You Could Look It Up:
An Introduction to SASHELP Dictionary Views

Michael Davis, Bassett Consulting Services, North Haven, Connecticut

ABSTRACT

Ever wonder what titles were already set in a batch SAS® session? Need a list of the members in a library so your macro can automatically hack at each one? Curious as to how many observations are in that data set without running a procedure or DATA step? Want to see what macro variables already exist? Anytime that one has a question about what is going on in their SAS session, they can answer it the same way that SAS itself does the task... they can look it up in the SASHELP dictionary views.

This presentation is an introduction to all those views automatically created by the SAS System and found in the SASHELP libref and prefixed with a "V". The dictionary views track items and their attributes for features such as catalogs, data set variables (columns), external files, indices, macro variables, data set members, titles, portable system options, and of course, views. The focus of the presentation is to illustrate how the dictionary views can be used in common situations. Last, for the SQL-phobic, it will be shown how these views can be accessed through Display Manager and through DATA step programming.

WHY LEARN ABOUT THIS SUBJECT?

One might ask, "Why should I learn about the SASHELP Dictionary Views?" This is a fair question since even some experienced SAS users have not heard about the SASHELP dictionary views. The one sentence answer is that the SASHELP dictionary views offer an easy way to monitor the activities and status associated with a SAS session.

For the more adventurous, SASHELP dictionary views often serve as building blocks for automated applications. SAS developers can use these views to communicate the SAS environment to their applications. This can reduce or eliminate the need to edit batch jobs before each run. These self-modifying jobs then can then be launched by a scheduler, further lessening programming drudgery.

However, less experienced SAS users should not be intimidated by this feature of SAS. Nearly any SAS programmer can benefit from using some of these techniques.

WHAT ARE THE DICTIONARY VIEWS?

The dictionary views are tables that are automatically available when a SAS session is started. They show the state of the session at the time that they are requested. These views can be used as you would use any read-only table.

Here is a summary of the types of information listed in the dictionary views:

- libraries, catalogs, and data sets
- external files allocated to the session
- macro, data set variables attributes
- indexes, titles, footnotes, and views
- system option settings

Dictionary tables are a standard feature of nearly all relational database management systems (DBMS). Other DBMS products such as Oracle® and DB2® also feature dictionary tables, accessed via Structured Query Language (SQL). The dictionary tables (*not* the SASHELP dictionary views) are not available outside the SQL procedure.

The dictionary tables available through PROC SQL include:

- CATALOGS
- COLUMNS
- EXTFILES
- INDEXES
- MEMBERS
- OPTIONS
- TABLES
- VIEWS

SASHELP VIEWS vs. DICTIONARY TABLES

The SASHELP dictionary views replicate the information stored in the dictionary tables. The most important difference is that they can be used outside of the SQL procedure. Thus any DATA step that a SAS programmer may imagine can take advantage of these views.

Another useful feature of the SASHELP dictionary views is that they can be opened from the ACCESS window in a Display Manager session. Thus, during an interactive session, users can interrogate the

appropriate SASHELP view to get the status and name information they seek.

Since the SASHELP views and the dictionary tables yield similar information, the question is often asked, "Which one should I use?". If one needs to obtain session status information outside of the SQL procedure, especially if one likes to "point and click" to the answer, then the SASHELP views will probably be the best choice.

On the other hand, when one uses a WHERE clause on a SASHELP view, every row must be generated before the clause can be applied. If the WHERE clause is applied instead to the analogous dictionary table, only the rows applicable need be generated. Hence accessing the dictionary tables might may be more computer-efficient (faster).

When querying SASHELP.VCOLUMN, be sure to apply a subsetting WHERE clause to speed filtering. While this paper features both SASHELP view and dictionary table examples, the author prefers the SASHELP views because they are easily accessed during a SAS session and also because he prefers DATA steps over the SQL procedure.

TOURING THE SASHELP VIEWS

For Version 6, the SASHELP views are documented in *SAS Technical Report P-222* on pages 290-91. In the Version 8 OnlineDoc, both the SASHELP views and the dictionary tables are covered in the *SAS Procedures Guide,* Chapter 34 under the SQL Procedure.

