

Data About Data: An Introduction to Dictionary Tables

Frank C. Dilorio, ASG, Inc., Cary, NC
Nancy J. Michal, MEDCO Research, Inc., Research Triangle Park, NC

INTRODUCTION

Some of the most useful and least heralded additions to Version 6.07 of the SAS System are dictionary tables and views. They contain a wealth of information about SAS datasets, catalogs, system options, and external files. Some of the information was available in earlier versions of the SAS System via cumbersome procedure output datasets. Other information was simply not accessible. The dictionary tables and views (which we will simply refer to as "the tables") put this information at the disposal of the application developer in a straightforward manner. Best of all, the tables are automatically created and maintained by the SAS System, suggesting that there really may be such a thing as a free lunch (or "low cost lunch," as we note in the Performance section).

This paper: explains the organization of the tables, describes their content, presents suggestions for their use, and discusses several short applications of interest to SAS programmers. The reader should come away from the discussion with an understanding of the benefits of the dictionary tables as well as some ideas for how they may be used in their programming environments.

BACKGROUND

SAS programmers have always needed high-level information about the contents of SAS datasets and other entities. Consider just a few scenarios.

- You want to determine if a SAS dataset exists. Possible solutions include processing a CONTENTS output dataset or a _NULL_ data step with strategically placed SET and STOP statements and calls to SYMPUT. These solutions are unnecessarily awkward.
- A project uses numerous SAS datasets, some of them with like-named variables. You want to identify instances of like-named variables with conflicting data types or lengths. For example, the variable GENDER may be stored as a character, length 1 in one dataset and numeric, length 3 in another.
- A macro has to adjust various system options. You'd like to capture the current settings, reset the options, and when exiting the macro reset the options to their original values. This is, for all practical purposes, not possible in Base SAS software.

Dictionary tables make these tasks and others feasible with a minimal amount of coding effort. A brief outline of some of the tables' characteristics follows. They are:

- Only in Version 6.07 and higher.
- Automatically created during SAS System startup
- Maintained automatically during the course of the interactive session or batch job. One caveat: the tables are tied to implicit (WORK-level) and explicit LIBNAMEs — files allocated in MVS systems with DD statements will not be found in the tables.
- Read-only. You *cannot* change the table or view organization. You *can* affect their contents by making changes in the SAS environment. Whenever you change a system option, create a dataset, delete a member from a catalog, add a label to a variable, etc. one or more tables are updated.
- Accessible from SQL with the reserved LIBNAME of DICTIONARY (yes, that's a 10-letter LIBNAME!)

- Accessible outside SQL by using views defined in the LIBNAME of SASHELP. SASHELP is allocated automatically during startup.
- Present and maintained during your session whether you use them or not. There is no system option to suppress them.

So much for what they are. Let's take a look at what's inside them.

TABLE AND VIEW ORGANIZATION

The SAS System currently maintains 8 dictionary tables (available in SQL) and 14 views derived from these tables (available with the reserved LIBNAME of SASHELP). The tables and their relationship to the views are presented in Exhibit 1. Note that with the exception of the MEMBERS, which is the basis for seven views, each table has a corresponding view in SASHELP. The breakout of the MEMBERS table is discussed in more detail later in this section.

EXHIBIT 1: Relationship Between Dictionary Tables and SASHELP Views

Dictionary Table	SASHELP View
catalogs	vcatalg
columns	vcolumn
extfiles	vextfl
indexes	vindex
members	vmember
	vslib, vstable, vscatlg,
	vsaccess, vsview, vstabvw
options	voption
tables	vtable
views	vview

The rest of this section discusses each table and view. The output for the exhibits was generated by the following code:

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

In the above, *table* is the name of the dictionary table of interest (COLUMNS, CATALOGS, etc.).

Several methods are available for examining view contents. The CONTENTS procedure could be used as follows:

```
proc contents data=sashelp.view;
```

In the above, *view* is the name of the view of interest (vcolumn, vcatalg, etc.).

A third method of examining views is using the LIBNAME and DIR windows available in Display Manager. Exhibit 2 displays a DIR window, called directly from the Command Line of Display Manager or from within the LIBNAME window. The user can enter "S" next to the VIEW of interest to get field information similar to that given by the CONTENTS procedure. Entering "B" next to the view invokes the FSBROWSE procedure for the view (we assume, of course, that SAS/FSF software is installed at the user's site). Once the "B" command executes successfully the user sees a screen resembling that in Exhibit 3.

