# Data About Data – A Look at Dictionary Tables & SASHELP Views

A component was added in SAS Version 6.07 called dictionary tables. This column will introduce you to these interesting and useful objects in the SAS system. Dictionary tables are special read-only PROC SQL objects that contain systems catalogs for all SAS datasets and other SAS files. Some of this information is available from PROC CONTENTS, PROC DATASETS, and other sources, but dictionary tables are more readily available and are usually easier to access. Using Dictionary Tables you might use a dictionary table when you want to...

• know if a dataset exists, how many variables it contains, and how many observations it contains

• check the current value of an option or title, change it, but then reset it back to the original value when you are finished

• determine if a macro variable exists

• capture the names of variables, as well as the attributes, from a SAS data file

Dictionary tables are generated at run time to retrieve information about SAS datasets, libraries, SAS catalogs, SAS macros, SAS titles, SAS options, and external files known to SAS. By using PROC SQL and the CREATE TABLE or CREATE VIEW clauses, they can be accessed as any other SAS dataset would be, to provide your programs with the information stored in the dictionary tables. Because they are read-only, you cannot alter dictionary table values directly. A partial list of the dictionary tables is shown below:

DICTIONARY.MEMBERS SAS files, catalogs
DICTIONARY.TABLES SAS datafiles
DICTIONARY.COLUMNS SAS vars
DICTIONARY.CATALOGS catalogs and entries
DICTIONARY.EXTFILES external files
DICTIONARY.VIEWS SAS views
DICTIONARY.INDEXES indexing info
DICTIONARY.OPTIONS SAS options
DICTIONARY.MACROS SAS macros vars
DICTIONARY.TITLES SAS Titles

For a more complete listing, reference the SAS documentation or use the PROC SQL DESCRIBE TABLE statement.

**Using the DESCRIBE TABLE statement**

To list the names of the columns stored in the DICTIONARY.MACROS entry, the following program can be run.

```
proc sql;
  describe table dictionary.macros;
quit;
```

The resulting Log:

```
 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'
   );
```

To print the data values of the MACROS entry, the PROC SQL CREATE table can make a SAS filed which contains an extract of the MACRO entry at that time. The extract file is a normal SAS dataset that can be processed like any SAS dataset.

```
proc sql;
 create table work.macros as
 select * from dictionary.macros;
quit;
proc print data=work.macros;
Title 'Work.macros';
run;
```

Partial Output:

Work.macros

| Obs | scope | name | offset | value |
|---|---|---|---|---|
| 1 | GLOBAL | SQLOBS | 0 | 0 |
| 2 | GLOBAL | SQLOOPS | 0 | 0 |
| 3 | AUTOMATIC | SYSDAY | 0 | Thursday |

```
4           AUTOMATIC  SYSTIME      0                09:14

.             .            .             .                .
```

Although using the PROC SQL code is probably the fastest way to access dictionary table data, SAS software comes standard with several views in the SASHELP library that you can access without PROC SQL. Some of the views are listed below:

| | |
|---|---|
| SASHELP.VMEMBER | SAS files and catalogs |
| SASHELP.VTABLE | SAS datafiles |
| SASHELP.VCOLUMN | SAS vars in all datasets |
| SASHELP.VCATALG | SAS catalogs and entries |
| SASHELP.VEXTFL | external files allocated |
| SASHELP.VVIEW | SAS views |
| SASHELP.VINDEX | SAS indexing information |
| SASHELP.VOPTION | SAS options |
| SASHELP.VSLIB | Sorted libname list |
| SASHELP.VSTABLE | Sorted table list |
| SASHELP.VMACRO | SAS macros |
| SASHELP.VTITLE | SAS Titles |

To see what is stored in each of the views, we could use a separate PROC CONTENTS and/or PROC PRINT, or we could use the SASHELP.VVIEW member to get the member names of all views that SAS knows of.