In both Technical Report P-222 and the OnlineDoc, the PROC SQL syntax used to create each SASHELP view is shown. Since all of the SASHELP dictionary views are prefixed with by "V", the following program can be used to print a listing of the available SASHELP views:

```
proc print data=sashelp.vsview ;
  where libname = 'SASHELP' and
  substr(memname, 1, 1) = "V"
run ;
```

LIBREF AND FILEREF VIEWS

The VSLIB view shows allocated library names and physical path name information. The VEXTFL view shows the allocated filerefs, the physical path name, and the engine (if any) associated with each fileref.

Please note that filerefs allocated by the SAS System will be prefixed by an underscore. Also, note the following trap. SASHELP views that show directory path names only show the first 80 characters of the physical path name. The actual path name may be longer than 80 characters!

TABLE AND SUMMARY TABLE VIEWS

The VMEMBER view lists data sets, views, and catalogs. If one needs the physical directory path name, they should consider the VMEMBER view to obtain this information. If other details are required, such as observation length, number of observations, data set labels, creation and modification dates, then the VTABLE view can be consulted. The VVIEW view lists only views and the engine associated with them.

The VSTABVW lists only the library and member name for data sets and views. The VTABLE view lists library and member names for data sets only. Its counterpart is the VSVIEW view, which shows both automatic and user-created views. The VSACCESS view shows only user-created views.

VARIABLE VIEWS

The VCOLUMN view shows data set information at the variable level. This view includes details such as variable type, length, position, format, label, and indexes. This is often a large table and should almost always be subsetted by a WHERE clause before using.

The VMACRO view shows details about macro variables such as scope and current values. Please note that this view will show the value of global and macro variables. However, the VMACRO view omits local macro variables, which exist only within the scope of macros.

CATALOG AND INDEX VIEWS

The VCATALOG view shows detailed information about catalogs and their members. The VSCATALG view just lists the catalogs and member names.

The VINDEX view contains detailed information about indexes that allow one to trace data set variables to the indexes in which they are used. You can also use the VINDEX view to identify the indexes available for each data set.

TITLE AND FOOTNOTE VIEW

Both information about titles and footnotes are contained in the VTITLE view. One can distinguish between titles and footnotes in this view by inspecting the title location. It will contain a “T” for titles, “F” for footnotes.

SYSTEM OPTION VIEW

The VOPTIONS view shows the status of SAS system options. It will show portable options that do not apply to a platform, such as NODMS, FSDEVICE, OPLIST, and TAPECLOSE under Windows. However, it omits system specific options, such as WINCHARSET under Windows. Please keep in mind that the information obtained from the VOPTIONS view might vary among SAS releases.

The next section of this paper will offer some examples of how the SASHELP views can be used in real-life situations to solve problems

CROSS-TABULATE VARIABLES BY DATA SETS

When one inherits a data library with many members, it can help to have a cross-table that shows in which members a variable appears. Likely applications for this type of information might be pharmaceutical trials and data warehousing projects. Another application would be to determine which SASHELP views might be consulted.

Eric Losby presented a paper on this subject at SUGI 22. RDB_MAP.SAS program uses the VCOLUMN view coupled with the TRANSPOSE procedure to produce listings similar to the following example.

Sample RDB_MAP.SAS Listing

Relational Database Map of MYLIB

OBS	NAME	SAVEIT	SPID2DSN	PRGM_NO
1	AUTHFILE	X	X	
2	CLIENT			X
3	CLNT_CD			
4	CLNT_ID	X	X	
5	CLNT_NM	X	X	
6	CLNT_NO			X
7	CRLNFILE	X	X	
8	CUSHION			

RDB_MAP.SAS

```
options ps=80 nodate ;

%let libref = ACCESS ;

libname &libref 'drive:[directory]' ;

title "Relational Database Map of
&libref." ;

* SECTION 1 ;
data temp1(keep=memname name x) ;
   set SASHELP.VCOLUMN ;
   if libname="&libref." and
memtype='VIEW' then do ;
      x='X' ;
      output ;
   end ;
   else delete ;
run ;

* SECTION 2 ;
proc sort data=temp1 ;
   by name memname ;
run ;

* SECTION 3 ;
proc transpose data=temp1
out=list(drop=_name_) ;
   id memname ;
   var X ;
   by name ;
run ;

* SECTION 4 ;
proc print data=list ;
run ;
```

The MEMTYPE variable can be used to restrict the members shown in the “Relational Database Map” to either data sets or views.