The full FSBROWSE command set is available for use when inspecting the views. A particularly useful feature is the ability to filter observations with a WHERE command. To find observations for a specific dataset, for example, enter:

```
where libname='PERM' & memname='FISCAL95'
```

EXHIBIT 2: Using DIR and FSBROWSE to Examine a SASHELP View

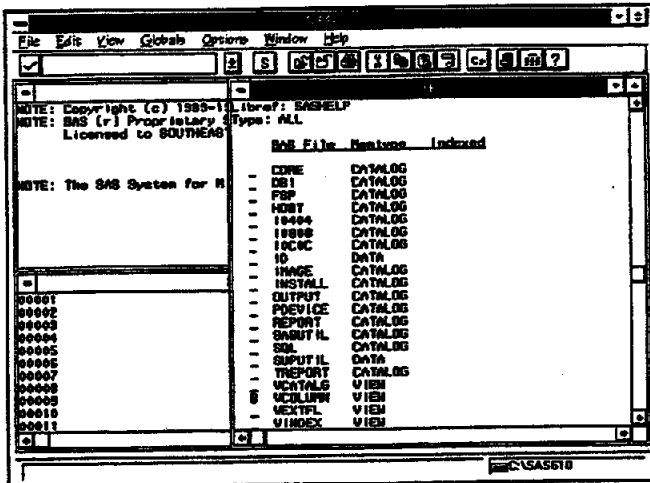
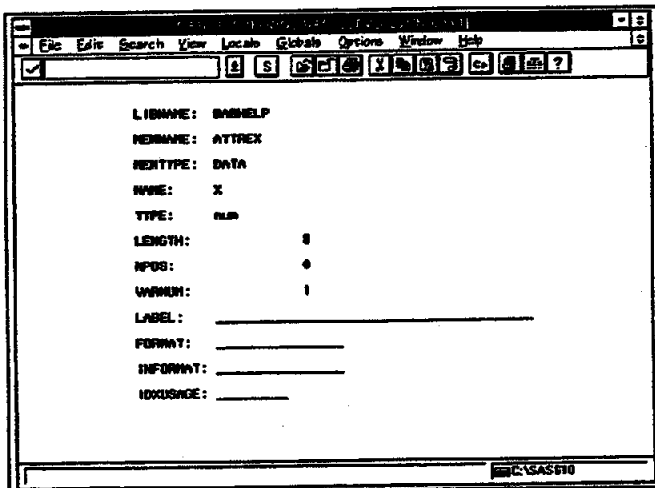


EXHIBIT 3: Using FSBROWSE to Examine a SASHELP View



CATALOGS

Contents: Information about all currently allocated SAS catalogs. See Exhibit 4 for details.

A Row/Observation Is: A catalog entry (e.g., PROGRAM, SCL, KEYS)

Comments: The table resembles the output from the CONTENTS option of the CATALOG procedure. CATALOG, however, cannot process more than one catalog at a time. See "Comparability" in the next section for a comparison of the CATALOGS table and CATALOG output datasets.

EXHIBIT 4: DICTIONARY.CATALOGS, SASHELP.VCATALG

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'

EXHIBIT 4: DICTIONARY.CATALOGS, SASHELP.VCATALG (CONTINUED)

OBJNAME	char(8)	label='Object Name'
OBJTYPE	char(8)	label='Object Type'
OBJDESC	char(40)	label='Object Description'
MODIFIED	char(8)	label='Date Modified'
ALIAS	char(8)	label='Object Alias'

COLUMNS

Contents: Information about variables in all currently allocated SAS datasets and views. See Exhibit 5 for details.

A Row/Observation Is: A variable in a dataset or view.

Comments: The table resembles the output from the CONTENTS procedure. CONTENTS, however, cannot process more than one library at a time. See "Comparability" in the next section for a comparison of the COLUMNS table and CONTENTS output datasets.

EXHIBIT 5: DICTIONARY.COLUMNS, SASHELP.VCOLUMN

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'
NAME	char(8)	label='Column Name'
TYPE	char(4)	label='Column Type'
LENGTH	num	label='Column Length'
NPOS	num	label='Column Position'
VARNUM	num	label='Column Number in Table'
LABEL	char(40)	label='Column Label'
FORMAT	char(16)	label='Column Format'
INFORMAT	char(16)	label='Column Informat'
IDXUSAGE	char(9)	label='Column Index Type'

EXTFILES

Contents: Information about currently allocated "external" (non-SAS) files. See Exhibit 6 for details.