If we are interested in columns (variables) that are stored in the SASHELP views, the SASHELP.VCOLUMN view contains a row for each variable stored in all views known to SAS at that time.

**Examples Using SASHELP Views**

To show all of the columns in all of the SASHELP views, use the code that follows:

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

Partial Output:

| Obs | libname | memname | memtype | name | type... |
|---|---|---|---|---|---|
| 156 | SASHELP | VCATALG | VIEW | libname | char |
| 157 | SASHELP | VCATALG | VIEW | memname | char |
| 158 | SASHELP | VCATALG | VIEW | memtype | char |
| 159 | SASHELP | VCATALG | VIEW | objname | char |
| 160 | SASHELP | VCATALG | VIEW | objtype | char |
| 161 | SASHELP | VCATALG | VIEW | objdesc | char |
| 162 | SASHELP | VCATALG | VIEW | created | num |
| . | . | . | . | . | . |

To use the SASHELP.VTABLE view, again just reference it like you would any other SAS dataset.

If you want to list all datafiles in the SASDATA library with corresponding create dates and number of observations, while selecting only members created after 1 January, 2012, use the following code:

```
proc print data=sashelp.vtable;
 where crdate ge '01JAN2012:0:0'dt
       and libname='SASDATA' ;
  var memname crdate nobs;
run;
```

Output:

| OBS | MEMNAME | CRDATE | NOBS |
|---|---|---|---|
| 1 | ADDRDATA | 13JAN12:10:19:58 | 2 |
| 2 | BOTH | 13JAN12:10:20:07 | 1730 |
| 3 | COMPFILE | 13JAN12:10:19:52 | 1 |
| 4 | FILE1 | 13JAN12:10:19:59 | 1730 |

**Other Applications**

Let's look at an example where you are writing a macro that will accept a SAS library and member as the input dataset. If the dataset doesn't exist or has 0 rows or columns, you want to fail the process.

```
%MACRO SSCTEST(MLIB=,MMEM=);
```

```
%LET MSASDS=&MLIB..&MMEM;
PROC SQL;
  CREATE TABLE WORK.VCOLUMN AS
    SELECT NAME, TYPE, FORMAT, LENGTH, LABEL
    FROM DICTIONARY.COLUMNS
    WHERE LIBNAME=UPCASE("&MLIB")
          & MEMNAME=UPCASE("&MMEM");
DATA _NULL_;
  IF NOBS=0 THEN
    DO;
      FILE LOG;
      PUT "*** MSASDS=&MSASDS DOESNT EXIST OR
HAS 0 VARS**";
      PUT "*** MLIB=&MLIB MMEM=&MMEM **";
      STOP;
    END;
  SET WORK.VCOLUMN END=EOF NOBS=NOBS;
/*  rest of macro  */
RUN;
%MEND SSCTEST;
```

If the macro is called using a non-existent dataset or one with no variables, WORK.VCOLUMN will have no observations. The data step checks for the number of observations and will issue the message and stop running if there are no observations. If there are rows and columns in the passed dataset, the program above now has access to the other column attributes such as name, type, length, format, label, and more.

**Capturing a SAS Option Value**

Suppose you would like to capture the current linesize setting from SAS, then do a report with a different linesize, and when finished reset the linesize to the original value.

PROC SQL can query DICTIONARY.OPTIONS and can store the current linesize value into a macro variable called mlinesiz. The program can then change the linesize value, do the report, then by using the macro variable mlinesiz, set the option back to its original value.

```
proc sql noprint; /* capture setting     */
 select setting into :mlinesiz
  from dictionary.options
   where optname='LINESIZE';
   quit;
options linesize=132; /* set linesize to 132 */
proc print data=mydata;
run;
options linesize=&mlinesiz; /* set it back         */
```

This technique can be used to capture any SAS option setting. Similar logic could be used to store SAS titles and footnotes if necessary.

In summary, there are lots of ways to capture data about your data. Dictionary tables are often an appropriate and easy way to incorporate that information into your applications.