PRINTING NOBS, SAMPLE OBSERVATIONS

Sometimes it is useful to have a listing of the members in a SAS library showing the number of observations in each member. A listing of a few sample observations for each data set can be helpful, too.

If a library has many members, writing a program to generate the NOBS listing and sample observations can take some effort. Fortunately, Janet Stuelpner and Elizabeth Kaptsanov presented a macro to do this at SUGI 22. The %PRINTIT macro and its two component macros, %PRT and %PRTDS appear on the following page along with some sample listings.

PRINTIT.SAS

```
/* Specify                          */
/* libref path, temporary file to */
/* store individual macro calls   */
/* and number of observations to  */
/* be printed (default is MAX).   */

%macro printit(path=, outfile=,
prtobs=MAX) ;

/* Part I:  to gather member names from
database */

* define database and output file ;
libname mylib "&path" ;
filename outfile "&outfile" ;
options nodate ;

* get member names and number of
observations ;
proc sql ;
   create table test1 as
   select libname, memname, nobs
   from dictionary.tables
   where libname='MYLIB' ;
quit ;

* print member names and number of
observations ;
proc print data=test1 ;
   title1 "DATA SETS FOR &path" ;
run ;

/* Part II: Print data from each data set
*/

%macro prt(memname, nobs) ;

options pageno=1 ps=60 obs=&prtobs ;

proc print data=mylib.&memname ;
   title1 "DATA SET=&memname NOBS=&nobs" ;
   footnote "RUN DATE:  &sysdate" ;
run ;

%mend prt ;

%macro prtds ;

options obs=max ;

data _null_ ;
   set test1 ;
   file "&outfile" ;
   put '%prt('MEMNAME $8.','NOBS 8.')' ;
run ;

%include "&outfile" ;

%mend prtds ;

%prtds ;

*to reset OBS parameter ;
options obs=max ;

%mend printit ;
```

%PRINTIT NOBS Listing

DATA SETS FOR C:\MYLIB

OBS	NAME	MEMNAME	NOBS
1	MYLIB	CLIENT	11
2	MYLIB	PRGM_NO	343
3	MYLIB	RPTPARMS	1
4	MYLIB	RSIPSYMB	60
5	MYLIB	SAVEIT	166

%PRINTIT Sample Listing

DATA SET=MYDATA NOBS=296

OBS	GROUP	CODEVAL
1	004	03657
2	005	03657
3	006	03657
4	007	03657
5	008	00080

RUN DATE: 1JUL00

The %PRINTIT macro uses the SQL procedure to read the TABLES dictionary table to generate the NOBS listing. The submacro %PRT prints the NOBS table shown above. The submacro %PRTDS prints the sample listing for each data set, also shown above. The %PRTDS macro calls are written to the file denoted by the macro variable OUTFILE, which is subsequently executed by a %INCLUDE.

RESTORING MACRO VARIABLES

To replicate program results where macro variables are involved, it is necessary to save the “state” of the original SAS session and restore that status to a subsequent session. John Gerlach presented a program that does this potentially tedious task at NESUG 97. His %PARAMS macro is shown on the following page.

%PARAMS uses the SASHELP VMACRO view to perform this wizardry. The macro parameters composing the session state are saved to a data set so that the state can be restored in a subsequent session. The restored macro variables are listed in the SAS Log. Macro aficionados will want to inspect the consecutive ampersands used in the %PARAMS macro.

PARAMS.SAS

```
* this data step gets the macro variables ;
data [project libref].params ;
   set sashelp.vmacro(where=(scope eq 'GLOBAL'
      and name not like 'SQL%' and name
      not like '%EXIST')) ;
   keep name value ;
run ;

* %PARAMS reinstitutes macro variables ;

%macro params ;
   data _null_ ;
      set [project libref].params end=eof ;
      call symput(name, value) ;
      call symput('var' ||
        trim(left(put(_n_,8.))),name) ;
      if eof then call symput
         ('nmvars',trim(left(put(_n_,8.)))) ;
   run ;
   %put Parameters: ;
   %do i = 1 %to &nmvars. ;
      %put &&var&i..   : &&&&&&var&i.. ;
      %end ;
   %put ;
%mend params ;
```

WRITING OUT FLATFILES

Before the introduction of the Export Wizard, writing all the variables of a complex data set to a flat file was a daunting task. Fortunately, Ian Whitlock presented the %FLATFILE macro at SUGI 19 and Michelle Buchecker presented it again at SUGI 21. There often are situations where %FLATFILE is still useful, especially when the flat file needs to be customized.

