SAS® DICTIONARY: Step by Step
Patrick Thornton, SRI International, Menlo Park, CA

ABSTRACT
This paper offers a step by step visual review of SAS® 9.1 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, as well as external file and library paths, is visually explored using common procedures and SASHELP views. The paper demonstrates that DICTIONARY is accessible and valuable to beginning SAS users, reviews in plain terms many uses of DICTIONARY, shows some alternative techniques for obtaining metadata when applicable, and directs users to papers that featured more advanced techniques for using DICTIONARY.

INTRODUCTION
The SAS DICTIONARY library contains metadata. Metadata is often defined as data about data. It is valuable information about data, such as data set names, that is intelligently structured and stored as data; however, metadata may also be information about the broader SAS programming environment, such as options, formats, file names, library names and paths. In a still wider definition, Zhang, Chen, and Wong (2003) refer to metadata as structured information about an “object or resource,” while describing the ultimate utility of metadata as a “management and resource discovery tool,” such that the promise of metadata is to enable information to be more effectively found and used. The potential of metadata to create organization from chaos is a common theme. Dilorio and Abolafia (2005) illustrated the need for metadata by describing the potentially chaotic context of many SAS projects as having: (a) an overwhelming number of files and file formats, (b) important specifications in a variety of inconvenient and/or programmatically inaccessible formats, such as Word, (c) duplication of syntax, such as syntax to establish libraries in every program, (d) changing requirements, and (e) repetitive output requirements, such as the generation of identical tables having only different variables and titles.

This paper was geared to beginning and intermediate users of SAS and generally used common procedures and data STEPS to explore the content of DICTIONARY tables. Many papers on DICTIONARY by SAS users rely heavily on PROC SQL and Macro programming to demonstrate the use DICTIONARY tables. While these techniques may be preferred and/or are essential for many uses of the metadata, this paper focused on a visual examination of the data sets and variables, by convention called tables and fields, found in the DICTIONARY library. The number one priority of this paper was to encourage exploration and familiarity with the DICTIONARY tables and their content. By the end of the paper you should have a greater understanding of the DICTIONARY tables, feel confident in exploring the tables yourself, and be aware of the papers that discussed the details of advanced techniques for using metadata.

INTRODUCTION TO SAS DICTIONARY LIBRARY
An excellent introduction to accessing and using the tables of metadata found in the DICTIONARY library was had by reading the following papers: Lafler, (2005), Eberhardt & Brill (2006), Mack (2006), and Dilorio & Abolafia (2004). 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 9.1, the DICTIONARY and SASHELP libraries were 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 by SAS in the SASHELP library (Davis, 2001; Gerlach, 2005; Mack, 2006). The DICTIONARY library had 22 metadata tables, 1 of which (named DICTIONAIRES) actually contained metadata about the metadata. The contents of the DICTIONAIRES table were available using the VDCTNRY view. Using the VDCTNRY view the following syntax demonstrates the typical procedures that will be used to explore all the views in this paper, and the PROC REPORT was used to generate the description and dictionary table names shown in the Table 1.

```sas
proc contents data=sashelp.vdctnry;run;
proc sort data=sashelp.vdctnry out=d nodupkey;
by memname;
run;
proc report data=d nowd;
columns memname memlabel;
define memlabel /width =30 flow;
run;
```
PROC CONTENTS showed that the VDCTNRY view returned 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. VDCTNRY returned 191 records, one for each table and field (NAME) in the DICTIONARY library, so PROC SORT was used to create the data set D that contained a unique list of table names found in MEMNAME. PROC REPORT was used to show the 22 table names and descriptions in the output window and the information was featured in Table 1.

Table 1 organizes the 22 metadata tables found in the DICTIONARY library into 6 categories (the first column in the table) and serves as an outline for this paper. The second column shows a short description of the contents of each table. The third column lists the name of the DICTIONARY table and the last column shows the name of the SASHELP view that was used in this paper to access the DICTIONARY tables through data STEPS and procedures.

