Paper 264-2011

SAS® DICTIONARY: Step by Step

Patrick Thornton, SRI International, Menlo Park, CA

ABSTRACT

This paper offers a step-by-step visual review of the SAS® 9.2 DICTIONARY tables for the beginning to intermediate SAS user who is generally unfamiliar with PROC SQL and macro programming. The content of the DICTIONARY tables (such as data set names; variable names, labels, and types; option names and values; titles and footnotes; format and macro program catalog entries; and external file and library paths) is visually explored using common procedures and SASHELP views. This paper demonstrates that the DICTIONARY is easily accessible and shows many valuable uses of the DICTIONARY. This paper also shows some alternative techniques for obtaining metadata when applicable and directs users to papers that feature advanced techniques using the DICTIONARY.

INTRODUCTION

The SAS DICTIONARY library contains metadata. Metadata is information about data, such as data set names and variable names, but it may also be information about the broader SAS programming environment. When discussing metadata the terms data set/table and variable/ field are often used interchangeably. Table 1 provides an overview of the type of metadata found in a special library called DICTIONARY. I have organized this metadata into seven categories that are shown as column headings. The first column lists the types of SAS session metadata that is available in DICTONARY, such as, option settings and titles. "Members" is the name of the second column. SAS stores some information about data sets (tables), catalogs, and views in a table called MEMBERS. The third column, lists the types of more detailed metadata that is available regarding data sets and views. This column also includes a listing of all tables and fields found in the DICTIONARY library which, not so coincidentally, is stored in a table called DICTONARY. Column four lists the types of metadata that is stored for catalogs and related entities, such as, formats. The last three columns list metadata that is perhaps less well known and less used. They include metadata on any constraints that exist to validate data, information maps, and an "other" category. This paper will explore all of the sources of metadata except the information map metadata.

Table 1 Overview of Metadata Found in the DICTIONARY Library

(1) SAS Session	(2) Members	(3) Data Set and Related Metadata	(4) Catalogs and Related Metadata	(5) Constraints	(6) Information Maps	(7) Other
SAS Options SAS/Graph Options TITLE Statements Defined macro variables LIBNAME Information Available Engines Files defined in FILENAME statements, or implicitly	Tables, Catalogs, and Views	Table and Table-Specific Information View and View-Specific Information Columns from every table DICTIONARY tables and their columns Indexes	Catalog and Catalog-Specific Information Available Formats Available Macro Programs	Check Constraints Referential Constraints Table Constraints Constraint table usage Constraint column usage	Information Maps* Information Map Filters* Information Map Data Items* Information Map Prompts* Information Map Prompts XML*	Open ODS Destinations* Styles Functions* Remember information

*New in 9.2;

Metadata is often used to help manage, organize and search data or other resources (Zhang, Chen, and Wong, 2003). For example, many programmers have experience using PROC CONTENTS in order to determine the existence of a variable, either by name or label, determine a variable type (i.e. character or numeric), or its length and assigned format. Metadata may also be used to create syntax. This use of metadata is particularly useful when there are many and/or changing specifications. Many papers demonstrate programs that use metadata to drive the creation of syntax (e.g. DiIorio & Abolafia, 2004; Eberhardt & Brill, 2006; Lafler, 2005; Varney, 2004), but this paper does not assume knowledge of advanced programming.

Exploration of metadata is also a terrific way to learn about the SAS System. This paper will explore the metadata in DICTIONARY and list or discuss many valuable uses of metadata, such as:

- Saving and restoring system options (Dilorio & Abolafia, 2004)
- Saving and restoring titles and footnotes (Eberhardt and Brill ,2006)
- Obtaining library or external file paths, in order to, for example, document output
- Discovering available LIBNAME engines (e.g. EXCEL, see Choate and Martell (2006))
- Counting and accounting for a wide range of SAS resources (e.g. data sets, views, catalogs)
- Determining the number of observations in one or more data sets
- Listing all data sets in a library that have variables in common (Lafler, 2005)
- Identifying inconsistencies in the length and type for variables of the same name across data sets in a library (Dilorio & Abolafia, 2004)
- Applying a consistent label and format to variables of the same name across all data sets in a library (Eberhardt & Brill, 2006)
- Listing indexes, and determining the existence of an index (Mack, 2006)
- Adding descriptions to formats and printing a list of formats and descriptions
- Adding descriptions to macro programs and printing a list of descriptions (Thornton, 2008)
- Listings integrity constraint syntax and descriptions (Thornton, 2007)

INTRODUCTION TO SAS DICTIONARY LIBRARY

Lafler, (2005), Eberhardt & Brill (2006), Mack (2006), and DiIorio & Abolafia (2004) all provide excellent introductions to accessing and using the tables of metadata found in the DICTIONARY library. Cates (2002) described a SAS library as a way of accessing a variety of SAS files as a group. On a Windows operating system a library usually refers to a folder containing files that are recognized by SAS, such as data files. A LIBNAME statement is used to define the name of a library and its folder; however, libraries may also be automatically defined by SAS. For example, using SAS 9.2, the DICTIONARY and SASHELP libraries are both among those automatically available upon starting the software.

DICTIONARY is a special case of a SAS library. SAS retains exclusive write-access to the metadata tables and only allows the tables to be read directly using PROC SQL. Fortunately much of DICTIONARY is assessable indirectly through procedures and DATA step syntax using views provided in the SASHELP library (Davis, 2001; Gerlach, 2005; Mack, 2006). The following syntax is used to generate a list of all the views in the SASHELP library (Figure 1):

```
proc contents data=sashelp._all_
memtype=view;
run;
```

With this list of views, exploration of the contents is very straight forward. For example, the following syntax is all that is necessary to examine both the fields and the records of metadata returned by the VTITLE view:

```
proc contents data=sashelp.vtitle;
run;
proc print data=sashelp.vtitle(obs=10);
run;
```

Some of these 37 views return redundant metadata, and it is important to remember that the views only return data from tables found in the DICTIONARY library.

Figure 1 Views Returning Metadata from DICTIONARY

#	Name	Member Type	Level	File Size	Last Modified
1	VALLOPT	VIEW	2	5120	22Dec08:13:40:56
2	VCATALG	VIEW	2	5120	22Dec08:13:40:56
3	VCFORMAT	VIEW	2	5120	22Dec08:13:41:02
4	VCHKCON	VIEW	2	5120	22Dec08:13:41:00
5	VCNCOLU	VIEW	2	5120	22Dec08:13:41:06
6	VCNTABU	VIEW	2	5120	22Dec08:13:41:06
7	VCOLUMN	VIEW	2	5120	22Dec08:13:41:10
8	VDATAIT	VIEW	2	5120	22Dec08:13:41:10
9	VDCTNRY	VIEW	2	5120	22Dec08:13:41:14
10	VDEST	VIEW	2	5120	22Dec08:13:41:16
11	VENGINE	VIEW	2	5120	22Dec08:13:41:18
12	VEXTFL	VIEW	2	5120	22Dec08:13:41:20
13	VFILTER	VIEW	2	5120	22Dec08:13:41:24
14	VFORMAT	VIEW	2	5120	22Dec08:13:41:24
15	VFUNC	VIEW	2	5120	22Dec08:13:41:28
16	VGOPT	VIEW	2	5120	22Dec08:13:41:28
17	VINDEX	VIEW	2	5120	22Dec08:13:41:32
18	VINFOMP	VIEW	2	5120	22Dec08:13:41:34
19	VLIBNAM	VIEW	2	5120	22Dec08:13:41:36
20	VMACRO	VIEW	2	5120	22Dec08:13:41:38
21	VMEMBER	VIEW	2	5120	22Dec08:13:41:42
22	VOPTION	VIEW	2	5120	22Dec08:13:41:42
23	VPRMXML	VIEW	2	5120	22Dec08:13:41:46
24	VPROMPT	VIEW	2	5120	22Dec08:13:41:46
25	VREFCON	VIEW	2	5120	22Dec08:13:41:50
26	VREMEMB	VIEW	2	5120	22Dec08:13:41:52
27	VSACCES	VIEW	2	5120	22Dec08:13:41:54
28	VSCATLG	VIEW	2	5120	22Dec08:13:41:56
29	VSLIB	VIEW	2	5120	22Dec08:13:42:00
30	VSTABLE	VIEW	2	5120	22Dec08:13:42:00
31	VSTABVW	VIEW	2	5120	22Dec08:13:42:04
32	VSTYLE	VIEW	2	5120	22Dec08:13:42:04
33	VSVIEW	VIEW	2	5120	22Dec08:13:42:08
34	VTABCON	VIEW	2	5120	22Dec08:13:42:10
35	VTABLE	VIEW	2	5120	22Dec08:13:42:12
36	VTITLE	VIEW	2	5120	22Dec08:13:42:14
37	VVIEW	VIEW	2	5120	22Dec08:13:42:18

There are 29 tables in DICTIONARY, and this paper will explore 29 SASHELP views that may be used to access the metadata. The following table organizes the 29 metadata tables into the seven categories previously introduced and shows a cross walk between each of the tables and a view that may be used to access the metadata.