A Row/Observation Is: An external file.

Comments: The table identifies files which are explicitly allocated (via FILENAME statements) as well as those which are allocated for internal use by the SAS System. The latter files are assigned filerefs by the SAS System. The naming convention varies by both operating system and release of the SAS System.

EXHIBIT 6: DICTIONARY.EXTFILES, SASHELP.VEXTFL

FILEREF	char(8)	label='Fileref'
XPATH	char(80)	label='Path Name'
XENGINE	char(8)	label='Engine Name'

INDEXES

Contents: Information about indexes used by the currently allocated SAS datasets. See Exhibit 7 for details.

A Row/Observation Is: Each occurrence of a variable used in an index.

Comments: Some of the information in this table is also found in the CONTENTS output dataset. See Exhibit 15 for a comparison.

EXHIBIT 7: DICTIONARY.INDEXES, SASHELP.VINDEX

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'
NAME	char(8)	label='Column Name'
IDXUSAGE	char(9)	label='Column Index Type'
INDXNAME	char(8)	label='Index Name'

EXHIBIT 7: DICTIONARY.INDEXES, SASHELP.VINDEX (CONTINUED)

INDXPOS	num	label='Position of Column in Concatenated Key'
NOMISS	char(3)	label='Nomiss Option'
UNIQUE	char(3)	label='Unique Option'

MEMBERS

Contents: Information about all SAS files currently allocated in the batch job or interactive session. These files include SAS datasets, view, access descriptors, catalogs, and stored programs. See Exhibit 8 for details.

A Row/Observation Is: A member of a currently allocated SAS file.

Comments: The PATH variable identifies either a single operating system directory name or *multiple* names, if the LIBNAME was part of a series of directories in a search path.

EXHIBIT 8: DICTIONARY.MEMBERS; SASHELP.VMEMBER, .VSLIB, .VSTABLE, .VSCATLG, .VSACCESS, .VSVIEW, .VSTABW

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'
ENGINE	char(8)	label='Engine Name'
INDEX	char(8)	label='Indexes'
PATH	char(80)	label='Path Name'

There are six SASHELP views which are row and column subsets of MEMBERS. The subsetting is described in Exhibit 9. To jog your memory, remember that the views begin with "VS" (presumably, view subset).

Exhibit 9: SASHELP views derived from MEMBERS

View Name	Subset of MEMBERS	Variables ¹
vslib	SAS libraries	L, PATH
vstable	SAS datasets	L, M
vscatlg	SAS catalogs	L, M
vsaccess	Access descriptors	L, M
vsviews	Views	L, M
vstabvw	SAS datasets	L, M, MEMTYPE

¹ L = LIBNAME, M = MEMNAME

OPTIONS

Contents: Information about current settings for all SAS system options. See Exhibit 10 for details.

A Row/Observation Is: A SAS system option.

Comments: The OPTIONS table and VOPTION view are good ways to review the meaning and current values of SAS System options. Unlike the Options Window in Display Manager, they also provide a description (variable OPTDESC) of each option.

Also unlike the window and OPTIONS procedure, the table and view permit options to be manipulated programmatically: you can retrieve the value of an option, store it in a macro variable, reset it, execute other statements, and then reset the option to its original value. This technique is especially useful for utility macros, since it enables them to alter system settings during the course of their execution and then reset them. Refer to Example 4 for an illustration of this technique.

EXHIBIT 10: DICTIONARY.OPTIONS, SASHELP.VOPTION

OPTNAME	char(16)	label='Session Option Name'
SETTING	char(200)	label='Session Option Setting'
OPTDESC	char(80)	label='Option Description'

TABLES

Contents: Information about currently allocated SAS data files. See Exhibit 11 for details.

A Row/Observation Is: A SAS data file.

Comments: The table contains information about SAS datasets, views, and SQL tables. See "Comparability" in the next section for a comparison of the TABLES table and CONTENTS output datasets.