<table>
<thead>
<tr>
<th>Category</th>
<th>DESCRIPTION</th>
<th>DICTIONARY Table</th>
<th>SASHELP VIEWS*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAS Session</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS Options</td>
<td>OPTIONS**</td>
<td>OPTION</td>
<td></td>
</tr>
<tr>
<td>SAS/Graph Options</td>
<td>GOPTIONS**</td>
<td>VGOPT</td>
<td></td>
</tr>
<tr>
<td>TITLE Statements</td>
<td>TITLES</td>
<td>VTITLE</td>
<td></td>
</tr>
<tr>
<td>Defined macro variables</td>
<td>MACROS</td>
<td>VMACRO</td>
<td></td>
</tr>
<tr>
<td>LIBNAME Information</td>
<td>LIBNAMES</td>
<td>VLIBMAM</td>
<td></td>
</tr>
<tr>
<td>Available Engines</td>
<td>ENGINES</td>
<td>VENGINED</td>
<td></td>
</tr>
<tr>
<td>Files defined in FILENAME</td>
<td>EXTFILES</td>
<td>VEXTFL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statements, or implicitly</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Members</strong></td>
<td>Tables, Catalogs, and Views</td>
<td>MEMBERS</td>
<td>VMEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Set and Related Metadata</strong></td>
<td>Table and Table-Specific Information</td>
<td>TABLES</td>
<td>VTABLE</td>
</tr>
<tr>
<td>View and View-Specific Information</td>
<td>VIEWS</td>
<td>VVIEW</td>
<td></td>
</tr>
<tr>
<td>Columns from every table</td>
<td>COLUMNS</td>
<td>VCOLUMN</td>
<td></td>
</tr>
<tr>
<td>DICTIONARY tables and their columns</td>
<td>DICTIONARIES</td>
<td>VDCTNRY</td>
<td></td>
</tr>
<tr>
<td>Indexes</td>
<td>INDEXES</td>
<td>INDEX</td>
<td></td>
</tr>
<tr>
<td><strong>Catalogs and Related Metadata</strong></td>
<td>Catalog and Catalog-Specific Information</td>
<td>CATALOGS</td>
<td>VCATALG</td>
</tr>
<tr>
<td>Available Formats</td>
<td>FORMATS***</td>
<td>VFFORMAT</td>
<td></td>
</tr>
<tr>
<td>Styles</td>
<td>STYLES</td>
<td>VSTYLE</td>
<td></td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Check Constraints</td>
<td>CHECK_CONSTRAINTS</td>
<td>VCHKCON</td>
</tr>
<tr>
<td>Referential Constraints</td>
<td>REFERENTIAL_CONSTRAINTS</td>
<td>VREFCON</td>
<td></td>
</tr>
<tr>
<td>Table Constraints</td>
<td>TABLE_CONSTRAINTS</td>
<td>VTABLECON</td>
<td></td>
</tr>
<tr>
<td>Constraint table usage</td>
<td>CONSTRAINT_TABLE_USAGE</td>
<td>VCNTABU</td>
<td></td>
</tr>
<tr>
<td>Constraint column usage</td>
<td>CONSTRAINT_COLUMN_USAGE</td>
<td>VCNCOLU</td>
<td></td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>Remember information</td>
<td>REMEMBER****</td>
<td>VREMemb</td>
</tr>
</tbody>
</table>

*Note a few more view provide access to some of the tables: VSACCES, VSCATLG, VSLIB, VSTABLE, VSTABVW, and VSVIEW
**VALLOPT view may be used to access both OPTIONS & GOPTIONS
***VCFORMAT view may be used to access only the user-created SAS formats
**** No information was found for the REMEMBER table; the table will not be discussed in this paper.

The following sections explored the metadata tables shown in Table 1 using the SASHELP views shown in last column. Many of the papers featuring DICTIONARY demonstrated powerful and sophisticated SAS macro and PROC SQL syntax to access and use the metadata (Dilorio & Abolafia, 2004; Varney, 2004). While these techniques offer distinct advantages, SASHELP views provide an excellent opportunity to learn the content of the metadata with familiar procedural and data STEP syntax (Davis, 2001; Gerlach, 2005; Mack, 2006). It is not necessary to understand the intricacies of SAS views to follow their use in this paper; in fact, an advantage of using views is that the intricacies are hidden (Boling, 1997).

**SAS SESSION METADATA**

There were 6 tables of metadata shown in Table 1 that contained SAS session information: OPTIONS, GOPTIONS, TITLES, MACROS, LIBNAMES, ENGINES, and EXTFILES. The following section explored the metadata using each
of the corresponding views listed in the last column of the table: VOPTION, VGOPC, VTITLE, VMACRO, VLIBNAM, VENGINE, and VEXTFL. The OPTIONS and GOPTIONS metadata were discussed together, and MACROS metadata was not fully covered due to the assumption that the reader is unfamiliar with the macro language.

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

The SASHELP.VOPTION view returned metadata records from the OPTIONS table of the DICTIONARY library. PROC CONTENTS was used to obtain a list of fields available through the VOPTION view, and PROC PRINT was used to create a quick listing of metadata records.

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

PROC CONTENTS showed that VOPTION returned metadata on the fields of OPTNAME, OPTTYPE, SETTING, OPTDESC, LEVEL, and GROUP; however, since views do not contain data, PROC CONTENTS did not report the number of observations that was returned from the view. While views do not contain data, they do return data to a data step or procedure, so the PRINT procedure was used to limit the number of records to the first 10. When exploring SASHELP views it is always a good idea to limit the metadata records returned by a view, especially because the DICTIONARY metadata is gathered in real-time, so the simple act of printing a view may take some time. As it turns out though, the VOPTION view only returned 326 records with 6 fields, and all the records were quickly listed in the OUTPUT window. These techniques were used throughout this paper to explore all the metadata returned from the views.

An examination of the fields and their contents on the first 10 observations returned by VOPTION provided valuable hints about how to further explore the metadata. The following syntax produced a summary of the content (Figure 1).

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

**Figure 1. Summary of the Option Groups, by Level and Type**

<table>
<thead>
<tr>
<th>Option Location</th>
<th>Host</th>
<th>Portable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optname</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Opttype</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Optdesc</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Setting</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The options metadata had a record for each of the options available in the SAS session. Each option was 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.

