when running programs in the SAS Environment, each program leaves a trail of values and objects in its wake, including temporary format catalogs, titles, footnotes, compiled macros, values in global macro variables, and temporary data sets. Every one of these

objects left over from a submitted program has the potential for unintended consequences when subsequent programs are submitted.

Final, production executions of a program should always be done in batch mode; however, development activities can sometimes be easier within the SAS Environment. With the exception of resetting line numbers in the SAS log, the SAS Environment can be scrubbed nearly as clean as that of a batch session. In addition to using data from the SASHELP library, temporary objects stored in the WORK data library need to be sanitized. Since much of the information to be deleted is useful for post-hoc review during the development process, I actually execute my "cleaning" code at the beginning of each program. The major areas to address are as follows:

- Delete temporary data sets in the WORK library;
- Delete temporary format catalog in the WORK library;
- Delete compiled macros stored in the WORK library;
- Clear all external file references;
- Clear all global macro variable values; and
- Clear titles and footnotes.

For the sake of brevity, I will omit a discussion of objects and options related SAS Graph. However, some of the same principles utilized below can be applied to SAS Graph-related items.

The macro name "CleanUp" is utilized throughout the examples below. While each example is displayed as its own distinct block of code, all of the blocks would ideally be parts of one single pre-processing macro. The ellipses in each of the code blocks represent other cleaning code which would precede or follow the example at hand.

Most users are familiar with the process to delete temporary data sets from the WORK library. Unfortunately, when there are no data sets in the WORK library, this code will return a warning that there are no matching members in the directory. To eliminate the possibility of this error, you can use information from SASHELP.VMEMBER to check for the presence of data sets in the WORK library and then selectively process the PROC DATASETS code as necessary:

```
%macro CleanUp;

  ...

  %local _workds;

  /*** Initialize value of _WORKDS to %str(NONE) ***/
  %let _workds = %str(NONE);


  /***
    Query SASHELP.VMEMBER to see if WORK data library contains
    any temporary data sets.
   ***/
  proc sql;
    select distinct memtype into :_workds
      from sashelp.vmember
      where libname eq 'WORK' and memtype eq 'DATA'
    ;

  quit;

  %put %str(After PROC SQL query of VMEMBER, _WORKDS is &_workds);


  /***
    If there were data sets present in the WORK library the value of _WORKDS
    would be %str(DATA) rather than %str(NONE) as set above.  The PROC DATASETS
    code will execute only when data sets are present in WORK.
   ***/
```

```
  %if (&_workds eq %str(DATA)) %then
    %do;
      proc datasets library=work memtype=data kill;

      quit;

      run;
    %end;

  ...

%mend CleanUp;
```

A similar, more targeted approach is used to delete the temporary format catalog if it exists.

```
%macro CleanUp;

  ...

  %local _formcat;

  /*** Initialize value of _FORMCAT to %str(NONE) ***/
  %let _formcat = %str(NONE);


  /***
    Query SASHELP.VMEMBER to see if WORK data library contains
    a catalog named FORMATS.
   ***/
  proc sql;
    select distinct memtype into :_formcat
      from sashelp.vmember
      where libname eq 'WORK' and memtype eq 'CATALOG' and memname eq 'FORMATS'
    ;

  quit;

  %put %str(After PROC SQL query of VMEMBER, _FORMCAT is &_formcat);


  /***
    If there were data sets present in the WORK library the value of _FORMCAT
    would be %str(CATALOG) rather than %str(NONE) as set above.  The PROC DATASETS
    code will execute only when WORK data library contains a catalog named FORMATS.
   ***/
  %if (&_formcat eq %str(CATALOG)) %then
    %do;
      proc datasets library=work memtype=catalog;
        delete formats;

      quit;

      run;
    %end;

  ...

%mend CleanUp;
```

Unfortunately, it's not quite as easy to get rid of the catalog in the WORK library which contains compiled macros, WORK.SASMACR. In SAS version 6.12 it was possible to delete this catalog as we did with the temporary format catalog above; however, in subsequent versions of SAS it's not possible to delete the catalog once it's been created.