The program can be downloaded from the SAS Institute web site at the following URL:

http://ftp.sas.com/pub/sugi21/paper.027/flatfile.sas

This version of %FLATFILE appears in the right column on this page. It uses the SQL procedure and the COLUMNS dictionary table to identify all of the variables and their attributes.

WRITING DATA TO MS EXCEL

One valuable variation of the %FLATFILE macro is a version that will write out a comma-delimited version to be imported into spreadsheets such as Microsoft Excel and other desktop programs. Dave Mabey presented such a variation at NESUG 97, %FLATFILC, which appears on the subsequent page.

%FLATFILE

```
options mprint;
%macro flatfile(lib=,dsn=,file=);
  %let lib=%upcase(&lib);  /* uppercase
library and dataset names */
  %let dsn=%upcase(&dsn);

 proc sql;
   create view temp as
     select name, type, format, length
     from dictionary.columns
     where libname = "&lib" and memname =
"&dsn";
 quit;

 data _null_;
   set temp end=last;
   call symput
('var'!!left(put(_n_,3.)),name);
   if format ne ' ' then
      call symput
('fmt'!!left(put(_n_,3.)),format);
   else
     if upcase(type) = 'CHAR' then
       call symput
('fmt'!!left(put(_n_,3.)),'$'!!put(length,3.
)!!'.');
     else
       call symput
('fmt'!!left(put(_n_,3.)),'best10.');

  if last then call
symput('numvar',left(put(_n_,3.)));

   data _null_;
     set &lib..&dsn;
     file "&file";
     put
     %do i = 1 %to &numvar;
       &&var&i &&fmt&i +1
     %end;
     ;   /* end put statement */
   run;
 %mend;

 %flatfile(lib=sasuser, dsn=houses,
file=flat.dat)
```

%FLATFILC is similar to %FLATFILE except that variable names appear in the first row, variable labels appear in the second row, and columns are tab delimited.

DELETING MANY VARIABLES

Sometimes one inherits a data set and discovers that it has several variables that are always populated with zeros, missing values, or blanks. To delete these variables by hand might prove tedious.

Two macros to remove such variables, written by Charles Patridge and Shiling Zhang, can be found on the SASCONSIG web site, http://www.sasconsig.com in the Tips and Techniques section as TI00122. The first macro uses the COLUMNS dictionary table and PROC SQL.

%FLATFILC

```
%macro flatfilc
   (lib=,  /* libref for input dataset  */
   dsn=,   /* memname for input dataset */
   file=); /* filename of output file   */

   %let lib=%upcase(&lib);
   %let dsn=%upcase(&dsn);
   proc sql noprint;
      select quote(name),
         quote(case when label ne ' ' then
label
            else name
          end),
      name
   || case when format ne ' ' then format
        when type='num' then 'Best10.'
        else "$"||put(length,z3.)||'.'
      end
   into :names separated by ' "09"x ',
      :labels separated by ' "09"x ',
      :string separated by ' "09"x '
   from dictionary.columns
   where libname = "&lib"
    and memname = "&dsn";
quit;

data _null_;
   set &lib..&dsn;
   file "&file";
   if _n_=1 then put &names / &labels;_
   put &string;
run;

%mend;
```

LIBRARY MEMBERS INTO MACRO VARIABLE

For some applications, it may be handy to have a macro variable that contains the members of a SAS library. This can be done with the MEMBERS dictionary table and a simple PROC SQL step. The following piece of code is also from the SASCONSIG web site and is TIP00112, written by Charles Patridge.

TIP00112

```
proc sql noprint;
   select distinct memname into
     :macvar separated by " "
       from dictionary.members
       where upcase(libname ) = "WORK" and
         upcase(memtype) = "DATA"
       order by memname
       ;
 quit;

 %put WORK data sets: &macvar.;
```

DOES A FORMAT EXIST?