EXHIBIT 11: DICTIONARY.TABLES, SASHELP.VTABLE

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'
MEMLABEL	char(40)	label='Dataset Label'
TYPEMEM	char(8)	label='Dataset Type'
CRDATE	num	label='Date Created'
MODATE	num	label='Date Modified'
NOBS	num	label='Number of Observations'
OBSLEN	num	label='Observation Length'
NVAR	num	label='Number of Variables'
PROTECT	char(3)	label='Type of Password Protection'
COMPRESS	char(8)	label='Compression Routine'
REUSE	char(3)	label='Reuse Space'
BUFSIZE	num	label='Bufsize'
DELOBS	num	label='Number of Deleted Observations'
INDXTYPE	char(9)	label='Type of Indexes'

VIEWS

Contents: Information about currently allocated SAS views. See Exhibit 12 for details.

A Row/Observation Is: A SAS view.

Comments: The table contains information about PROC SQL views, dataset views, and SAS/ACCESS view descriptors.

EXHIBIT 12: DICTIONARY.VIEWS, SASHELP.VVIEW

LIBNAME	char(8)	label='Library Name'
MEMNAME	char(8)	label='Member Name'
MEMTYPE	char(8)	label='Member Type'
ENGINE	char(8)	label='Engine Name'

COMMENTS AND USAGE NOTES

Before presenting examples of how the tables may be used we present some general comments concerning enhancements, performance, and comparability.

ENHANCEMENTS

Once a tool becomes well-worn and familiar the user often wants to tinker with its design and scope. Such is the case with the dictionary tables. We can identify at least two new tables which would be helpful to application developers.

The first is a system-specific table which would contain the current user id, system settings (default directory, path, environmental variables, etc.), and other session-related information. Some of this is already available in AF functions, some in automatic macro variables, and some via system-specific functions. It would be helpful to have it all in one location.

The other new table would contain macro variables and their settings. It would be helpful for developers if they could interactively examine or print such a table any time in a program. A similar table for macros would also be useful. Among the variables it could contain are: the macro name, source (autocall or in-stream), and parameter list. (This paper was originally presented at SUGI 20, April 1995. At the conference we learned that a macro table will be available in Version 6.11).

PERFORMANCE

As the Introduction stated, the tables are not a "free lunch." Rather, they are a "low cost lunch." Still a bargain, but you need to be aware of some of the performance penalties surrounding their use. In order

to get a feel for how performance of accessing tables compared to views we ran a series of benchmarks in several operating systems. The task was simple: retrieve all COLUMNS and TABLES information for several datasets, then do the same for their views (VCOLUMN and VTABLE). The access strategies are described below:

- **SQL Only.** Use only SQL to retrieve the data for the datasets of interest.
- **SQL, PROC.** Use SQL to create a table, then the PRINT procedure to display the information.
- **DATA, PROC.** Use a DATA step to create a dataset based on SASHELP views, then use the PRINT procedure to display the data.
- **View Only.** Use SASHELP views directly in a PRINT procedure.

The four test programs were run five times in each system. The last four runs' elapsed times were averaged. The results vary significantly across operating systems, and are presented in the graph in Exhibit 13. In all systems using the tables, either solely in PROC SQL or in SQL and a subsequent DATA step, yields consistently fast elapsed times. Once we use the SASHELP views in a DATA step performance suffers dramatically, although the difference between the view and table timings is less pronounced in the Unix system. The last test, using views directly in a PRINT procedure, produced the most varied timings. The Alpha and Windows timings were very slow compared to the table-oriented tests. OS/2 was only slightly slower and Unix was actually faster than the SQL timings.

It appears that an SQL-oriented approach to using the tables will, on average, produce time somewhat or significantly faster than methods using the SASHELP views. In some cases the selection of method is critical to the success of the application. Submitting a DATA step and procedure from a Windows AF application would require over 8 seconds to complete. Equivalent results could be obtained by submitting SQL and a procedure taking just over 2 seconds. The less time users of the system spend watching the Windows hourglass icon, the better.

COMPARISONS WITH PROCEDURE OUTPUT DATASETS

As we will see in the next section, one of the most appealing applications of the tables and views is generation of reports about SAS entities such as datasets, views, and catalogs. The "pre-table" way to do this outside the SAS/AF environment was to generate output datasets from the CATALOG or CONTENTS procedures. The datasets would then be used in the PRINT or REPORT procedures, possibly supplemented by user-written formats to improve readability.