Table 2 An Organized List of DICTIONARY Tables and SASHELP Views

Category	DESCRIPTION	DICTIONARY Table	SASHELP VIEWS
SAS Session	SAS Options	OPTIONS	VOPTION
	SAS/Graph Options	GOPTIONS	VGOPT
	TITLE Statements	TITLES	VTITLE
	LIBNAME Information	LIBNAMES	VLIBNAM
	Available Engines	ENGINES	VENGINE
	Files defined in FILENAME statements, or implicitly	EXTFILES	VEXTFL
	Defined macro variables	MACROS	VMACRO
Members	Tables, Catalogs, and Views	MEMBERS	VMEMBER
Data Set and Related Metadata	Table and Table-Specific Information	TABLES	VTABLE
	View and View-Specific Information	VIEWS	VVIEW
	Columns from every table	COLUMNS	VCOLUMN
	Indexes	INDEXES	VINDEX
	DICTIONARY tables and their columns	DICTIONARIES	VDCTNRY
Catalogs and Related Metadata	Catalog and Catalog-Specific Information	CATALOGS*	VCATALG
	Available Formats	FORMATS	VFORMAT
Constraints	Check Constraints	CHECK_CONSTRAINTS	VCHKCON
	Referential Constraints	REFERENTIAL_CONSTRAINTS	VREFCON
	Table Constraints	TABLE_CONSTRAINTS	VTABCON
	Constraint table usage	CONSTRAINT_TABLE_USAGE	VCNTABU
	Constraint column usage	CONSTRAINT_COLUMN_USAGE	VCNCOLU
Information Maps	Information Maps	INFOMAPS	VINFOMP
	Information Map Filters	FILTERS	VFILTER
	Information Map Data Items	DATAITEMS	VDATAIT
	Information Map Prompts	PROMPTS	VPROMPT
	Information Map Prompts XML	PROMPTSXML	VPRMXML
Other	Open ODS Destinations	DESTINATIONS	VDEST
	Styles	STYLES	VSTYLE
	Functions	FUNCTIONS	VFUNC
	Remember information	REMEMBER**	VREMEMB

* SASHELP.VCATALG may be used to list, among other entities, macro programs; **No documentation was found.

This paper will use Table 2 as an online and systematically explore the metadata returned by most of the DICTIONARY tables via the SASHELP views. The paper will not discuss information map or functions metadata.

SAS SESSION METADATA

There are 6 tables of metadata shown in Table 2 that contain SAS session information: OPTIONS, GOPTIONS, TITLES, LIBNAMES, ENGINES, EXTFILES and MACROS. The following section explores the metadata using each of the

corresponding views listed in the last column of the Table 2: VOPTION, VGOPT, VTITLE, VLIBNAM, VENGINE, VEXTFL and VMACRO. The OPTIONS and GOPTIONS metadata tables have the same fields, so GOPTIONS is only discussed by proxy. The MACRO metadata is only very briefly mentioned.

SAS AND SAS/GRAPH OPTIONS METADATA (EXPLORING OPTIONS/GOPTIONS TABLES USING VOPTION/VGOPT VIEWS)

The SASHELP.VOPTION view returns metadata records from the OPTIONS table of the DICTIONARY library. I use the results of a PROC CONTENTS and PROC PRINT to explore the metadata returned by VOPTION:

```
proc contents data=sashelp.voption;
proc print data=sashelp.voption (obs=10);
run;
```

I use these basic procedures to explore all the VIEWS discussed in this paper. The result of PROC CONTENTS on VOPTION returns metadata on the fields OPTNAME, OPTTYPE, SETTING, OPTDESC, LEVEL, and GROUP; however, since views do not contain data, PROC CONTENTS does not show the number of observations. While views do not contain data, they do return data to a DATA step and to most procedures, so I used PROC PRINT to show the first 10 observations returned by VOPTION. Looking at this metadata, OPTNAME contains the option name, SETTING contains the current setting of the option, and OPTDESC contains a description of the option. The other fields all seem to classify the option. So, based on the metadata, I deduced that the VOPTION view returns a record for each of the options available in the SAS session. I built the following PROC TABULATE syntax to summarize the metadata:

```
proc tabulate data=sashelp.voption noseps format=8.0 ;
      class level opttype group;
      table group all,level*opttype all /rts=20 misstext="0";
run;
```

The PROC TABULATE result shows that the VOPTION view returns 394 (SAS 9.1 326) records (Figure 2). Each option is classified into a functional group shown in the first column and as being host specific or portable. The GOPTIONS metadata table has the same structure as the OPTIONS table, and the metadata from both may be accessed in the same way using the VALLOPT view.

Figure 2 Count of Options by Group, Level and Type

	Option Location						
	Host			Portable			
	Option type			Option type			All
	Boolean	char	num	Boolean	char	num	All
	N	N	N	N	N	N	N
Option Group							
CODEGEN	0	0	0	0	0	1	1
COMMUNICATIONS	0	5	0	11	9	6	31
DATAQUALITY	0	0	0	0	2	0	2
EMAIL	0	2	0	1	4	1	8
ENVDISPLAY	16	14	0	4	2	1	37
ENVFILES	0	13	0	3	13	1	30
ERRORHANDLING	0	1	0	12	2	0	15
EXECMODES	0	3	0	7	6	0	16
EXTFILES	0	0	0	2	0	1	3
GRAPHICS	0	1	0	1	2	0	4
HELP	0	2	0	1	5	1	9
INPUTCONTROL	1	2	2	5	3	5	18
INSTALL	0	2	0	1	0	0	3
LANGUAGECONTROL	2	4	0	1	2	0	9
LISTCONTROL	2	0	0	4	4	3	13
LOGCONTROL	5	3	0	7	4	2	21
LOG_LISTCONTROL	0	0	0	5	1	2	8
MACRO	0	0	0	19	4	2	25
MEMORY	0	1	5	0	0	1	7
META	0	0	0	0	12	1	13
ODSPRINT	2	1	0	7	13	3	26
PDF	0	0	0	6	3	0	9
PERFORMANCE	0	0	1	2	10	9	22
SASFILES	2	0	2	4	16	6	30
SECURITY	0	0	0	2	4	0	6
SORT	0	5	2	2	2	0	11
SQL	0	0	0	2	4	2	8
SVG	0	0	0	1	7	0	8
UNKNOWN	0	1	0	0	0	0	1
All	30	60	12	110	134	48	394

The metadata does not just serve to summarize the available options, it can also provide the real-time status of the options. For example, the following PROC REPORT syntax is used to list metadata for the DATE option:

```
proc report data=sashelp.voption nowd;
    column optname opttype optdesc setting;
    define optdesc / width=35 flow;
    define setting / width=10 flow;
    where optname in ("DATE");
run;
```

PROC REPORT, rather than the PRINT procedure is used because the OPTDESC field returned by VOPTION is quite long. In this example, PROC REPORT is functionally very similar to PROC PRINT, where the COLUMNS statement determines the order of the fields in the listing, synonymous to a VAR statement. However, the DEFINE statement allows field widths to be controlled and FLOW allows the text in OPTDESC to wrap. The option name, type, description, and setting for the DATE option are listed in Figure 3.

Figure 3 VOPTION Metadata for the DATE Option

Option Name	Option type	Option Description	Option Setting
DATE	boolean	Print date and time on top of each page of SAS log and procedure output	DATE

The DATE option has an OPTTYPE value of Boolean, so it may either have a value of DATE or NODATE for the SETTING field. As shown in Figure 2, some option settings may also be character or numeric. In order to demonstrate the real-time information provided by DICTIONARY, the following OPTION statement is used to change the setting to NODATE, and new output from PROC REPORT is shown in Figure 4.

```
Option nodate;
```

Figure 4 VOPTION Metadata after Setting NODATE

Option Name	Option type	Option Description	Option Setting
DATE	boolean	Print date and time on top of each page of SAS log and procedure output	NODATE

The metadata returned by VOPTION immediately reflects the change in option setting; the SETTING field changed to the value NODATE (Figure 4). The OPTIONS metadata has been exploited by capturing it prior to changes and using the captured metadata to reset the SAS session to the original option settings. For example, Dilorio and Abolafia (2005) created a program to save the options and settings to a data set in the WORK library in order to later use it to reestablish the original option settings. While the authors created an elegant utility macro using PROC SQL, the following is a more basic syntax.

```
❶ proc sort data=sashelp.voption out=m_original; by optname; run;
data m_original;
      set m_original;
      length keyword $1060.;
❷     keyword = getoption(optname,'keyword');
run;
❸ option nodate nocenter pageno=100;
❹ proc sort data=sashelp.voption out=m_current; by optname; run;
data _null_;
❺ merge m_original m_current (keep=optname setting rename=(setting=csetting));
      by optname;
❻     if setting ne csetting then do;
            put "Changed Setting: " optname $10. setting $50.;
❼           call execute("option "||trim(keyword)||";");
      end;
run;
```

❶ PROC SORT creates the data set M_ORIGINAL to store the current option settings returned by VOPTION.

❷ The GETOPTION function is used (Dilorio & Abolafia, 2005) to set the value of the KEYWORD field to the syntax needed in an OPTION statement.

❸ Options are changed using an OPTION statement to demonstrate how a program might alter existing settings.