Both in ad hoc situations and when developing automated applications, the need often emerges to determine if a particular user-created format exists in any of the available SAS catalogs. Chris Roper posted the following snippet of code on the SAS-L distribution list. It uses the CATALOGS dictionary table as shown below to test for a format named TEST:

TEST FOR EXISTENCE OF FORMAT

```
proc sql ;
   create table aa as select *
   from dictionary.catalogs
   where objtype contains 'FORMAT'
     and objname = 'TEST' ;
quit ;
```

SUBMIT BATCH FROM DISPLAY MANAGER

Ah, the choices we SAS programmers have to make. Submitting in batch allows one to launch a second SAS job without waiting for the first job to complete. However, if you work from within Display Manager, you gain access to those wonderful "DM" commands and SAS windows, which are just a click away. What should you do?

If one is working under UNIX, you might want to make use of an imaginative application of the EXTFILES dictionary table, the %DOBATCH macro. This macro was presented at SUGI 24 by Jingren Shi and Shiling Zhang.

Shi and Zhang point out that to submit a batch SAS program in UNIX using the %DOBATCH macro, the actual submission employs the following syntax.

```
call system("sas -sysin &_execpgm &");
x "sas -sysin &_execpgm &";
rc=system("sas -sysin &_execpgm &");
%sysexec sas -sysin &_execpgm &;
```

In order for the above UNIX syntax to work, the macro variable _EXECPGM needs to contain the full path and filename of the program to be executed. This information is available from the EXTFILES dictionary table after a Display Manager SAVE or SAVE AS command. This is done by the %DOBATCH macro shown on the next page.

%DOBATCH

```
%macro dobatch;
%local _execpgm;

/* clear pgm window */
*dm 'clear pgm';

/* getting file name information */
proc sql noprint;
reset inobs=1;
select xpath into: _execpgm
from dictionary.extfiles
quit;

/* executing a batch job */
data _null_;
call system ("sas -sysin &_execpgm &");
run;

%put &_execpgm is submitted as a batch job.;
%mend;
```

If this macro is included in the AUTOCALL library or the AUTOEXEC.SAS program, one avoids having to submit the above code in each session where one wishes to use %DOBATCH.

The last detail in implementing %DOBATCH is to program the "trigger" to submit the program to SAS. Perhaps the easiest way is to assign %DOBATCH to a function key as part of the AUTOEXEC.SAS program using the following statement:

```
(%keydef f12 submit '"%dobatch"';)
```

When this statement is submitted, the F12 function key, the DM command "submit '%dobatch' " is issued. Again this approach requires that the program be saved first in order to work. However, saving your program regularly is usually a good practice.

WHERE DID I SAVE THAT ENTRY?

The author fondly recalls one assignment where more than one person would check out a SAS catalog over the weekend. On Monday, we had to get together for a few minutes and identify what entries we changed. Those meetings would take a bit longer when more than one person touched a particular catalog entry.

While a source control system might have eliminated that forced need for consultation, that scenario and others often leads to questions such as, "What changed since my application stopped working?" and "What was the name of that catalog entry that I copied that entry to?".

Peter Crawford posted the following piece of code on SAS-L earlier this year that creates a catalog entry listing to solve those types of problems.

Create View of Catalog Entries in Saved Order

```
proc sql noprint;
 create view sasuser.vlatest as
   select libname, memname, objname, objtype,
          objdesc format=$23. ,
          input( modified, mmddyy8. )
            format=date7. as mod
     from dictionary.catalogs
    where libname ne 'SASHELP'
        & libname ne 'MAPS'
   order by  mod desc
;
quit;
```

This code uses the CATALOGS dictionary table. It omits changes made in the SASHELP and MAPS libraries. However, one can alter the WHERE clause to change the libraries included in the listing to better accommodate your requirements.

The view SASUSER.VLATEST can be inspected from the ACCESS window in Display Manager or by using another technique. The beauty of this example is that it illustrates that useful information can be generated with only a few lines of code and the SASHELP views or dictionary tables.

TIPS TO EXPLOIT DICTIONARY VIEWS

In his presentation at the 1998 Western Users of SAS Software conference, Jack Hamilton offered some general tips when using the dictionary tables when creating one's own utilities. Some these tips are employed in the programs that have been reviewed.