The OPTNAME field contained the name of the options available in SAS. For example, the following PROC REPORT was used to list metadata for the DATE option:

```plaintext
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;

**Figure 2. VOPTION Metadata for the DATE Option**

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Option Type</th>
<th>Option Description</th>
<th>Option Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>boolean</td>
<td>Print date and time on top of each page of SAS log and procedure output</td>
<td>DATE</td>
</tr>
</tbody>
</table>

The option name, type, description, and setting for the DATE option were listed in Figure 2. PROC REPORT, rather than the PRINT procedure was used to produce the listing in Figure 2 because the OPTDESC field returned by VOPTION had a length (256 characters) that exceeded the maximum for PROC PRINT. In the example above, PROC REPORT was functionally very similar to PROC PRINT, where the COLUMNS statement determined the order of the fields in the listing, synonymous to a VAR statement, but the DEFINE statement allowed field widths to be controlled and FLOW allowed the text to wrap. The DATE option has an OPTTYPE value of "Boolean"; the DATE option may either have a value of DATE or NODATE for the SETTING field. As shown in Figure 1, some option settings may be character or numeric. In order to demonstrate the real-time information provided by DICTIONARY, the following syntax used an OPTION statement to change the setting to NODATE and the previous REPORT procedure to generate Figure 3.

```sas
option nodate;
```

**Figure 3. VOPTION Metadata After Setting NODATE**

The metadata returned by VOPTION immediately reflected the change in option setting; the SETTING field changed to the value NODATE (Figure 3). Heaton (2003) was a good source of information on SAS options. VOPTION metadata has been exploited by capturing option metadata prior to changes and using it to reset the SAS session to the original options 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 syntax, the following is a more basic syntax designed to illustrate the use of metadata.

```sas
proc sort data=sashelp.voption out=m_ooption; by optname; run;
data m_ooption;
length keyword keyword2 $1060.;
kward = getoption(optname,'keyword');
rune; 
option nodate nocenter pageno=100;
proc sort data=sashelp.voption out=m_coption; by optname; run;
data _null_; 
merge m_ooption m_coption (keep=optname setting rename=(setting=csetting));
by optname; 
if setting ne csetting then do;
    put "Changed Setting: " optname $10. setting $50.;
call execute("option "||trim(keyword)||";");
end; 
run;
```

1. PROC SORT was used to create a data set M_OOPTION to store the current option settings returned by VOPTION. The GETOPTION function was used (Dilorio & Abolafia, 2005) to set the value of the KEYWORD field to the syntax needed in an OPTION statement. 2. Options were changed using an OPTION statement to demonstrate how a program might alter existing settings. 3. The new option settings were returned from VOPTION and saved to M_COPTION data set. 4. The original option settings and the new option settings were compared by merging M_OOPTION and M_COPTION by OPTNAME. In the data STEP merge, only the OPTNAME and SETTING fields of M_COPTION were kept and the SETTINGS field was renamed to CSETTINGS. 5. When the value of original option settings (SETTINGS) was different from the new options settings (CSETTING), the KEYWORD field having the original option statement syntax was used to reset the option. 6. CALL EXECUTE was used to transform the metadata to syntax, and the following shows a portion of the LOG output that was generated:

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

```sas
n
PROC SORT was used to create a data set M_OOPTION to store the current option settings returned by VOPTION. The GETOPTION function was used (Dilorio & Abolafia, 2005) to set the value of the KEYWORD field to the syntax needed in an OPTION statement. Options were changed using an OPTION statement to demonstrate how a program might alter existing settings. The new option settings were returned from VOPTION and saved to M_COPTION data set. The original option settings and the new option settings were compared by merging M_OOPTION and M_COPTION by OPTNAME. In the data STEP merge, only the OPTNAME and SETTING fields of M_COPTION were kept and the SETTINGS field was renamed to CSETTINGS. When the value of original option settings (SETTINGS) was different from the new options settings (CSETTING), the KEYWORD field having the original option statement syntax was used to reset the option. CALL EXECUTE was used to transform the metadata to syntax, and the following shows a portion of the LOG output that was generated:

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

```sas
```
This is the one and only place in this paper where such an advanced technique as CALL EXECUTE was discussed, and it was included only to demonstrate in the simplest case the power of metadata in generating and controlling syntax. For each observation of the data STEP where the new option setting was different from the original option setting, CALL EXECUTE produced the OPTION statement from the literal value of “option “concatenated with the value of KEYWORD and an ending semicolon.

**TITLE STATEMENTS METADATA (EXPLORING TITLES TABLE USING VTITLE VIEW)**

Additional SAS session metadata includes the TITLES table. The records of the TITLES table were accessed using VTITLE. The CONTENTS procedure was used to list the fields of metadata returned from the VTITLE view and the PRINT procedure created the listing shown in Figure 4.

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

**Figure 4. VTITLE Metadata**

<table>
<thead>
<tr>
<th>Obs</th>
<th>type</th>
<th>number</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>1</td>
<td>The SAS System</td>
</tr>
</tbody>
</table>