❹ The new option settings are returned from VOPTION and saved to the M_CURRENT data set.

❺ The original option settings and the new option settings are compared by merging M_ORIGINAL and M_CURRENT by OPTNAME. In the merge, M_CURRENT fields OPTNAME and SETTING are kept and the SETTINGS field is renamed to CSETTINGS.

❻ When the value of original option settings (SETTINGS) is different from the new options settings (CSETTING), the KEYWORD field having the original option statement syntax is used to reset the option.

❼ CALL EXECUTE is used to submit the KEYWORD values as syntax, and the following shows a portion of the LOG output that is generated:

```
 NOTE: CALL EXECUTE generated line.
1   + option CENTER;
2   + option DATE;
3   + option PAGENO=1;
```

For each observation of the DATA step where the new option setting was different from the original option setting, CALL EXECUTE produces the OPTION statement from the literal value of “option “concatenated with the value of KEYWORD and an ending semicolon.

There is at least one other way to capture and reset system options that may be much easier than using VOPTION (Heaton, 2003):

```
option date center pageno=1;
Proc optSave out=m_original;
Run ;
option nodate nocenter pageno=100;
Proc optLoad data=m_original;
Run ;
```

A downside to this method though is that every option is reset by PROC OPTLOAD which had the effect of repositioning my editor window. An advantage to using VOPTION in a DATA step seems to be more control.

TITLE AND FOOTNOTE METADATA (EXPLORING TITLES TABLE USING VTITLE VIEW)

The records of the TITLES table are accessed using VTITLE. The CONTENTS procedure is used to list the fields of metadata returned from the VTITLE view and the PRINT procedure creates the listing shown in Figure 5.

```
proc contents data=sashelp.vtitle;
run;
proc print data=sashelp.vtitle;
run;
```

Figure 5 Metadata Returned by VTITLE

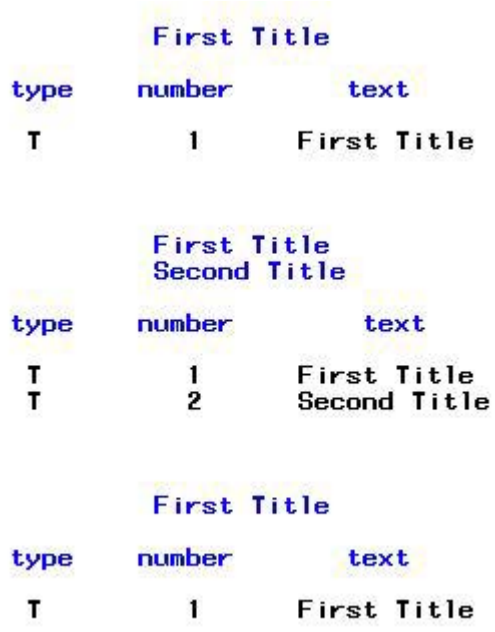
Obs	type	number	text
1	T	1	The SAS System

Figure 5 shows that the default title, “The SAS System,” is stored in the TEXT field. The TYPE field has the value of “T” indicating that the record stores a title. Type may also have the value “F” which indicates that the record stores a footnote. The NUMBER field identifies the level of title or footnote.

Figure 6 shows the VTITLE metadata produced by the following syntax.

```
option nodate formdlim=' ' nonumber;
title1 "First Title";
proc print data=sashelp.vtitle noobs;
run;
title2 "Second Title";
proc print data=sashelp.vtitle noobs;
run;
title2;
proc print data=sashelp.vtitle noobs;
run;
option date formdlim='' number;
```

Figure 6 Example Metadata Returned from VTITLE



First Title

type	number	text
T	1	First Title

First Title
Second Title

type	number	text
T	1	First Title
T	2	Second Title

First Title

type	number	text
T	1	First Title

The FORMDLIM option is set to a blank space which allows the multiple PROC PRINT output to appear on the same page of the output window. The TITLE1 statement is used to create "First Title" and the first PROC PRINT output in Figure 6 shows that the new title is in the metadata. The TITLE2 statement is used to create the "Second Title" and the second PROC PRINT again shows that the new title is in the metadata. Last, the TITLE2 is blank and the final listing in Figure 6 shows that the second title is no longer in the metadata. Changes in session footnotes may be similarly demonstrated.

I only found one use of title and footnote metadata documented in the literature. Eberhardt and Brill (2006) used VTITLE in a manner analogous the previous use of VOPTION metadata. The authors used a macro program and DATA step syntax to capture the current titles and footnotes in a SAS session by saving the metadata records retuned by VTITLE to a data set and then later using that metadata to reset the titles.

LIBNAME INFORMATION METADATA (EXPLORING LIBNAMES TABLE USING VLIBNAM VIEW)

The VLIBNAM view of the SASHELP library is used to explore the LIBNAMES table metadata that contains information about libraries known to the SAS session. PROC CONTENTS lists the 10 fields returned by VLIBNAM and PROC PRINT shows the 3 columns of metadata presented in Figure 7.

```
proc contents data=sashelp.vlibnam;
run;
proc print data=sashelp.vlibnam noobs;
var libname level path;
where libname in ("SASHELP","MAPS");
run;
```

Figure 7 Listing of Metadata Returned by VLIBNAM from the LIBNAMES Table

libname	level	path
SASHELP	1	C:\Program Files\SAS\SASFoundation\9.2\nls\en\SASCFG
SASHELP	2	C:\Program Files\SAS\SASFoundation\9.2\core\sashelp
SASHELP	3	C:\Program Files\SAS\SASFoundation\9.2\genetics\sashelp
SASHELP	4	C:\Program Files\SAS\SASFoundation\9.2\graph\sashelp
SASHELP	5	C:\Program Files\SAS\SASFoundation\9.2\stat\sashelp
MAPS	0	C:\Program Files\SAS\SASFoundation\9.2\maps

Figure 7 shows 6 metadata records associated with the MAPS and SASHELP libraries. The LIBNAME field stores the name of the library and the PATH field stores the folder on the Windows operating system where the library members (e.g. data sets) are located. The PATH field has a maximum length of 1024. The MAPS library has a single metadata record; however, the SASHELP library has 5 records (4 in SAS 9.1). SASHELP is termed a concatenated library, thus SASHELP has more than one path. The metadata contains a record for each of the paths and a sequence of LEVEL values. SAS provided full support for library concatenation with Version 7 in order to allow users to create a single library that may be used to access different versions of SAS files (Beatrous & Clifford, 1998).

Documented uses of VLIBNAM metadata are very limited. The most common use of library metadata seems to be for documentation, and there is a convenient alternative to the use of VLIBNAM. The path of a library may be obtained using SAS functions, for example the following syntax creates a footnote with the MAPS library name and path.

```
%let libname=maps;
%let libpath =%sysfunc(pathname(&libname));
footnote1 "&libname : &libpath";
```

The %LET statement is used to set a macro variable LIBNAME equal to the name of an existing library, MAPS. The value of LIBNAME is included as the argument to the function PATHNAME by placing an "&" before the macro variable. The "&" causes SAS to resolve the value of LIBNAME to MAPS, thus the function PATHNAME, with the help of %SYFUNC, sets the macro variable LIBPATH equal to the operating system path of the MAPS library. Both the library name and path are shown in a footnote, again by placing an "&" as the prefix to both macro variables.

Other uses of library metadata are to save to a permanent data set the library names and paths used in a project and then use this data set to create LIBNAME statements at the beginning of a session (Kolosova & Berestizhevsky, 1996).

AVAILABLE ENGINES METADATA (EXPLORING ENGINES TABLE USING VENGINE VIEW)

Engines are software components that are referenced through an engine name and various options. Engines are important because they allow files to be recognized and used by the SAS system. The VENGINE view of the SASHELP library is used to assess the engine metadata found in the ENGINES table of the DICTIONARY library. The result of the PROC PRINT is shown in Figure 8.

```
proc contents data=sashelp.vengine;
run;
proc print data=sashelp.vengine (obs=10) noobs;
var engine description;
run;
```

Figure 8 ENGINES Metadata Returned by the VENGINE View

engine	description
ACCESS	SAS/ACCESS Interface to PC Files
ACCESS99	Read engine for CRSP ACCESS97 database
BASE	Base SAS I/O Engine
BLOOMBRG	SAS/Access Interface To Bloomberg
BMDP	BMDP Save file engine
CRSPACC	Read engine for CRSP ACCESS97 database
CVP	Character Variable Padding Engine
DB2	SAS/ACCESS Interface to DB2
EXCEL	SAS/ACCESS Interface to PC Files
FAMECHLI	Seamless libname interface to FAME db

There are a total of 50 metadata records returned by VENGINE (46 in SAS 9.1); however, Figure 8 only shows the first 10. Simply obtaining a list of engines is a convenient opportunity to learn about the capabilities of SAS. When a LIBNAME statement is used to establish a library, SAS often determines the appropriate engine to use; however, there are other times when specifying an explicit engine in a LIBANME statement offers powerful access to files that would otherwise be inaccessible or far less accessible. For example, there is an ENGINE that allows SAS to recognize, access, and use EXCEL files. Choate and Martell (2006) described the use of LIBNAME and the Excel ENGINE to enable SAS to recognize as data sets Excel worksheets within a workbook. While the authors pointed out notable limitations in the engine, access is enabled with the simple inclusion of the engine name, EXCEL, in the LIBNAME statement. For example, the following syntax obtains basic metadata for all the sheets in an Excel workbook without having to actually import the Excel data.

```
libname exc excel "c:\temp\MyExcelFile.xls";
proc contents data=exc._all_;
run;
libname exc clear;
```

The use of _ALL_ generates a PROC CONTENTS output for all the worksheets found in the Excel workbook. By default, the values of the first row of each sheet are assumed to be the variable names. SAS also determines the variable TYPE, LENGTH, FORMAT, and INFORMAT; however, options of the EXCEL engine in the LIBANAME statement may be used to guide SAS in its determination of some of these specifications (Choate & Martell, 2006). More will be discussed in later sections about the type and use of metadata that are available for data sets and variables. I have not discovered any other uses of the ENGINE metadata.