First, use the INTO and SEPARATED BY clauses in the SQL procedure to create macro variables. Use CALL EXECUTE in a data step to loop through the values extracted from a dictionary table.

Second, use a FILENAME pointed to a catalog entry to save the extracted parameters as SAS code. Then use %INCLUDE to submit the statements in the catalog entry. Using a catalog entry is better than writing to a temporary disk file since the catalog technique is platform independent.

In that paper, Jack offered a piece of code that puts the variables in a data set into alphabetic order. Suppose that we have a data set as created by the code shown on the next page.

```
data one ;
   key= '9' ; a=1 ; c=2 ; b= . ;
   output
run ;
```

The above code will yield a data set named ONE with four variables in the order in which they appeared after the DATA statement. The code offered to write a new data set with the variables in alphabetic order is:

```
proc sql noprint ;
   select name into :newcmd Separated by ','
   from dictionary.columns
   where libname='WORK' and memname='ONE'
   order by name ;

   create table two as
   select &newcmd
   from one ;
```

The above code creates a new data set, TWO, with the variables appearing in alphabetical order.

CAUTION!

In the same presentation, Jack Hamilton offered some wise cautions. In both his paper and this one, the examples do not show error-checking. It is always a good idea to program checking to handle situations where the expected data is missing or parameters are not received.

For example, make sure that any utilities that are created work correctly if no rows or observations are selected by WHERE clauses. Another good caution is to blank output macros before the actual code starts.

Finally, the COLUMNS table often gets extremely large, especially when the SAS/GRAPH® map data sets are allocated. Apply a subsetting WHERE clause in the code at the earliest point possible.

CONCLUSION

The author hopes that the examples and other materials in the paper shed some light on the scope of information available in the SASHELP dictionary views. He hopes that his audience will be inspired by these examples and will start to use them and the SASHELP dictionary views to make their lives easier.

At the end of this paper is an appendix that lists all of the current SASHELP dictionary views alphabetically in an abridged contents listing format.

The SASHELP dictionary views are not difficult to apply. So please go forth and "look it up".

BIBLIOGRAPHY

Buchecker, M. Michelle, "%FLATFILE and Make Your Life Easier," *Proceedings of the Twenty-First Annual SAS Users Group International Conference,* 21, 178-80

Gerlach, John R., "The Six Ampersand Solution," *Proceedings of the 1997 Northeast SAS User Group Conference,* 1997. 629-30

_____, "A Cross-reference for SAS Data Libraries," *Proceedings of the 1999 SouthEast SAS User Group Conference,* 1999. 217-20

Hamilton, Jack, "Some Utility Applications Of The Dictiionary Tables in PROC SQL," *Proceedings of the 1998 Western Users of SAS Software Conference*, 1998. 85-90

Losby, Eric (1997), "How Are All of These Tables Related? - Relational Database Map - RDB_MAP.SAS," *Proceedings of the Twenty-Second Annual SAS Users Group International Conference,* 22, 1145-1149

Mabey, David A., "Eliminating Tedium by Building Applications That Use SQL Generated SAS Code," *Proceedings of the 1997 Northeast SAS User Group Conference,* 1997. 755-60

SAS Institute Inc., SAS® Technical Report P-222 *Changes and Enhancements to Base SAS Software, Release 6.07*, Cary, NC: SAS Institute Inc., 1991. 290-91

SAS Institute Inc., *SAS© Procedures Guide, Version 8*, Cary, NC: SAS Institute Inc., 2000. 1062-6

Shi, Jiang, and Zhang, Shiling, "Submitting a Batch SAS Job within the Display Manager Mode under Unix," *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference,* 24, 1458-9

Stuelpner, Janet E. and Kaptsanov, Elizabeth (1997), "All the Data That Fits, We Print," *Proceedings of the Twenty-Second Annual SAS Users Group International Conference,* 22, 474-6

Whitlock, H. Ian (1996), "How to Write A Macro to Make External Flat Files," *Proceedings of the Twenty-First Annual SAS Users Group International Conference,* 21, 163-71

ACKNOWLEDGEMENTS

SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc. Acrobat and PostScript are registered trademarks of Adobe Systems Inc. Microsoft and Microsoft Excel are registered trademarks of the Microsoft Corporation. DB2 is a registered trademark of the IBM Corporation. Oracle is a registered trademark of Oracle Corporation.