Figure 4 shows that the default title created by the SAS System was stored in the TEXT field. Since TITLES metadata data may contain footnote information, the TYPE field had the value of “T” indicating that the value of TEXT was a title. A value of “F” for the TYPE field would indicate that the value of TEXT was a footnote. The NUMBER field identified the level of title. Figure 5 shows the VTITLE metadata produced by the following syntax.

```sas
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 5. Example Metadata Returned from VTITLE**

<table>
<thead>
<tr>
<th>type</th>
<th>number</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1</td>
<td>First Title</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>Second Title</td>
</tr>
</tbody>
</table>

The FORMDLIM option was set to a blank space which allowed the multiple PROC PRINT listings to appear on the same page of the output window. The TITLE1 statement was used to create ”First Title” and the first PROC PRINT output in Figure 5 showed the new title in the metadata returned by VTITLE. The TITLE2 statement was used to create the ”Second Title” and the second PRINT again showed that the new title was added as a title and in the metadata. Last, the TITLE2 statement was used where no title was specified and the final listing in Figure 5 showed that the second title was removed. Changes in session footnotes using the FOOTNOTE statement resulted in analogous changes in the metadata returned from VTITLE.

**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 was beyond the scope of this paper. Carpenter (1998) provided an excellent book for learning SAS macro, and diTommaso & Hutchison (2003) demonstrated the use of metadata returned from the VMACRO view.

**LIBNAME INFORMATION METADATA (EXPLORING LIBNAMES TABLE USING VLIBNAM VIEW)**

The VLIBNAM view of the SASHELP library was used to explore the LIBNAMES table metadata that contained information about libraries known to the SAS session. PROC CONTENTS listed the 10 fields returned by VLIBNAM
and PROC PRINT showed the 3 columns of metadata in Figure 6.

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

Figure 6. Listing of VLIBNAM Metadata for Libraries MAPS and SASHELP

<table>
<thead>
<tr>
<th>libname</th>
<th>level</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP</td>
<td>1</td>
<td>C:\Program Files\SAS\SAS 9.1\nl\en\SASCFG</td>
</tr>
<tr>
<td>SASHELP</td>
<td>2</td>
<td>C:\Program Files\SAS\SAS 9.1\core\sashelp</td>
</tr>
<tr>
<td>SASHELP</td>
<td>3</td>
<td>C:\Program Files\SAS\SAS 9.1\graph\sashelp</td>
</tr>
<tr>
<td>SASHELP</td>
<td>4</td>
<td>C:\Program Files\SAS\SAS 9.1\stat\sashelp</td>
</tr>
<tr>
<td>MAPS</td>
<td>0</td>
<td>C:\Program Files\SAS\SAS 9.1\maps</td>
</tr>
</tbody>
</table>

Figure 6 showed 5 metadata records associated with the MAPS and SASHELP libraries. The LIBNAME field stored the name of the library and PATH field stored the folder on the Windows operating system where the library members (e.g. data sets) were located. The PATH field had a maximum length of 1024. The MAPS library had a single metadata record; however, the SASHELP library had 4 records. SASHELP is termed a concatenated library, thus SASHELP has more than one path. The metadata contained 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 could be accessed to different versions of SAS files (Beatrous & Clifford, 1998).

Documented uses of VLIBNAM metadata were very limited. One obvious use of VLIBNAM was to obtain the path of a library in order to document output. The path of a library may also be obtained using SAS functions, for example the following syntax created a footnote with the MAPS library name and path.

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

The %LET statement was used to set a macro variable LIBNAME equal to the name of an existing library, MAPS. The value of LIBNAME was included as the argument to the function PATHNAME by placing an "&" before the macro variable. The "&" caused SAS to resolve the value of LIBNAME to MAPS, thus the function PATHNAME, with the help of %SYFUNC, set the macro variable LIBPATH equal to the operating system path of the MAPS library. Both the library name and path were shown in a footnote, again by placing an "&" as a prefix to both macro variables. The PATHNAME function was also used to return to the log all 4 paths for the SASHELP library:

```
%put %sysfunc(pathname(SASHELP));
```

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 was used to assess the engine metadata found in the ENGINES table of the DICTIONARY library.

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

Figure 7. ENGINES Metadata Returned by the VENGINE View

<table>
<thead>
<tr>
<th>engine</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS</td>
<td>SAS/ACCESS Interface to PC Files</td>
</tr>
<tr>
<td>ACCESS99</td>
<td>Read engine for CRSP ACCESS97 database</td>
</tr>
<tr>
<td>BASE</td>
<td>Base SAS I/O Engine</td>
</tr>
<tr>
<td>BLOOMBERG</td>
<td>SAS/Access Interface To Bloomberg</td>
</tr>
<tr>
<td>DMDP</td>
<td>DMDP Save file engine</td>
</tr>
<tr>
<td>CRSPACC</td>
<td>Read engine for CRSP ACCESS97 database</td>
</tr>
<tr>
<td>CVP</td>
<td>Character Variable Padding Engine</td>
</tr>
<tr>
<td>DR2</td>
<td>SAS/ACCESS Interface to DR2</td>
</tr>
<tr>
<td>EXCEL</td>
<td>SAS/ACCESS Interface to PC Files</td>
</tr>
<tr>
<td>FAMECHLI</td>
<td>Seamless libname interface to FAME db</td>
</tr>
</tbody>
</table>

There were a total of 46 metadata records returned by VENGINE; however, Figure 7 only showed the first 10. Simply obtaining a list of engines offered 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 LIBNAME statement offers powerful access to files that would otherwise be inaccessible or far less accessible. For example, SAS 9.1 has 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 Excel worksheets within a workbook as a data set. While the authors pointed out notable limitations in engine, access was enabled with the simple inclusion of EXCEL (the engine name) in the LIBNAME statement. For example, the following syntax was used to obtain basic metadata for all the sheets in an Excel workbook without having to actually import the Excel data.

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

The use of _ALL_ generated a PROC CONTENTS output for all the worksheets found in the Excel workbook. The output reported the sheet names. By default, the values of the first row of each sheet were assumed to be the variable names. SAS also determined the variable TYPE, LENGTH, FORMAT, and INFORMAT; however, options for the EXCEL engine in the LIBNAME statement may be used to guide SAS in its determination of some of these metadata (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.

### EXTERNAL FILE METADATA (EXPLORING EXTFILES TABLE USING VEXTFL VIEW)

The EXTFILES table of DICTIONARY stores external file references and paths known to the SAS session. The following syntax used a FILENAME statement to establish a filename MYFILE. The VEXTFL view returned records from the EXTFILES table to a PRINT procedure that listed the 3 columns of metadata shown in Figure 8.