EXTERNAL FILE METADATA (EXPLORING EXTFILES TABLE USING VEXTFL VIEW)

The EXTFILES table of DICTIONARY stores references to external files and folders known to the SAS session. The output of PROC CONTENTS on VEXTFL is shown in Figure 9.

Figure 9 Fields Returned by VEXTFL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Flags	Format	Informat	Label
7	directory	Char	3	P--			Directory?
8	exists	Char	3	P--			Exists?
1	fileref	Char	8	P--			Fileref
5	filesize	Num	8	P--			Size of File
6	level	Num	8	P--			File Concatenation Level
4	modate	Num	8	P--	DATETIME.	DATETIME.	Date Modified
9	temporary	Char	3	P--			Temporary?
3	xengine	Char	8	P--			Engine Name
2	xpath	Char	1024	P--			Pathname

The value of the FILREF field contains the name by which SAS refers to operating system files/folders. The paths are stored in the XPATH field. As a demonstration, the following syntax establishes a filename called MYFILE, and VEXTFL and PROC PRINT are used to return and list metadata for MYFILE (Figure 10).

```
filename myfile "c:\temp\test.txt";
proc print data=sashelp.vextfl;
where fileref='MYFILE';
run;
```

Figure 10 Listing of Metadata Returned by VEXTFL

Obs	fileref	xpath	xengine	modate
18	MYFILE	c:\temp\test.txt	DISK	21JAN11:08:41:53

Obs	filesize	level	directory	exists	temporary
18	0	0	no	yes	no

The following FILENAME statement with the CLEAR option removes the MYFILE filename from the metadata.

```
filename myfile clear;
```

If you print all of the records in VEXTFL you will notice that there are metadata for files and folders that you did not create with a FILENAME statement. These FILEREF values seem to begin with "#." Some are created by SAS automatically and reference SAS-specific folders on your operating system. Other file references may exist if you have saved short cuts in your favorite's folder using the SAS Explorer window. In SAS 9.1, EXTFILES contained the fields FILEREF, XPATH, and XENGINE, but in SAS 9.2 there are 6 additional fields:

- EXISTS - yes =XPATH file or folder exists on the OS, no = XPATH file or folder does not exist on the OS
- DIRECTORY - yes =XPATH contains a folder, no=XPATH does not contain a folder
- TEMPORARY - yes=file exists only as long as the filename is assigned (e.g. FILENAME MYFILE TEMP;) no=when the filename is not assigned the file still exists
- FILESIZE – size of the file
- LEVEL – 0 when FILEREF refers to single file or folder, additional rows with level 1-n will exist if FILEREF refers to more than one file or folder
- MODATE – date of last modification

VEXTFL has been used to obtain the path for a file or the current program (e.g. Dilorio & Abolafia, 2004). In some instances, SAS functions allow even more concise access to some of this information. For example, beginning with SAS 9, the following may be used in the SAS Interactive Window to obtain the current program name and path (Tyndall, 2005):

```
%let programname = %sysget(sas_execfilename);
%let fullfile = %sysget(sas_execfilepath);
Footnote1 "&programname";
Footnote2 "&fullfile";
```

The %LET statement is used to create the macro variable PROGRAMNAME and it is set to the value of the environmental variable SAS_EXEXFILENAME by using the %SYSGET function. The value of the macro variable is resolved as the text of a footnote by placing a leading "&" within the double quotes. Both the path and filename of the current program are obtained in similar fashion using the SAS_EXEXFILEPATH environmental variable.

DEFINED MACRO VARIABLES METADATA (EXPLORING MACROS TABLE USING VMACRO VIEW)

Since this paper assumes that the reader has no prior experience with SAS macro, an introduction to the topic in order to explain the metadata is beyond the scope of this paper. Carpenter (1998) provides an excellent book for learning SAS macro, and diTommaso & Hutchison (2003) demonstrate some of the uses of metadata returned from the VMACRO view.

MEMBERS METADATA (EXPLORING MEMBERS TABLE USING VMEMBER VIEW)

SAS uses the term member to refer to a number of different entities that are available through a library, including the familiar data set and view, as well as SAS catalogs. While data sets and catalogs in particular have extensive metadata table(s) and fields devoted exclusively to them (see later sections), the MEMBER table provides consolidated access to some important information for all members in a library.

VMEMBER view returns 7 columns of metadata from the MEMBERS table of the DICTIONARY library. Each record returned by VMEMBER is uniquely identifiable by LIBNAME, MEMNAME, and MEMTYPE fields. LIBNAME stores the name of the library. The MEMNAME field in VMEMBER contains the name of the member (e.g., a data set name), and MEMTYPE identifies the member type (e.g., DATA). The following TABULATE syntax is used to generate the table in Figure 11 in order to summarize the member metadata returned from VMEMBER.

```
proc tabulate data=sashelp.vmember noseps format=10.;
    class libname memtype; table libname all,memtype all/ misstext="0";
    title "vmember records";
run;
```

Figure 11 Summary of Member Metadata Returned by VMEMBER View

	Member Type					
	CATALOG	DATA	ITEMSTOR	MDDB	VIEW	All
	N	N	N	N	N	N
Library Name						
MAPS	0	303	0	0	0	303
SASHELP	88	83	2	2	37	212
SASUSER	4	3	2	0	0	9
WORK	0	1	0	0	0	1
All	92	390	4	2	37	525

Figure 11 shows that SAS members may include not only data sets and views, but also catalogs, item stores, and multidimensional databases (MDDB). VMEMBER also returns a library path (PATH) for each LIBNAME. The following PROC REPORT syntax lists LIBNAME, MEMNAME, MEMTYPE, and PATH fields for SASHELP.SHOES (Figure 12).

```
proc report data=sashelp.vmember nowd;
    columns libname memname memtype path;
    define path /flow width=50;
    where libname = "SASHELP" and memtype = "DATA" and memname in ("SHOES");
format libname memname memtype $10.;
run;
```

Figure 12 Selected Member Metadata Returned by VMEMBER

Library Name	Member Name	Member Type	Pathname
SASHELP	SHOES	DATA	('C:\Program Files\SAS\SASFoundation\9.2\nls\en\SASCFG' 'C:\Program Files\SAS\SASFoundation\9.2\core\sashelp' 'C:\Program Files\SAS\SASFoundation\9.2\genetics\sashelp' 'C:\Program Files\SAS\SASFoundation\9.2\graph\sashelp' 'C:\Program Files\SAS\SASFoundation\9.2\stat\sashelp')

Recall that the SASHELP library is a concatenated library, so in contrast to the LIBNAME table that stores each of the 4 paths as separate records, the MEMBERS metadata stores all the paths in one field with a maximum length of 1024 characters.

VMEMBER has been used to determine the existence of a data set (Varney, 2004), save library data set names (found in MEMNAME) to macro variables (Davis, 2001), and list the data set names by library (Csont,2004). Csont chose to create a format where the values were the names of libraries and the value labels showed a study name, so when he created a list of libraries and data sets using TABULATE, the format showed an informative label.

Since VMEMBER consolidates the library, path, and data set names, it makes a convenient source for using all 3 of these pieces of information. For example, Beakley and McCoy (2004) used VMEMBER in a DATA step to find a data set that had been moved to a new library. They obtain the path of the data set, and assign the path to a library name of their choosing.

DATA SET AND RELATED METADATA

There are 5 data-related metadata tables shown in Table 2: TABLES, VIEWS, COLUMNS, INDEXES and DICTIONARIES. DICTIONARIES and VIEWS contain metadata about the metadata (e.g. some of the information shown in Tables 1 & 2), so this section is limited to exploring TABLES, COLUMNS, and INDEXES using the SASHELP views VTABLE, VCOLUMN, and VINDEX.