Dictionary tables often offer a more complete range of information about the entity in question. Since they can be used directly by the reporting procedure or DATA step they require less coding effort. You have to consider, however, what you want to report and if it's available in the tables. The following Exhibits compare items in catalog and dataset-related output datasets and tables. Exhibit 14 compares the information in a CATALOG procedure output dataset with that contained in the CATALOGS table.

EXHIBIT 13: Access method timings

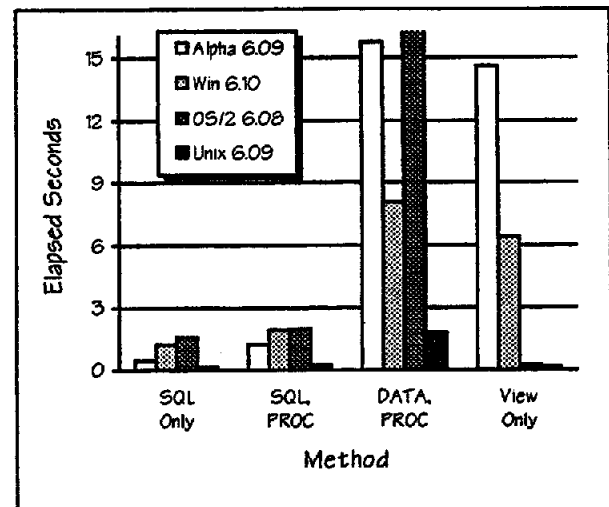


EXHIBIT 14: CATALOG output dataset compared to CATALOGS table

Item	CATALOG output dataset	CATALOGS table
LIBNAME	libname	libname
Catalog name	memname	memname
Catalog entry name	name	objname
Entry type	type	objtype
Member type		memtype
Description	desc	objdesc
Date modified	date	modified
Alias		alias

The comparison is not nearly as straightforward when we compare the CONTENTS output dataset to the dictionary tables. The complexity arises in part because the CONTENTS output dataset is not normalized. The unit of observation is the *variable*, but each observation contains repeating information about the *dataset* as a whole. This is convenient, but can render the task of gathering similar table information difficult. Exhibit 15 looks at the output dataset and the related dictionary tables. Please note that the comparisons are made for Version 6.10. CONTENTS variables may vary on other platforms and versions of the SAS System.

EXHIBIT 15: CONTENTS output dataset compared to dictionary tables

Item	CONTENTS output dataset	Table ¹
Dataset-level		
LIBNAME	libname	libname
Member	memname	memname
Dataset label	memlabel	t.memlabel
Member type - DATA, VIEW, etc.	memtype	c.memtype
Engine	engine	m.engine
# obs	nobs	t.nobs
# deleted obs	delobs	t.delobs
# vars in dataset		t.nvar
Date created	crdate	t.crdate
Date modified	moddate	t.moddate
Obs length		t.obslen
Protection	protect	t.protect
Compress option?	compress	t.compress
Reuse option?	reuse	t.reuse
Sorted?	sorted	
Character set	charset	
Collating sequence	collate	

EXHIBIT 15: CONTENTS output dataset compared to dictionary tables (continued)

Item	CONTENTS output dataset	Table ¹
NODUPREC when sorted?	noduprec	
NODUPKEY when sorted?	nodupkey	

Index information

# indexes	idxcount	
Usage (simple, composite, etc.)	idxusage	i.idxusage
Position of column in a concatenated key		i.idxpos
NOMISS option		i.nomiss
UNIQUE option		i.unique
Index name		i.idxname

Variable-level

Variable name	name	c.name
Data type	type	c.type
Length	length	c.length
Var # in dataset	varnum	c.varnum
Position in record (offset)	npos	c.npos
Variable label	label	c.label
• Format		
• name	format	c.format
• length	formatl	
• dec. places	formatd	
Informat		
• name	informat	c.informat
• length	informatl	
• dec. places	informatd	
Justification	just	
Var usage in indexes	idxusage	c.idxusage
Sort key #	sortedby	

¹ Two-level names in this column refer to dictionary tables, abbreviated as follows: c = columns, i = indexes, m = members, t = tables.

EXAMPLES OF USE

There are many applications for the information contained in the dictionary tables. In this section we present a series of examples taken from "real world" applications. Application background, code and, when appropriate, output are shown. Notice that most of the examples show the tables and views used as part of macros. This is in keeping with the high-level, meta-data nature of the information. Abstracted information and generalized tools such as a macros are a logical, complementary fit.