```plaintext
filename myfile "c:\temp\test.txt";
proc print data=sashelp.vextfl;
run;
```

![Figure 8. Listing of Metadata Returned by VEXTFL](image)

The value of the FILeref field contained the name by which SAS referred to operating system paths, stored in the XPATH field. The FILeref values that begin with "#" were created by the SAS System by default, and FILeref values of MYFILE identified the metadata for the user defined name created with the FILENAME statement above. The following FILENAME statement with the CLEAR option removed MYFILE filename from the metadata.

```plaintext
filename myfile clear;
```

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):

```plaintext
%let programname = %sysget(sas_execfilename);
%let fullfile = %sysget(sas_execfilepath);
```

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

### 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 Table 1), the MEMBER table provides consolidated access to some important information for all members in a library.

The VMEMBER view returned 7 columns of metadata from the MEMBERS table of the DICTIONARY library. Each record returned by VMEMBER was uniquely identifiable by LIBNAME, MEMNAME, and MEMTYPE fields. As
shown in the LIBNAMES metadata, LIBNAME stored the name of the library. The MEMNAME field in VMEMBER contained the name of the member (for example, the SHOES data set in the SASHELP library), and MEMTYPE identified the member type (for example, as DATA). The following TABULATE syntax was used to generate Figure 9 in order to summarize the member metadata returned from VMEMBER.

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

Figure 9. Summary of Member Metadata Returned by VMEMBER View

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Data</th>
<th>Itemstor</th>
<th>MDDB</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP</td>
<td>290</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SASUSER</td>
<td>5</td>
<td>62</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>All</td>
<td>38</td>
<td>352</td>
<td>4</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 9 demonstrated that SAS members include not only data sets and views, but also catalogs, item stores, and multidimensional databases. VMEMBER also contained a library path (PATH) for each LIBNAME that was repeated across member records. The following PROC REPORT syntax was used to show the PATH field for the SASHELP.SHOES metadata (Figure 10 below).

```sas
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 10. Selected Member Metadata Returned by VMEMBER

Recall that the SASHELP library is a concatenated library, so in contrast to the LIBNAME table that stored each of the 4 paths as separate records, the MEMBERS metadata stored all the paths in one fields 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 (MEMNAME) to macro variables (Davis, 2001), or 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 created 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 within the libraries known to SAS, obtain the path of the data set, and assign the path to a library name of their choosing.

DATA SET AND RELATED METADATA

There were 5 data-related metadata tables shown in Table1: TABLES, VIEWS, COLUMNS, DICTIONARIES, and INDEXES. DICTIONARIES and VIEWS contain metadata about the metadata (e.g. information shown in Table 1), so this section will explore the metadata in TABLES, COLUMNS, and INDEXES using the SASHELP views VTABLE, VCOLUMN, and VINDEX.

DATA SET METADATA (EXPLORING TABLES TABLE USING VTABLE VIEW)

The VTABLE view was 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 listed all the 39 fields of
metadata found in the VTABLE, and the PRINT procedure was used to create the listing of the 7 selected fields shown in Figure 11:

```sas
proc contents data=sashelp.vtable; run;
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 11. Listing of VTABLE Metadata for SASHELP Data Sets RETAIL and SHOES**

<table>
<thead>
<tr>
<th>libname</th>
<th>memname</th>
<th>memtype</th>
<th>memlabel</th>
<th>nvar</th>
<th>nobs</th>
<th>modate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP</td>
<td>RETAIL</td>
<td>DATA</td>
<td>Retail Sales (Quartly: 1980Q1-1994Q2)</td>
<td>5</td>
<td>58</td>
<td>11MAR04:09:43:30</td>
</tr>
<tr>
<td>SASHELP</td>
<td>SHOES</td>
<td>DATA</td>
<td>Fictitious Shoe Company Data</td>
<td>7</td>
<td>395</td>
<td>11MAR04:09:43:35</td>
</tr>
</tbody>
</table>

Figure 11 showed two metadata records, the first recording information about the data set RETAIL and the second the data set SHOES. The columns LIBNAME and MEMNAME stored the name of the library and data set, respectively. The LIBNAME field had a maximum length of 8 characters, while the MEMNAME field had a maximum length of 32 characters. The value of "DATA" in the MEMTYPE field identified 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 stored the label assigned to the data set, NVAR and NOBS stored the number of variables and observations found in each data set, respectively, and MODATE stored the date and time of the last modification of the data set.

The WHERE statement in the PROC PRINT limited the output to 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 12 shows an accounting of all the metadata records returned from VTABLE using the following TABULATE procedure:

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

Figure 12. Summary of Metadata Returned from VTABLE

<table>
<thead>
<tr>
<th>Member Type</th>
<th>DATA</th>
<th>VIEW</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Library Name</td>
<td>MAPS</td>
<td>290</td>
<td>0</td>
</tr>
<tr>
<td>SASHELP</td>
<td>62</td>
<td>30</td>
<td>92</td>
</tr>
<tr>
<td>All</td>
<td>352</td>
<td>30</td>
<td>383</td>
</tr>
</tbody>
</table>

The 382 metadata records returned by VTABLE contained information from data sets and views. There were metadata for 352 data sets and 30 views. All 30 views were found in the SASHELP library, and the data sets were located in 2 different libraries, MAPS and SASHELP. In the SASHELP library, there were 62 data sets, of which 2, RETAIL and SHOES (Figure 11), were explicitly listed with the previous PRINT procedure.

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

For example, the following syntax was used to add the SHOES data set to the WORK library and the PROC TABULATE syntax produced the output in Figure 13:

```sas
proc sort data=sashelp.shoes out=shoes;
  by region;
proc print data=sashelp.vtable noobs;
  var libname memname memtype memlabel nvar nobs modate;
  where datepart(modate) = date();run;
```

Figure 13. Summary of Metadata Returned from VTABLE, Showing New WORK Metadata

<table>
<thead>
<tr>
<th>Member Type</th>
<th>DATA</th>
<th>VIEW</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Library Name</td>
<td>MAPS</td>
<td>290</td>
<td>0</td>
</tr>
<tr>
<td>SASHELP</td>
<td>62</td>
<td>30</td>
<td>92</td>
</tr>
<tr>
<td>WORK</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>353</td>
<td>30</td>
<td>383</td>
</tr>
</tbody>
</table>
Figure 14. Listing of VTABLE Metadata for WORK.SHOES

<table>
<thead>
<tr>
<th>libname</th>
<th>memname</th>
<th>memtype</th>
<th>name</th>
<th>type</th>
<th>label</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>DATA</td>
<td>Data</td>
<td>num</td>
<td>7</td>
<td>395</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23JUL07:08:00:49</td>
</tr>
</tbody>
</table>

Since WORK.SHOES was a copy of SASHELP.SHOES, the metadata shown in Figure 14 was nearly identical to the second row of Figure 11 with the notable exception of the LIBNAME and MODATE columns. In Figure 14, LIBNAME showed the value WORK and MODATE showed the date and time the data set was created by PROC SORT. The WHERE statement in the PROC PRINT was used to list only metadata records returned by VTABLE that were modified on the date the syntax was submitted, thus only a single metadata record was listed out of the 383 records available from VTABLE.

Just as adding a data set resulted in an additional VTABLE record, deleting a data set using PROC DELETE removed a record of metadata from VTABLE:

```sas
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 appeared to only be accurate after the library where the data set existed was refreshed through the submission of a LIBNAME statement.

Some of the documented uses of the TABLES table/VTABLE view have been to:

- Print 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)
- Calculate size of each library using NPAGE (Dilorio & Abolafia, 2004)
- Determine the number observations in any data set and/or use it to determine the existence of a data set (Dilorio & Abolafia, 2004)
- Determine whether a data set is empty (Dilorio & Abolafia, 2004; Varney, 2004)
- To comply with FDA standards, check that data set labels are less than 40 characters (Teng & Wang, 2006)

Figure 15 showed 7 metadata records one for each variable in the SHOES data set. The listing of 7 variables was consistent with the value of the NVAR field for the SHOES record returned by VTABLE (Figure 11). The first 3 fields of Figure 15 were also found in the VTABLE listing, LIBNAME, MEMNAME, and MEMTYPE. These fields identified 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 listed in Figure 15 included: (a) the NAME field that stored the name of each variable found in SHOES, (b) the TYPE field that identified each variable as character or numeric, (c) the LABEL field that stored the variable label, and (d) the FORMAT field that contained the name of the format assigned to each variable.

Table 15. Listing of VCOLUMN Metadata for SASHELP.SHOES

<table>
<thead>
<tr>
<th>libname</th>
<th>memname</th>
<th>memtype</th>
<th>name</th>
<th>type</th>
<th>label</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP</td>
<td>SHOES</td>
<td>DATA</td>
<td>Region</td>
<td>char</td>
<td>Number of Stores</td>
<td>DOLLAR</td>
</tr>
</tbody>
</table>
A PROC TABULATE was used to summarize the metadata returned by VCOLUMN (Figure 16):

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

Figure 16 showed that there were 2,498 records returned by VCOLUMN. Each record had information on a single variable. There were metadata records for 211 variables returned from SAS views, and 2,287 variables found in data sets. Consistent with VTABLE metadata, all the views were found in SASHELP (Figure 12). Figure 16 showed that all the 211 variables in views originated from views in the SASHELP library.

Of the 2,287 records having information on variables in data sets, 527 were found in SASHELP data sets. And as listed in Figure 15, 7 of the 527 metadata records were for variables found in SASHELP.SHOES.

A PROC TABULATE was again used with VCOLUMN to summarize the variable level metadata for the SHOES and RETAIL data sets (Figure 17):

```sas
proc tabulate data=sashelp.vcolumn;
class libname memname memtype type;
table libname*memname*memtype, type all/rtspace=40;
where libname = "SASHELP" and memtype = "DATA" and memname in
   ("SHOES", "RETAIL");
run;
```

Figure 17 showed that counting records within LIBNAME, MEMNAME, and MEMTYPE created an accurate count of variables found in the data sets RETAIL and SHOES. Seven variables were found in SHOES: 3 were character and 4 were numeric, while all 5 variables in RETAIL were numeric.