DATA SET METADATA (EXPLORING TABLES TABLE USING VTABLE VIEW)

The VTABLE view is used to access metadata from the TABLES table of the DICTIONARY library. The TABLES metadata contains information about SAS data sets. The following PROC CONTENTS lists all the 41 fields (39 in SAS 9.1) of metadata found in the VTABLE. The option VALIDVARNAME=ANY may be necessary to allow the PRINT procedure to run without error. PROC PRINT is used to create the listing of the 7 selected fields shown in Figure 13:

```
proc contents data=sashelp.vtable; run;
options validvarname=any;
proc print data=sashelp.vtable noobs;
var libname memname memtype memlabel nvar nobs modate;
where libname = "SASHELP" and memtype = "DATA" and memname in ("SHOES","RETAIL"); run;
```

Figure 13 Listing of VTABLE Metadata for SASHELP Data Sets RETAIL and SHOES

libname	memname	memtype	memlabel	nvar	nobs	modate
SASHELP	RETAIL	DATA	Retail Sales (Quarterly: 1980Q1-1994Q2)	5	58	11MAR04:09:43:39
SASHELP	SHOES	DATA	Fictitious Shoe Company Data	7	395	11MAR04:09:43:35

Figure 13 shows two metadata records, the first with information about the data set RETAIL and the second about data set SHOES. The columns LIBNAME and MEMNAME store the name of the library and data set, respectively. The LIBNAME field has a maximum length of 8 characters, while the MEMNAME field has a maximum length of 32 characters. The value of "DATA" in the MEMTYPE field identifies the metadata as originating from a data set. MEMTYPE may also have the value of "VIEW" indicating that the SAS member name is a view rather than a data set. The MEMLABEL column stores the label assigned to the data set, NVAR and NOBS stores the number of variables and observations found in each data set, respectively, and MODATE stores the date and time of the last modification of the data set.

The WHERE statement in the PROC PRINT limits the output to specific data sets (i.e. MEMTYPE="DATA" and MEMNAME in ("SHOES","RETAIL")) in the SASHELP library (i.e. LIBNAME="SASHELP"). Without the WHERE statement the output from PROC PRINT would have been much longer. Figure 14 shows an accounting of all the metadata records returned from VTABLE using the following TABULATE procedure:

```
options validvarname=any;
proc tabulate data=sashelp.vtable noseps format=4.0;
       class libname memtype;
       table libname all,memtype all /rts=15 misstext="0";
run;
```

Figure 14 Summary of Metadata Returned from VTABLE

	Member Type		
	DATA	VIEW	All
	N	N	N
Library Name			
MAPS	303	0	303
SASHELP	83	37	120
All	386	37	423

The 423 metadata records returned by VTABLE contains information from data sets and views. There are metadata for 386 data sets and 37 views. All 37 views are found in the SASHELP library, and the data sets are located in 2 different libraries, MAPS and SASHELP. In the SASHELP library, there are 83 data sets, of which 2, RETAIL and SHOES (Figure 13), are explicitly listed with the previous PRINT procedure.

Since the VTABLE view is dynamically created during the procedure, metadata for any newly created data sets are accessible in VTABLE immediately following a procedure or DATA step that created them.

For example, the following syntax is used to add the SHOES data set to the WORK library and the PROC TABULATE syntax produces the output in Figure 15:

```
options obs=0;
proc sort data=sashelp.shoes out=shoes;
by region;
run;
options obs=max;
```

Figure 15 Summary of Metadata Returned from VTABLE, Showing New WORK Metadata

	Member Type		
	DATA	VIEW	All
	N	N	N
Library Name			
MAPS	303	0	303
SASHELP	83	37	120
WORK	1	0	1
All	387	37	424

Figure 15 shows that the total number of metadata records increased by 1 compared with Figure 14 from 386 to 387 and the WORK library is shown to have 1 data set. The following PRINT procedure is used to list the metadata record for the newly created WORK.SHOES by limiting the output metadata to that modified on the current date (Figure 16):

```
proc print data=sashelp.vtable noobs;
var libname memname memtype nvar nobs modate
num_character num_numeric;
where datepart(modate) = date();
run;
```

Figure 16 Listing of VTABLE Metadata for WORK.SHOES

libname	memname	memtype	nvar	nobs	modate	num_ character	num_ numeric
WORK	SHOES	DATA	7	0	13FEB11:10:41:44	3	4

Figure 16 shows that LIBNAME has the value “WORK” and MODATE is the date and time the data set was created by PROC SORT. The WHERE statement in the PROC PRINT is used to list only metadata records returned by VTABLE that are modified on the date the syntax was submitted, thus only a single metadata record is listed from the 424 records available from VTABLE. Since option OBS=0 was set prior to creating SHOES, the data set has no observations which is shown in the NOBS field. Also included in the PROC PRINT are 2 fields returned by VTABLE that are new to SAS 9.2. These fields contain the number of character and numeric variables found in the data set, so out of the 7 variables in SHOES, 3 are character and 4 are numeric.

Just as adding a data set creates an additional VTABLE record, deleting a data set using PROC DELETE removes a record of metadata from VTABLE:

```
proc delete data=work.shoes;
run;
```

It's important to note that if data sets are deleted from the operating system using Windows rather than SAS, then the VTABLE view will return inaccurate metadata. The metadata appears to only be accurate after the library where the data set exists is refreshed through the submission of a LIBNAME statement.

Some documented uses of the TABLES table/VTABLE view are:

- Printing a sample of n observations from every data set in a library, and/or observations by ID having a specific value on another variable (Dilorio & Abolafia, 2004)
- Calculating size of each library using NPAGE (Dilorio & Abolafia, 2004)
- Determining the number observations in any data set and/or using it to determine the existence of a data set (Dilorio & Abolafia, 2004)
- Determining whether a data set is empty (Dilorio & Abolafia, 2004; Varney, 2004)
- Complying with FDA standards, checking that data set labels are less than 40 characters (Teng & Wang, 2006)

VARIABLES FROM EVERY DATA SET METADATA (EXPLORING COLUMNS TABLE USING VCOLUMN VIEW)

The VCOLUMN view is used to access metadata found in the COLUMNS table. The COLUMNS table contains information about variables found in data sets and views. The following PROC CONTENTS is used to list the 18 fields of metadata returned by the VCOLUMN view. The PRINT procedure lists the 7 columns shown in Figure 17 and a WHERE statement limits the records to only those for variables found in SASHELP.SHOES.

```
proc contents data= sashelp.vcolumn;
run;
proc print data=sashelp.vcolumn label noobs;
var libname memname memtype name type label format;
where libname = "SASHELP" and memtype = "DATA" and memname in ("SHOES");
run;
```

Figure 17 Listing of Metadata Returned from VCOLUMN for SASHELP.SHOES

libname	memname	memtype	name	type	label	format
SASHELP	SHOES	DATA	Region	char		
SASHELP	SHOES	DATA	Product	char		
SASHELP	SHOES	DATA	Subsidiary	char		
SASHELP	SHOES	DATA	Stores	num	Number of Stores	
SASHELP	SHOES	DATA	Sales	num	Total Sales	DOLLAR12.
SASHELP	SHOES	DATA	Inventory	num	Total Inventory	DOLLAR12.
SASHELP	SHOES	DATA	Returns	num	Total Returns	DOLLAR12.

Figure 17 shows 7 metadata records one for each variable in the SHOES data set. The listing of 7 variables is consistent with the value of the NVAR field in the SHOES record returned by VTABLE (Figure 16). The first 3 fields of Figure 17 are also found in the VTABLE listing, LIBNAME, MEMNAME, and MEMTYPE.

These fields identify the origin of the variables, and may be used to join or merge the VTABLE and VCOLUMN metadata together to create a single table of metadata. Additional fields of information shown in Figure 17 include: (a) NAME , (b) TYPE, (c) LABEL and (d) FORMAT.

A PROC TABULATE is used to summarize the metadata returned by VCOLUMN (Figure 18)

```
proc tabulate data=sashelp.vcolumn noseps
format=4.0;
class libname memtype;
table libname all,memtype all /rts=15
misstext="0";
run;
```

Figure 18 Summary of Metadata Returned from VCOLUMN

	Member Type		
	DATA	VIEW	All
	N	N	N
Library Name			
MAPS	1940	0	1940
SASHELP	739	278	1017
All	2679	278	2957

There are 2,957 records returned by VCOLUMN. Each record has information on a single variable. There are metadata records for 278 variables returned from SAS views, and 2,679 variables found in data sets. Consistent with VTABLE metadata, all the variables originating from views are found in

SASHELP. Of the 2,679 records having information on variables in data sets, 739 are found in SASHELP data sets, and as listed in Figure 17, 7 of the 739 metadata records are for variables found in SASHELP.SHOES.

A PROC TABULATE is again used with VCOLUMN to summarize the variable level metadata for the SHOES and RETAIL data sets (Figure 19):

```
proc tabulate data=sashelp.vcolumn noseps format=6.0;
    class libname memname memtype type;
    table libname*memname*memtype, type all/rtspace=40 misstext=[label="0"];
    where libname = "SASHELP" and memtype = "DATA" and memname in ("SHOES","RETAIL");
run;
```

Figure 19 shows that counting records within LIBNAME, MEMNAME, and MEMTYPE creates an accurate count of variables found in the data sets RETAIL and SHOES. Seven variables are found in SHOES, 3 are character and 4 are numeric, while all 5 variables in RETAIL are numeric.