The author would like to thank the Hartford Area SAS User Group Steering Committee, which encouraged him to prepare this paper. Special thanks are offered to Robert Krajcik, Jack Hamilton, Charles Patridge, Clinton Rickards, Ian Whitlock, and Michael Zdeb.

CONTACT INFORMATION

The author may be contacted as follows:

Michael L. Davis
Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
E-Mail: michael@bassettconsulting.com
Web: http://www.bassettconsulting.com
Telephone: (203) 562-0640
Facsimile: (203) 498-1414

Please note that the code supplied in this paper is designed only to illustrate the concepts being discussed and will need to be modified to work in other applications. Modified code is not supported by the author of this paper.

APPENDIX – SASHELP VIEWS CONTENTS

SASHELP.VCATALG

Variable	*Type*	*Len*	*Label*
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
OBJNAME	Char	8	Object Name
OBJTYPE	Char	8	Object Type
OBJDESC	Char	40	Object Desc.
MODIFIED	Char	8	Date Modified
ALIAS	Char	8	Object Alias

SASHELP.VCOLUMN

Variable	*Type*	*Len*	*Label*
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
NAME	Char	8	Column Name
TYPE	Char	4	Column Type
LENGTH	Num	8	Column Len.
NPOS	Num	8	Column Pos.
VARNUM	Num	8	Col # in Tbl
LABEL	Char	40	Col Label
FORMAT	Char	16	Col Format
INFORMAT	Char	16	Col Infmt
IDXUSAGE	Char	9	Col Idx Type

SASHELP.VEXTFL

Variable	*Type*	*Len*	*Label*
FILEREF	Char	8	Fileref
XPATH	Char	80	Path Name
XENGINE	Char	8	Engine Name

Note: It is possible to have a path longer than 80 characters!

SASHELP.VINDEX

Variable	*Type*	*Len*	*Label*
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
NAME	Char	8	Column Name
IDXUSAGE	Char	9	Column Index Type
INDXNAME	Char	8	Index Name
INDXPOS	Num	8	Pos of Col in Concatenated Key
NOMISS	Char	3	Nomiss Option
UNIQUE	Char	3	Unique Option

SASHELP.VMACRO

Variable	Type	Len	Label
SCOPE	Char	9	Macro Scope
NAME	Char	8	Macro Var. Name
OFFSET	Var	8	Offset into Var
VALUE	Char	200	Macro Var Value

SASHELP.VMEMBER

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
ENGINE	Char	8	Engine Name
INDEX	Char	8	Indexes
PATH	Char	80	Path Name

includes data sets, views, catalogs

SASHELP.VOPTION

Variable	Type	Len	Label
OPTNAME	Char	16	Session Option Name
SETTING	Char	200	Session Option Setting
OPTDESC	Char	80	Option Description

SASHELP.VSACCESS

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name

SASHELP.VSCATLG

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name

SASHELP.VSLIB

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
PATH	Char	80	Path Name

Note: It is possible to have a path longer than 80 characters!

SASHELP.VSTABLE

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name

SASHELP.VSTABVW

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type

SASHELP.VSVIEW

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name

SASHELP.VTABLE

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
MEMLABEL	Char	40	Dataset Label
TYPEMEM	Char	8	Dataset Type
CRDATE	Num	8	Date Created
MODATE	Num	8	Date Modified
NOBS	Num	8	Number of Obs
OBSLEN	Num	8	Obs Length
NVAR	Num	8	Number of Vars
PROTECT	Char	3	Type Password Protect
COMPRESS	Char	8	Compress Routine
REUSE	Char	3	Reuse Space
BUFSIZE	Num	8	Bufsize
DELOBS	Num	8	No of Deleted Obs
INDXTYPE	Char	9	Type of Indexes

SASHELP.VTITLE

Variable	Type	Len	Label
TYPE	Char	1	Title Location
NUMBER	Num	8	Title Number
TEXT	Char	200	Title Text

SASHELP.VVIEW

Variable	Type	Len	Label
LIBNAME	Char	8	Library Name
MEMNAME	Char	8	Member Name
MEMTYPE	Char	8	Member Type
ENGINE	Char	8	Engine Name