Various uses of the COLUMNS metadata included:
- Determine all data sets in a library that have a variable (s) in common (Lafler, 2005)
- Identify inconsistencies in length and type for variables of the same name across data sets in a library (Diliorio & Abolafia, 2004)
- Save a specific set of variables identified by a naming convention to macro variables and dynamically build syntax including those variable names (Diliorio & Abolafia, 2004)
- Identify variable names, name lengths, and label lengths that would be incompatible with SAS version 8 transport format (Diliorio & Abolafia, 2004)
- Determine the existence of a variable(s) (Varney, 2004)
- Rename specific variables (Mack, 2006)
- Create an empty copy of a data set (Mack, 2006)
- Count the number of data sets in a library having each variable. As an extension list the data sets, variable name, type, label and format for all variables appearing in more than one data set (Eberhardt & Brill, 2006)
- Apply 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 is searched. Indexes often improve performance. Raithel (2004) provided an extensive discussion of when and how to use SAS indexes.
The VINDEX view was used with PROC COTENTS and PRINT to explore the fields and metadata returned from the INDEXES table of DICTIONARY. There were 42 VINDEX records associated with data sets in the SASHELP library and the following PROC PRINT was used to list 10 records (Figure 18).

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

Figure 18. First 10 OBS from INDEXES returned via the VINDEX view

<table>
<thead>
<tr>
<th>Obs</th>
<th>Libname</th>
<th>Memname</th>
<th>Memtype</th>
<th>Name</th>
<th>Idxusage</th>
<th>Indxname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SASHELP</td>
<td>ADOMSG</td>
<td>DATA</td>
<td>MNEMONIC</td>
<td>SIMPLE</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>2</td>
<td>SASHELP</td>
<td>ADOMSG</td>
<td>DATA</td>
<td>MSG1D</td>
<td>SIMPLE</td>
<td>MSG1D</td>
</tr>
<tr>
<td>3</td>
<td>SASHELP</td>
<td>ADOMSG</td>
<td>DATA</td>
<td>MNEMONIC</td>
<td>SIMPLE</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>4</td>
<td>SASHELP</td>
<td>ADOMSG</td>
<td>DATA</td>
<td>MSG1D</td>
<td>SIMPLE</td>
<td>MSG1D</td>
</tr>
<tr>
<td>5</td>
<td>SASHELP</td>
<td>AFMSG</td>
<td>DATA</td>
<td>MNEMONIC</td>
<td>SIMPLE</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>6</td>
<td>SASHELP</td>
<td>CLMSG</td>
<td>DATA</td>
<td>MNEMONIC</td>
<td>SIMPLE</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>7</td>
<td>SASHELP</td>
<td>CLNMSG</td>
<td>DATA</td>
<td>MSG1D</td>
<td>SIMPLE</td>
<td>MSG1D</td>
</tr>
<tr>
<td>8</td>
<td>SASHELP</td>
<td>DESKACT</td>
<td>DATA</td>
<td>OBJECTYPE</td>
<td>COMPOSITE</td>
<td>objsub</td>
</tr>
<tr>
<td>9</td>
<td>SASHELP</td>
<td>DESKACT</td>
<td>DATA</td>
<td>OBJECTYPE</td>
<td>COMPOSITE</td>
<td>objsub</td>
</tr>
<tr>
<td>10</td>
<td>SASHELP</td>
<td>DESKACT</td>
<td>DATA</td>
<td>OBJECTYPE</td>
<td>COMPOSITE</td>
<td>objsub</td>
</tr>
</tbody>
</table>

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

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

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 VCATALG and VFORMAT views, respectively. STYLES metadata will not be discussed.

CATALOGS AND CATALOG-SPECIFIC METADATA (EXPLORING CATALOGS TABLE USING VCATALG VIEW)
The VCATALG view returned 10 fields of metadata from the CATALOGS table of DICTIONARY. As a SAS member a catalog is similar to a data set in the sense 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. The following syntax created formats that SAS automatically stored in a catalog called FORMATS in the WORK library. The VCATALG metadata for the WORK library was listed using PROC PRINT (Figure 19).

```
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 19. Gender and Grade Entries for a Format Catalog

<table>
<thead>
<tr>
<th>Libname</th>
<th>Memname</th>
<th>Memtype</th>
<th>Objname</th>
<th>Objtype</th>
<th>Objdesc</th>
<th>Created</th>
<th>Modified</th>
<th>Alias</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>FORMATS</td>
<td>CATALOG</td>
<td>GENDER</td>
<td>FORMAT</td>
<td>23JUL07:11:34:00 23JUL07:11:34:00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>FORMATS</td>
<td>CATALOG</td>
<td>GRADE</td>
<td>FORMAT</td>
<td>23JUL07:11:34:00 23JUL07:11:34:00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two records of metadata were listed for the WORK.FORMATS catalog. The MEMTYPE field verified that the FORMATS member was in fact a catalog. The two entries in the catalog were shown in the OBJNAME field as GENDER and GRADE. The OBJTYPE field defined GENDER and GRADE as format entries. There were 48 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 this paper to discuss the various other entries.

The documented uses of the CATALOGS metadata seemed limited. Carpenter (1998) used the CATALOGS
metadata to obtain a list of members in library and copy them to another library. PROC CATALOG also allows each user defined format in a catalog to be assigned a maximum of a 256 character description that may then be obtained from the OBJDESC field in CATALOGS. For example, the following syntax used PROC CATALOG to assign a description to the GENDER format created above. C=WORK.FORMAT defined the catalog and gender was referenced in the MODIFY statement as GENDER.FORMAT. The familiar PROC PRINT was used to list the specific metadata in Figure 20.