Figure 19 Count of Variables by Type Found in RETAIL and SHOES

			Column Type		
			char	num	All
			N	N	N
Library Name	Member Name	Member Type			
SASHELP	RETAIL	DATA	0	5	5
	SHOES	DATA	3	4	7

Various uses of the metadata returned by VCOLUMN view from COLUMNS table include:

- Determining all data sets in a library that have variables in common (Lafler, 2005)
- Identifying inconsistencies in length and type for variables of the same name across data sets in a library (Dilorio & Abolafia, 2004)
- Saving a specific set of variables identified by a naming convention to macro variables and dynamically build syntax that include those variable names (Dilorio & Abolafia, 2004)
- Identifying variable names, name lengths, and label lengths that would be incompatible with SAS version 8 transport format (Dilorio & Abolafia, 2004)
- Determining the existence of a variable(s) (Varney, 2004)
- Renaming specific variables (Mack, 2006)
- Creating an empty copy of a data set (Mack, 2006)
- Counting the number of data sets in a library having each variable. And llisting the data sets, variable names, types, labels and formats appearing in more than one data set (Eberhardt & Brill, 2006)
- Applying a consistent label and format to variables of the same name across all data sets in a library (Eberhardt & Brill, 2006)

INDEXES METADATA (EXPLORING THE INDEXES TABLE USING VINDEX VIEW)

Cates (2002) described a SAS index as acting like an index in a book in order to make specific groups of observations accessible without requiring that every observation be searched. Indexes often improve performance. Raithel (2004) provided an extensive discussion of when and how to use SAS indexes.

The VINDEX view is used with PROC COTENTS and PRINT to explore the 9 fields of metadata returned from the INDEXES table of DICTIONARY. VINDEX returned 52 records (42 in 9.1) associated with data sets in the SASHELP library and the following PROC PRINT was used to list 10 records (Figure 20).

```
proc print data=sashelp.vindex (obs=10);
var libname memname memtype name  idxusage   indxname;
where libname = "SASHELP";
run;
```

Figure 20 First 10 OBS from INDEXES Returned via the VINDEX View

Obs	libname	memname	memtype	name	idxusage	indxname
1	SASHELP	ADOMSG	DATA	MNEMONIC	SIMPLE	MNEMONIC
2	SASHELP	ADOMSG	DATA	MSGID	SIMPLE	MSGID
3	SASHELP	ADSMSG	DATA	MNEMONIC	SIMPLE	MNEMONIC
4	SASHELP	ADSMSG	DATA	MSGID	SIMPLE	MSGID
5	SASHELP	AFMSG	DATA	MNEMONIC	SIMPLE	MNEMONIC
6	SASHELP	AFMSG	DATA	MSGID	SIMPLE	MSGID
7	SASHELP	CLNMSG	DATA	MNEMONIC	SIMPLE	MNEMONIC
8	SASHELP	CLNMSG	DATA	MSGID	SIMPLE	MSGID
9	SASHELP	DESKACT	DATA	objtype	COMPOSITE	objsub
10	SASHELP	DESKACT	DATA	subtype	COMPOSITE	objsub

The LIBNAME, MEMNAME, MEMTYPE, and NAME fields returned by VINDEX contain the same metadata as shown for the COLUMNS table; however, these values do not necessarily define a unique record in the table. When the IDXUSAGE value is COMPOSITE, there is more than on one variable involved in the index. So, for example, the last 2 records document the composite index OBJSUB which has 2 metadata records one for each of 2 variables, OBJTYPE and SUBTYPE, found in the DESKACT data set. SIMPLE indexes have just one record.

There are limited documented uses of the INDEXES metadata. Mack (2006) demonstrated a macro program that used VINDEX to create an index on one or more variables in a data set only after checking the metadata to make sure that an index was not already created using the variables.

WHAT'S IN DICTIONARY (EXPLORING THE DICTIONARY TABLE USING VDCTNRY VIEW)

The DICTIONARY library has 29 metadata tables, 1 of which (named DICTIONAIRES) actually contains metadata about the metadata. The contents of the DICTIONARIES table are available using the VDCTNRY view. PROC CONTENTS shows that the VDCTNRY view returns 10 fields, including MEMNAME that stored the names of the tables in the DICTIONARY library and MEMLABEL that stored up to a 256 character description of each table.

```
proc contents data=sashelp.vdctnry;run;
proc report data=sashelp.vdctnry nowd;
columns memname memlabel;
define memname /group ;
define memlabel /group width =30 flow;
run;
```

PROC REPORT reads the 258 records (191 in SAS 9.1) returned by VDCTNRY and creates one record of output for each table and label in the DICTIONARY library. This information is organized and shown in Tables 1 and 2 at the beginning of this paper.

CATALOGS AND RELATED METADATA

SAS catalog files do not store data, but they store a variety of other SAS entities, the most common of which may be user-defined formats. The following section explores the metadata found in the CATALOGS and FORMATS tables of the DICTIONARY library, by examining the records returned by the VCATLG and VFORMAT views, respectively.

CATALOGS AND CATALOG-SPECIFIC METADATA (EXPLORING CATALOGS TABLE USING VCATALG VIEW)

The VCATALG view returns 10 fields of metadata from the CATALOGS table of DICTIONARY. As a SAS member, a catalog is similar to a data set, in that like a data set that often has many variables, a catalog may often have many entries. Formats are a familiar example of an entry in a catalog. To demonstrate, the following syntax creates formats that SAS automatically stores in a catalog called FORMATS in the WORK library. The VCATALG metadata for the WORK library is listed using PROC PRINT (Figure 21).

```
proc format;
    value gender 1 = "male" 2="Female";
    value grade 0= "Kindergarten" 1 = "First" 2="Second";
run;
proc print data=sashelp.vcatalg;
    where libname = "WORK";
run;
```

Figure 21 Gender and Grade Entries for a Format Catalog

libname	memname	memtype	objname	objtype	objdesc	created	modified	alias	level
WORK	FORMATS	CATALOG	GENDER	FORMAT		23JUL07:11:34:00	23JUL07:11:34:00		0
WORK	FORMATS	CATALOG	GRADE	FORMAT		23JUL07:11:34:00	23JUL07:11:34:00		0

Two records of metadata are listed for the WORK.FORMATS catalog. The MEMTYPE field verifies that FORMATS is in fact a catalog. The two entries in the catalog are shown in the OBJNAME field as GENDER and GRADE. The OBJTYPE field defines GENDER and GRADE as entries of type FORMAT. There were 46 different types of catalog entries returned by the VCATALG view as defined by the number of unique values on the OBJTYPE field. It's beyond the scope of this paper to discuss the various other entries.

The documented uses of the CATALOGS metadata seems limited. Carpenter (1998) used the CATALOGS metadata to obtain a list of members in a library and copy them to another library. PROC CATALOG also allows each user-defined format in a catalog to be assigned up to a 256 character description that may then be obtained from the OBJDESC field in CATALOGS. For example, the following syntax uses PROC CATALOG to assign a description to the GENDER format created above. C=WORK.FORMAT defines the catalog. Gender is referenced in the MODIFY statement as GENDER.FORMAT. The familiar PROC PRINT is used to list the specific metadata in Figure 22.

```
proc catalog c=work.formats;
      modify gender.format (description="My 1=Male 2=Femle Gender Format");
run;
proc print data=sashelp.vcatalg noobs;
var libname  memname  memtype objname objtype  objdesc;
where libname = "WORK";
run;
```

Figure 22 Gender and Grade Entries for a Format Catalog with Gender Description

libname	memname	memtype	objname	objtype	objdesc
WORK	FORMATS	CATALOG	GENDER	FORMAT	My 1=Male 2=Femle Gender Format
WORK	FORMATS	CATALOG	GRADE	FORMAT	

For documentation, format descriptions may be assigned and then harvested from VCATALG view, and such descriptions are not available in the FORMAT DICTIONARY table or through the data set created by PROC FORMAT CNLTOUT=. SAS macro programs are stored by default in a catalog WORK.SASMACR and so a similar tactic may be used to associate descriptions with SAS macro programs (Thornton, 2007).