EXAMPLE 1: DISPLAYING VALUES FROM SASHELP VIEWS

The descriptions of the tables and views in the preceding section have, by necessity, been brief. It is helpful to have a listing of both the variables in each view and some of their observations. The macro in Exhibit 16 creates this reference material. In the first DATA step it counts the number of views associated with the LIBNAME of SASHELP. The DATA step also writes macro variables VIEW1-VIEWn, which contain the actual names of the views (VSTABLE, VCOLUMN, etc.). The %DO loop is executed for each VIEWx macro variable. It runs CONTENTS for the view, then prints a user-controlled number of observations from it. The VSVIEW view in the first DATA step accesses the views in alphabetical order, so the listing is also produced alphabetically: VCOLUMN is processed first, followed by VEXTFL and so on through VVIEW.

EXHIBIT 16: Macro to list contents and values from SASHELP views

```
%macro showview(obs=10);
data _null_;
set sashelp.vview(keep=memname libname)
end=eof;
if libname = 'SASHELP' & memname = 'V' then do;
call symput('view' ||
left(put(_n_,2.)), memname);
nview + 1;
end;
if eof then call symput('nview', put(nview,2.));
run;

%do i = 1 %to &nview.;
proc contents data=sashelp.&view&i.;
title "SASHELP view &lib..&view&i.";
run;

proc print data=sashelp.&view&i(obs=&obs.);
title2 "First &obs. observations";
run;
%end;
%mend;

%showview(obs=50);
```

EXAMPLE 2: USING DATASET NAME AND LABEL IN A PRINT UTILITY

The program in Exhibit 17 puts the VTABLE view to work. It prints a user-specified number of observations from a dataset. The title associated with the PRINT procedure is the dataset name and label. This is more informative than the default "The SAS System" title and more reliable than hand-coding the information. Notice that we could with relatively little effort put other useful, dataset-level information in the title: number of observations, number of variables, creation date, and other characteristics are available from the VTABLE view.

EXHIBIT 17: Use VTABLE to Create Informative TITLES

```
%macro printit(libname, memname);
proc sql noprint;
select label
into :label
from sashelp.vtable
where libname = "&libname." &
memname = "&memname." &
mentype in ("DATA", "VIEW");
quit;

proc print data=&libname..&memname.;
title "Dataset &libname..&memname. - &label.";
run;
%mend printit;
```

EXAMPLE 3: LIBRARY-LEVEL DATASET REPORTING

Project-related data is usually organized into distinct operating system files or directories. Clinical trials data for a particular compound, for example, may be in a directory (in VMS) such as "client:compound_name.data.sas". It would be helpful to have a compact listing of dataset characteristics (variable name, type, length, format, etc.) for all datasets in the library. The REPORT code which follows uses the VCOLUMN view to create such a listing. Notice that the report is sorted by dataset name. Variations on this are easily created: you could sort by dataset name-variable name, or variable name only. Some pre-processing with a DATA step or SQL could create a subset of the data identifying variable names occurring more than once, variables using user-written formats, and the like. The code in Exhibit 18, then, is only a starting point for what could be a library of reports.

EXHIBIT 18: Use VCOLUMN to Produce Report for an Entire Library

```

%macro lib_rept(libname=WORK);
%let libname = %upcase(&libname.);

options pageno=1;
proc report panels=3 pspace=2 spacing=1 headline
      nocenter nowindows
      data=sashelp.vcolumn
      (where=(libname="&libname."));
column memname name type length format ;
define memname / order ;
define name / 'Name' width=8;
define type / 'Type' width=4;
define length / 'Len.' width=4;
define format / 'Format' width=10;
title1 'Sorted by Dataset Name, Variable Name';
title2 'Created at &sysstime on &sysday,
      &sysdate';
run;
%mend lib_rept;

libname prj "client:[compound_name.data.sas]";
%lib_rept(prj);

```

EXAMPLE 4: Using OPTIONS to Save System Settings

An occasional, and unwanted, byproduct of using macros or included programs is that they reset system options. Aesthetic features such as centering or page numbering may be adjusted. Other, more critical, features may be reset to meet the needs of the program: options such as REPLACE, NODSNFERR, and NOFMterr can have a significant impact on execution of the "calling" program. One obvious solution is to know exactly what goes on in the subordinate code. Many factors - time constraints, unpublished revisions of working code, and the like - conspire to make complete knowledge an elusive goal.