It is possible to remove all of the catalog entries in WORK.SASMACR, and even selectively delete or keep each one based on a user-defined white list. For the purpose of this example, we will assume that the CleanUp macro was %included from an external file rather than executed from a permanent compiled macro. Once %included, the compiled CleanUp macro will be stored in WORK.SASMACR, so we need not check for the existence of the catalog. Since we are using CleanUp to perform our cleaning activities, however, we must exclude CleanUp from being deleted.

```
%macro CleanUp (_cmsave=%str(OtherMac.macro));

  ...

  %local _current _numobs _i;

  /*** Initialize the value of _NUMOBS to zero ***/
  %let _numobs = %eval(0);


  /***
    Extract the catalog entries from the compiled macro catalog in the
    WORK data library into the data set _SASMACR.
   ***/
  proc catalog catalog=work.sasmacr;
    contents out=_sasmacr;

  quit;

  run;


  /***
    Count the number of compiled macros to be deleted, excluding CleanUp
    and any macro names specified in the white list parameter, _CMSAVE.
   ***/
  data _sasmacr;
    set _sasmacr
      (keep=name type
       where=(left(type) eq 'MACRO' and
              not((name eq 'CLEANUP' and type eq 'MACRO') or
                  index(upcase("&_cmsave."),
                        trim(left(name)) || '.' || trim(left(type))
                       )
                  )
             )
      ) end=eof
    ;

    length _index  8;

    _index = _n_;

    if (eof) then call symput('_numobs', trim(left(put(_index, best.))));

  run;
```

```
  /*** Execute this block if we have at least one compiled macro to delete ***/
  %if (&_numobs ge 1) %then
    %do _i = 1 %to &_numobs;

      /*** Initialize current to null ***/
      %let _current = ;


      /***
        Successively select a macro name from the list of compiled macros targeted
        for deletion and put that name into the macro variable _CURRENT.
       ***/
      proc sql noprint;
        select upcase(trim(left(name)) || '.' || trim(left(type))) into :_current
          from work._sasmacr
          where _index eq &_i
        ;

      quit;

      run;


      /*** Delete the current compiled macro entry targeted for deletion ***/
      proc catalog catalog=work.sasmacr;
        delete &_current.;

      quit;

      run;
    %end;

  /*** Delete the temporary data set _SASMACR ***/
  proc datasets library=work memtype=data nolist;
    delete _sasmacr;

  run;

  quit;

  ...

%mend CleanUp;
```

Macro variables cannot be deleted, so to clean our workspace we must instead remove the values from each of the macro variables that are not on our white list, which is passed in through the parameter _MVSAVE. Specifically targeting only the global macro variables ensures that we avoid the SAS system macro variables. As noted above with regard to the structure of the VMACRO data view, a single macro variable may actually be represented by multiple rows in VMACRO. A step could be added to the code below in order to generate a data set containing a distinct list of global macro variable names. The data set generated by this additional step would be the input to the data step below in which all of the targeted macro variable values are set to null via CALL SYMPUT.

```
%macro CleanUp (_mvsave=%str(_saveone, _savetwo));

  ...

  /***
    Set values for all global macro variables to null, excluding those
    in the white list parameter, _MVSAVE.
   ***/
```

```
  data _null_;
    set sashelp.vmacro
      (keep=scope value name
       where=(scope eq 'GLOBAL' and left(value) ne ' ' and
              not(index(upcase(trim(left("&_mvsave"))), upcase(trim(left(name)))))
             )
      )
    ;
      by scope name;

    if (last.name) then
      do;
        put "The value of macro variable " name "will be cleared";

        call symput(left(name), '');
      end;

  run;

  ...

%mend CleanUp;
```

External file references should be cleared by calling the FILENAME statement, the name of the reference, and the statement option CLEAR. The names of external file references are contained in the SASHELP table VEXTFL. File reference names which begin with the hash, or pound sign, character "#" are SAS-defined file references and should not be cleared.

```
%macro CleanUp;

  ...

  %local _current _numobs _i;

  /*** Initialize the value of _NUMOBS to zero ***/
  %let _numobs = %eval(0);


  /***
    Count the number of external file references to be cleared, excluding any
    SAS-defined references denoted by the leading # character in the file
    reference name.
   ***/
  data _extfl;
    set sashelp.vextfl
      (keep=fileref
       where=(substr(left(fileref), 1, 1) ne '#')
      ) end=eof
    ;

    length _index  8;

    _index = _n_;

    if (eof) then call symput('_numobs', trim(left(put(_index, best.))));

  run;

  %if (&_numobs ge 1) %then
    %do _i = 1 %to &_numobs;
```

```
      /*** Initialize current to null ***/
      %let _current = ;


      /***
        Successively select a file reference name from the list of
        file references targeted to clear and put that name into the
        macro variable _CURRENT.
       ***/
      proc sql noprint;
        select fileref into :_current
          from work._extfl
          where _index eq &_i
        ;

      quit;

      run;


      /*** Clear the current file reference ***/
      filename &_current. clear;

    %end;

  /*** Delete the temporary data set _EXTFL ***/
  proc datasets library=work memtype=data nolist;
    delete _extfl;

  run;

  quit;

  ...

%mend CleanUp;
```

Finally, we should clear the assigned titles and footnotes by simply calling the following statements with no additional arguments:

```
title;

footnote;
```

CONCLUSION

The native data views in the SASHELP data library are a powerful repository of information which you can use to have a more thorough understanding and greater control of your SAS programming environment. You can use those views to help you to build tools, automate tasks, and perform quality checks of your data.

REFERENCES

SAS Institute, Inc. 2002-2008. "SAS OnlineDoc® 9.1.3." http://support.sas.com/onlinedoc/913/docMainpage.jsp

SAS Institute, Inc. 1999. "SAS OnlineDoc®, Version 8." http://v8doc.sas.com/sashtml/

Merriam-Webster, Incorporated 2009. "Merriam-Webster Online Dictionary" http://www.merriam-webster.com/dictionary/metadata

ACKNOWLEDGMENTS

I would like to thank David Carr, Stephen Hunt, Jackie Lane, Tony Pisegna, and Syamala Ponnapalli for their help, guidance, and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author:

William Turner
ICON Clinical Research
1700 Pennbrook Pkwy
North Wales, PA 19454

Work Phone: (215) 616-4812
Fax: (215) 616-8685
E-mail: william.turner@iconplc.com
Web: www.iconplc.com