AVAILABLE FORMATS METADATA (EXPLORING FORMATS TABLE USING VFORMAT VIEW)

The VFORMAT view returns13 fields of metadata from the FORMAT table of DICTIONARY. A PROC TABULATE is used to summarize the metadata records (Figure 23).

```
proc format;
    value $source (default=20) 'B'='(B) SAS'
                               'U' = '(U) SAS'
                               'C' = '(C) User';
run;
proc contents data=SASHELP.vformat;
run;
proc tabulate data=SASHELP.vformat noseps format=10. missing;
    class libname memname  source FMTTYPE;
    table libname*memname*source all,FMTTYPE all/ misstext="0" rtspace=40;
    format source $source.;
run;
```

Figure 23 Summary of Metadata Returned from VFORMAT

			Format Type		
			F	I	All
			N	N	N
Library Name	Member Name	Format Source			
		(B) SAS	116	65	181
		(U) SAS	534	275	809
WORK	FORMATS	(C) User	1	0	1
All			651	340	991

Unlike the previous tables with LIBNAME and MEMNAME fields, VFORMAT may return missing values for these fields. For example, Figure 23 shows that LIBNAME and MEMNAME only have values for the user defined format $SOURCE that was saved in the WORK.FORMATS catalog. The majority of the metadata are SAS defined FORMATS (FMTTYPE=F) and INFORMATS (FMTTYPE=I). A PROC PRINT is used to show some of the types of records returned by VFORMAT.

```
proc print data=sashelp.vformat noobs;
    var libname memname fmtname fmttype source minw mind maxw maxd defw defd;
    where libname="WORK" or fmtname in ("MMDDYY","MINGUO");
run;
```

Figure 24 Listing of Selected VFORMAT Records

libname	memname	fmtname	fmttype	source	minw	mind	maxw	maxd	defw
WORK	FORMATS	$SOURCE	F	C	0	0	0	0	0
		MINGUO	F	U	1	0	10	0	8
		MINGUO	I	U	6	0	10	0	6
		MMDDYY	F	B	2	0	10	9	8
		MMDDYY	I	B	6	0	32	31	6

The first metadata record in Figure 24 shows information about the user defined format $SOURCE which is confirmed to be a format (FMTTYPE=F) and created by a user (SOURCE=C). According to SAS Support (see references FAQ # 3987) a SORUCE=C indicates a user-defined format/informat and SOURCE=B refers a SAS defined format/informat; SOURCE=U was not mentioned. SOURCE=U also appears to be a SAS defined format/informat. The only difference appears to be that SOURCE=U formats/informats have a value for the PATH field (not shown in Figure 24).

The documented uses for the FORMATS metadata seems limited. SAS Support (see references FAQ # 3987) provides a macro program that uses FORMATS to determine the existence on a format or informat name in the metadata. Crawford (2006) uses FORMAT metadata to produce a list of perspective informat that could be used to read a variable. He uses VFORMAT in a DATA step to apply all existing informat to a variable value and lists in the LOG those informat that are applied without error.

CONSTRAINTS METADATA

Constraints metadata does not exist until it is created, so this section will introduce and created example constraints in order to examine the metadata. Franklin and Jensen (2000) defined constraints as "…a set of data validation rules that you can specify to restrict the data values accepted into a SAS file." More commonly, DATA step syntax is used to enforce such data integrity rules. For example, the following syntax creates the SHOES data set where observations are not allowed to have RETURNS greater than $20,000.

```
data shoes;
set sashelp.shoes;
if returns >=. and returns <=20000 then output shoes;
run;
```

Data integrity may also be enforced by CHECK constraints using PROC DATASETS, and while the syntax is a little longer and more complex, metadata is created by SAS to document the process. Documentation is extremely important in larger projects, and using CHECK constraints allows the syntax and documentation to be accomplished at the same time. The following syntax shows how rules may be enforced using PROC DATASETS and a CHECK constraint.

```
❶ options obs=0;
    data shoes;
          set sashelp.shoes;
    run;
options obs=max;
proc datasets library=work;
          ❷ modify shoes;
          ❸ ic create c_returns1=check(where=(returns >=. and returns <= 20000))
          ❹ message = "returns over $20,000" msgtype=user;
          ❺ ic create c_sales1=check(where=(sales > returns))
          message = "sales <= returns" msgtype=user;
          run;
          ❻ append base=shoes data=sashelp.shoes;
          run;
    quit;
```

❶ A constraint cannot be defined if data violates the constraint, so I create an empty data set from SASHELP.SHOES using the OBS=0 OPTION (Gerlach, 2005).

❷ MODIFY SHOES starts a block of statements used to modify shoes, in this case, to add a constraint.

❸ The integrity constraint statement begins with IC and creates a CHECK constraint named C_RETURNS1. The constraint rule is defined through CHECK (WHERE=).

❹ The MESSAGE option is used along with option MSGTYPE=USER to define a custom message with a maximum of 250 characters. The message is stated to identify observations that do not comply with the constraint.

❺ A second IC statement creates C_SALES1 constraint. This constraint is included to illustrate that cross-variable integrity may also be enforced using CHECK constraints. In this case, only observations with SALES greater than RETURNS are allowed.

❻ APPEND puts the observations from SASHELP.SHOES that pass the constraint criteria into WORK.SHOES.

The LOG shows that 391 of the 395 observations in SASHELP.SHOES were added to SHOES and a warning message is issued: “WARNING: Returns over $20,000 (Occurred 4 times.)”

The most convenient access to the constraint metadata is through PROC CONTENTS (Figure 25).

```
proc contents data=shoes out2=m_contents_constraints;
run;
proc print data=m_contents_constraints noobs;
var libname member name type message where;
run;
```

Figure 25 Constraint Metadata Obtained from PROC CONTENTS OUT2=

Libname	Member	Name	Type	Message	Where
WORK	SHOES	c_returns1	Check	Returns over $20,000	(Returns>=. and Returns<=20000)
WORK	SHOES	c_sales1	Check	Sales <= Returns	Sales>Returns

The OUT2 option of PROC CONTENTS (Jensen & Franklin, 2000) captures metadata data for the newly created constraints. LIBNAME and MEMBER fields identify the origin of each constraint. The VCNCOLU returns both the constraint and the variables involved in the constraint from the CONSTRAINT_COLUMN_USAGE table of DICTIONARY (Figure 26).

```
proc print data=sashelp.vcncolu;
var table_catalog table_name
column_name constraint_name;
run;
```

Figure 26 Constraints and Variables from SASHELP.VCNCOLU

table_ catalog	table_ name	column_ name	constraint_ name
WORK	SHOES	Returns	c_returns1
WORK	SHOES	Returns	c_sales1
WORK	SHOES	Sales	c_sales1

Table 3 shows the key fields for the 4 metadata tables that SAS uses to track CHECK constraint information.

Table 3 Key fields in CHECK Constraint Metadata Tables

DICTIONARY	Constraint Field	Library Field	Data Set Field	Variable Field
SASHELP.VCHKCON via CHECK_CONSTRAINTS	CONSTRAINT_NAME			
SASHELP.VTABCOM via TABLE_CONSTRAINTS	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_NAME	
SASHELP.VCNTABU via CONSTRAINT_TABLE_USAGE	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_NAME	
SASHELP. VCNCOLU via CONSTRAINT_COLUMN_USAGE	CONSTRAINT_NAME	TABLE_CATALOG	TABLE_NAME	COLUMN_NAME

The first view, VCHKCON returns the list of CHECK constraints along with the syntax shown in the WHERE variable of Figure 25. The constraint names are found in the CONSTRAINT_NAME field. There were no fields identifying the library or data set for the constraint. Few fields are returned by the second and third views and very little information was found regarding their purpose.

There seem to be few documented uses of the constraint metadata. Thornton (2007, 2010) demonstrated a specific use of constraints and the integration of metadata for documentation.

OTHER METADATA

This category of metadata includes the DESTINATIONS, STYLES, FUNCTIONS and REMEMBER DICTIONARY tables. No information was found on the REMEMBER table. The metadata from the other tables is explored using the SASHELP VDEST, VSTYLE, and VFUNC views.

DESTINATION METADATA (EXPLORING THE DESTINATIONS TABLE USING VDEST VIEW)

The VDEST view returns the fields DESTINATION and STYLE from the DESTINATION table. The term destination refers to where output is sent. The default destination is the listing or output window. I used a PRINT procedure to first list the metadata for the default destination (Figure 27).

```
options formdlim=' ' nodate nonumber;
ods listing;
title1 'VDEST Metadata for Listing
Destination';
proc print data=sashelp.vdest noobs;
run;
ods pdf style=ocean;
title1 'VDEST Metadata for Listing and
PDF Destination';
proc print data=sashelp.vdest noobs;
run;
ods pdf close;
title1 'VDEST Metadata for Listing
Destination';
proc print data=sashelp.vdest noobs;
run;
options formdlim='' date number;
```

Figure 27 Destinations Metadata

VDEST Metadata for Listing Destination

destination	style
LISTING	Listing

VDEST Metadata for Listing and PDF Destination

destination	style
LISTING	Listing
PDF	Ocean

VDEST Metadata for Listing Destination

destination	style
LISTING	Listing

The ODS PDF STYLE=OCEAN statement caused the output from the PROC PRINT to go to both the LISTING and a PDF, so the second PROC PRINT from VDEST shows both a LISTING and a PDF destination. The ODS PDF CLOSE statement removed PDF as a destination for output, and the third PROC PRINT reflects that change. I have not found any uses for DESTINATION metadata, but it could clearly be used to conditionally open/close destination and selectively apply styles depending on the type of destination.

STYLES METADATA (EXPLORING THE STYLES TABLE USING VSTYLE VIEW)

The VSTYLE view returns 4 fields of metadata from the STYLES table of DICTIONARY: LIBNAME, MEMNAME, STYLE, and CRDATE. On my system, the view returns 15 user-created styles that I had saved in the SASUSER.TEMPLATE. The view also returns 54 SAS-defined styles found in SASHELP.TMPMST. The OCEAN style that I used in the previous example is among the 54 styles (Figure 28),

```
proc print data=sashelp.vstyle
noobs;
where style="Styles.Ocean";
run;
```

Figure 28 Selected Record Returned from VSTYLE

libname	memname	style	crdate
SASHELP	TMPLMST	Styles.Ocean	25MAR10:03:19:49

There does not seem to be many documented uses of STYPES metadata. On use might be to check to see if a style exists before creating one with the same name.