The macro in Exhibit 19 uses the OPTIONS table to remember and restore the value of several system options. The macro requires options CENTER and DATE to be turned off, so it obtains those values from the Dictionary Table, sets them (the OPTIONS statement), runs a procedure, then resets CENTER and DATE to their original values prior to terminating.

EXHIBIT 19: Save/Restore Values of System Options

```

%macro util1;
proc sql noprint;
select setting into :tablcnt
  from dictionary.options
  where optname="center";
select setting into :tabldate
  from dictionary.options
  where optname="date";
quit;

options nodate nocenter;
... procs go here ...
* Reset to original values
options &tablcnt. &tabldate.;
%mend;

```

EXAMPLE 5: Moving Command Procedure Code into the SAS Program

An applications programmer will typically not let SAS code exist in a vacuum, unaffected by the host operating system. Systems usually have a command language which perform tasks more efficiently and easily than would be possible in the SAS System. The VAX command language, DCL, for example has numerous functions which allow the programmer to test for the presence of files, carry on a dialog with the user, and other tasks.

The dictionary tables do not replace all of this functionality. However, the information they contain may encourage some programmers to move activities previously reserved for system command languages into SAS code. This makes programs more self-contained and thus more portable across platforms.

A before-and-after example is presented in Exhibit 20. Output generated from procedures needs to be routed to the user's SASUSER directory. Since the path is system dependent, code specific to the host system can be used to retrieve the name of the directory (top portion of the Exhibit). However, using dictionary tables, the path can be obtained without using code specific to the host (bottom portion of the Exhibit). This makes the application more portable.

EXHIBIT 20: Moving Command Language Code into the SAS System

```

Using Operating-System Specific Code
x vms 'mydir = f$trlnm("sasuser");
%let path = %sysget(mydir);

proc printto print="&path.myfile.lis";

Using Dictionary Tables
proc sql noprint;
select path into :path
  from dictionary.members
  where libname = 'SASUSER';

%let path = %qtrim(&path);

proc printto print="&path.myfile.lis";

```

EXAMPLE 6: Counting Observations in a Dataset

Decision-making during macro execution is sometimes based on the number of observations in a dataset. There are many ways to do this in Base SAS (and still more in SAS/AF). Not surprisingly, the topic is brought up here because the TABLES table provides a fairly clean method for answering two questions simultaneously: does the dataset exist, and if so, how many observations does it contain? The macro NINDSET is found in Exhibit 21. The user specifies the name of the dataset using the standard two-level name notation. NINDSET creates macro variable COUNT. It is -1 if the dataset could not be located, 0 or higher otherwise (indicating the number of observations in the dataset).

EXHIBIT 21: Observation Counting with TABLES Table

```

%macro nindset(data);
%let data = %upcase(&data.);
%global count;
%let count=-1;
%if &data. = %str() %then %let data = _last_;
%let libname = %scan(&data., 1, ". ");
%let dsname = %scan(&data., 2, ". ");
%if &dsname. = %str() %then %do;
  /* Adjust if we have a 1-level dset name ;
  %let dsname = &libname.;
  %let libname = work;
%end;

proc sql noprint;
select nob
  into :count
  from sashelp.vtable
  where memname="&dsname." &
        libname = "&libname." &
        memtype = 'DATA';
quit;

%put NINDSET: COUNT value for &data. is &count.;
%mend nindset;

```

SUMMARY

Dictionary tables and their associated SASHELP views offer the SAS programmer a rich variety of meta data to use in applications. The programmer should be aware of possible performance penalties. However, as the examples demonstrated, having information about most aspects of the SAS environment far outweighs the usually negligible cost in CPU time. The tables can be used to create useful macro-based utilities which can make programs and systems more reliable and easier to craft.

COMMENTS?

Your comments are valued and encouraged. Contact either of us:

Frank Dilorio
ASG, Inc.
2000 Regency Pkway, Suite 355
Cary, NC 27511

7216987@mcimail.com

919-467-0505

Nancy Michal
MEDCO Research, Inc.
85 Alexander Drive
Research Triangle Park, NC 27709

919-549-8117