FUNCTIONS

The VFUNC view returns 7 fields (Figure 29) from the FUNCTIONS table of DICTIONARY, and 859 records of metadata.

Figure 29 Seven Fields Returned the VFUNC view from the FUNCTIONS table

#	Variable	Type	Len	Flags	Label
1	source	Char	1	P--	Format Source
2	fncname	Char	32	P--	Function name
3	minarg	Num	8	P--	Minimum args to function
4	maxarg	Num	8	P--	Maximum args to function
5	fnctype	Char	1	P--	Function type
6	fncargs	Num	8	P--	Argument attributes
7	fncprod	Char	1	P--	Function implementation type

The records returned by the view are uniquely defined by FNCNAME and FNCTYPE. FNCNAME contains the name of a function and FNCTYPE has the values of “B, C, or N.” I believe these indicate whether a function is bitwise, character, or numeric function, and there are 48 function names that appear in more than one function type. The FUNCTION table is new to SAS 9.2, and I have not found document uses of it. An obvious use is to try and learn about functions by looking at the list of functions of various types, but even that use is of less utility without a description of the function, its arguments, and what the function returns.

CONCLUSION

A great deal of DICTIONARY metadata is available through SASHELP views and accessible using common procedures and DATA step syntax. These tools may be used to explore and to learn the metadata that SAS makes available. Understanding the metadata is the first step in the critical thinking necessary to exploit the metadata. This paper demonstrated or cited a number of techniques for leveraging the metadata with procedures and DATA step syntax, and there are many papers available to learn the power of macro and SQL techniques for leveraging metadata.

REFERENCES

Beakley S.& McCoy S.(2004). 'Dynamic SAS programming techniques, and how not to create job security,' Proceedings of the Twenty Ninth Annual SAS® Users Group International Conference, Montréal, Québec, Canada

Beatrous S.& Clifford B.(1998). 'Sometimes you get what you want: SAS I/O enhancements for version 7,' Proceedings of the Twenty Third Annual SAS® Users Group International Conference, Nashville, TN

Carpenter A. (1998). 'Advanced macro topics: Utilities and examples,' Proceedings of the Twenty Third Annual SAS® Users Group International Conference, Nashville, TN

Cates R. (2002). 'What's in a name: Describing SAS file types,' Proceedings of the Twenty Seventh Annual SAS® Users Group International Conference, Orlando, FL

Choate P.& Martell C.(2006). 'De-Mystifying the SAS® LIBNAME engine in Microsoft Excel: A practical guide,' Proceedings of the Thirty First Annual SAS® Users Group International Conference, San Francisco, CA

Crawford, P. (2006). _____. VIEWS International SAS Programmer Community Issues 35 3rd Quarter

Csont, W. (2004). 'Swiss Army Knife of Listing All Filenames For Multiple Libraries,' Proceedings of the 2004 Annual Pharmaceutical Industry SAS® Users Group Conference,San Diego, CA.

Davis M. (2001). 'You can look it up: An introduction to SASHELP Dictionary Views,' Proceedings of the Twenty Sixth Annual SAS® Users Group International Conference, Long Beach, CA

Dilorio F.& Abolafia J. (2004). 'Dictionary tables and views: Essential tools for serious applications,' Proceedings of the Twenty Ninth Annual SAS® Users Group International Conference, Montréal, Québec, Canada

Dilorio F. & Abolafia J. (2005). 'The design and use of metadata: Part fine art, part black art,' CodeCrafters, Inc. (http://www.codecraftersinc.com/).

diTommaso D.& Hutchison F.(2003). 'Taking control of macro variables,' Proceedings of the Twenty Eighth Annual SAS® Users Group International Conference, Seattle, WA

Eberhardt P.& Brill I. (2006). 'How do I look it up if I cannot spell it: An introduction to SAS dictionary tables,' Proceedings of the Thirty First Annual SAS® Users Group International Conference, San Francisco, CA

Franklin G.& Jensen A.(2000). 'Integrity constraints and audit trails working together,' Proceedings of the Twenty Fifth Annual SAS® Users Group International Conference, Indianapolis, IN

Gerlach J. (2005). 'A better perspective on SASHELP views,' Proceedings of the 2005 Annual Pharmaceutical Industry SAS® Users Group Conference, Phoenix, AZ

Heaton E. (2003). 'SAS® Systems Options are your friends,' Proceedings of the Twenty Eighth Annual SAS® Users Group International Conference, Seattle, WA

Kolosova, T. & Berestizhevsky, S. (1996). 'Table-Driven Strategies for Rapid SAS Applications Development' SAS Publishing, Cary, NC.

Lafler K.P. (2005). 'Exploring DICTIONARY tables and SASHELP views.,' Proceedings of the 2005 Annual Western Users of SAS® Software Conference, San Jose, CA

Mack C. (2006). 'The wealth of information found in the SASHELP data dictionary views and how to use it,' Proceedings of the 2006 Annual Pacific Northwest SAS® Users Group Conference, Seaside, WA

Raithel M. (2004). 'Creating and exploiting SAS® indexes,' Proceedings of the Twenty Ninth Annual SAS® Users Group International Conference, Montréal, Québec, Canada

SAS Support FAQ # 3987 (http://support.sas.com/faq/039/FAQ03987.html) – Determining whether a format is available for use

Teng C.& Wang W.(2006). 'Simple ways to use PROC SQL and SAS DICTIONARYTABLES to verify data structure of the electronic submission data sets,' Proceedings of the 2006 Annual Pharmaceutical Industry SAS® Users Group Conference, Bonita Springs, FL

Thornton, S. P. (2007) 'SAS DICTIONARY: Step by Step.' Paper in the Proceedings of the 15th Annual Western Users of SAS® Software Conference, San Francisco, CA

Thornton, S. P. (2008). 'Documenting SAS® Macro Programs using CATALOGS.' Paper presented at the Sixteenth Annual Western Users of the SAS® Software Conference, Universal City, CA.

Thornton, S. P. (2010). Check and Document Data Integrity with PROC DATASETS. Paper presented at the Eighteenth Annual Western Users of the SAS® Software Conference, San Diego, CA.

Tyndall R. (2005). 'Give your macro code an extreme makeover: Tips for even the most seasoned macro programmer,' Your SAS Technology Report. SAS Institute, Inc., Philadelphia, NC

Varney B. (2004). 'Using meta and project data for data driven programming,' Proceedings of the Twenty Ninth Annual SAS® Users Group International Conference, Montréal, Québec, Canada

Zhang J., Chen D.& Wong T.(2003). 'Metadata application on clinical trial data in drug development,' Proceedings of the Twenty Eighth Annual SAS® Users Group International Conference, Seattle, WA

ACKNOWLEDGMENTS

Thanks to Cyndi Williamson, Mary McCracken and Ethan Miller for their time in reviewing the paper.

RECOMMENDED READING

Abolafia J. (2005). 'What would I do without PROC SQL and the macro language,' Proceedings of the 2005 Annual Western Users of SAS® Software Conference, San Jose, PA

Boling J. (1997). 'SAS data views: A virtual view of data,' Proceedings of the Twenty Second Annual SAS® Users Group International Conference, San Diego, CA

Carpenter A. (1998). 'Carpenter's complete guide to the SAS Macro language,' SAS Institute, Inc., Cary, NC

Dilorio F. (2005). 'Rules for Tools - -The SAS Utility Primer,' Proceedings of the Thirtieth Annual SAS® Users Group International Conference, Philadelphia, PA

Fickbohm D. (2006). 'Using the DATASETS Procedure,' Proceedings of the Thirty First Annual SAS® Users Group International Conference, San Francisco, CA

Hamilton J. (1998). 'Some Utility Applications of the Dictionary Tables in PROC SQL,' Proceedings of the 1998 Annual Western Users of SAS® Software Conference, Oakland, CA

Lafler K.P. (2010). 'Exploring DICTIONARY Tables and SASHELP Views,' Proceedings of the 2010 SAS® Global Forum Conference, Seattle, WA

Lund P. (2002). 'A quick and easy data dictionary macro,' Proceedings of the Twenty Seventh Annual SAS® Users Group International Conference, Orlando, FL

Ray R. (2002). 'Save time today using SAS® views,' Proceedings of the Twenty Seventh Annual SAS® Users Group International Conference, Orlando, Fl

Shoemaker J. (2001). 'Eight PROC FORMAT Gems,' Proceedings of the Twenty Sixth Annual SAS® Users Group International Conference, Long Beach, FL

Troxell J.K. & Chen C. (2003). 'Invisible environmental awareness techniques,' Proceedings of the 2003 Annual Pharmaceutical Industry SAS® Users Group Conference, Miami Beach, FL

Ye Y. (2000). 'Using SAS functions in data steps,' Proceedings of the 2000 Annual Pharmaceutical Industry SAS® Users Group Conference, Seattle, WA

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

S. Patrick Thornton Ph.D., Sr. Social Science Programmer
SRI International, Center for Education and Human Services
Phone: 650 859-5583
Fax: 650 859-2861
Email: patrick.thornton@sri.com
Web: www.sri.com