```
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 20. Gender and Grade Entries for a Format Catalog with Gender Description**

The format descriptions may be assigned and then harvested from VCATALG view to use in documentation, and the descriptions are not available in the FORMAT DICTIONARY table or through the data set created by PROC FORMAT CNLTOUT=.

**AVAILABLE FORMATS METADATA (EXPLORING FORMATS TABLE USING VFORMAT VIEW)**

The VFORMAT view returned 13 fields of metadata from the FORMAT table of DICTIONARY. A PROC TABULATE was used to summarize the metadata records (Figure 21).

```
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 21. Summary of Metadata Returned from VFORMAT**

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Member Name</th>
<th>Format Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>FORMATS</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

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

Figure 20. Gender and Grade Entries for a Format Catalog with Gender Description

The familiar PROC PRINT was used to list the specific metadata in Figure 20.

```
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;
```

Unlike the previous tables with LIBNAME and MEMNAME fields, VFORMAT may return missing values for these fields. For example, Figure 21 shows that LIBNAME and MEMNAME only have values for the user defined format $SOURCE that was saved in the WORK library and FORMATS catalog. The majority of the metadata were SAS defined FORMATS (FMTTYPE=F) and INFORMATS (FMTTYPE=I). A PROC PRINT was 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;
```
The first metadata record in Figure 22 shows the user defined format $SOURCE that is indicated to be a format (FMTTYPE=F) created by a user (SOURCE=C). According to SAS Support (see references FAQ # 3987) a SOURCE=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 20). The documented uses for the FORMATS metadata seemed limited. SAS Support (see references FAQ # 3987) provided a macro program that used FORMATS to determine the existence on a format or informat name in the metadata. Crawford (2006) used FORMAT metadata to produce a list of perspective informat that could be used to read a variable. He used VFORMAT in a data STEP to apply all existing informat to a variable value and listed in the LOG those informat that were applied without error.

CONSTRANTS 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 created the SHOES data set where observations where not allowed to have RETURNS greater than $20,000.

```sas
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.

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

A constraint cannot be defined if data violates the constraint, so the first step in the strategy was to create an empty copy of the SASHELP.SHOES data set. An empty data set was created from SASHELP.SHOES using the OBS=0 OPTION (Gerlach, 2005). MODIFY SHOES started a block of statements used to modify shoes, in this case, to add a constraint. The integrity constraint statement began with IC and created a CHECK constraint named C_RETURNS1. The constraint rule was defined through CHECK (WHERE=). The MESSAGE option was used along with option MSGTYPE=USER to define a custom message with a maximum of 250 characters. The message was stated to identify observations that do not comply with the constraint. A second IC statement was included to create the C_SALES1 constraint. This constraint was included to illustrate that cross-variable integrity may also be enforced using CHECK constraints. In this case, only observations with SALES greater than RETURNS were allowed. APPEND put the observations from SASHELP.SHOES that passed the constraint criteria into WORK.SHOES. The LOG showed that only 391 of the 395 observations in SASHELP.SHOES were added to
SHOES and a warning message was issued: “WARNING: Returns over $20,000 (Occurred 4 times.)” Later it will be shown that additional syntax may be used to create metadata to account for the 4 observations that were noted appended into SHOES; however, the previous syntax still prompted SAS to create several sources of valuable metadata.

The most convenient access to the constraint metadata was through PROC CONTENTS (Figure 23).

```sas
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 23. Constraint Metadata Obtained from PROC CONTENTS OUT2=  

<table>
<thead>
<tr>
<th>Libname</th>
<th>Member</th>
<th>Name</th>
<th>Type</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_returns</td>
<td>Check</td>
<td>Returns over $20,000 (Returns&gt;=. and Returns&lt;=20000)</td>
</tr>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_sales</td>
<td>Check</td>
<td>Sales &lt;= Returns Sales &lt;= Returns</td>
</tr>
</tbody>
</table>

The OUT2 option of PROC CONTENTS (Jensen & Franklin, 2000) was used to generate metadata data for the newly created constraints. The familiar LIBNAME field was present. Unfortunately, the MEMNAME field was named MEMBER; however, the LIBNAME and MEMBER fields did identify the origin of each constraint. This list of constraints was easy to obtain, however, it did not contain a field storing the variable names involved in each constraint. SASHELP views may be used to obtain a list of variables involved in each constraint. Table 2 shows the key fields for the 4 metadata tables that SAS used to track CHECK constraint information.

<table>
<thead>
<tr>
<th>DICTIONARY</th>
<th>Constraint Field</th>
<th>Library Field</th>
<th>Data Set Field</th>
<th>Variable Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASHELP.VCHKCON via CHECK_CONSTRAINTS</td>
<td>CONSTRAINT_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SASHELP.VTABCOM via TABLE_CONSTRAINTS</td>
<td>CONSTRAINT_NAME</td>
<td>TABLE_CATALOG</td>
<td>TABLE_NAME</td>
<td></td>
</tr>
<tr>
<td>SASHELP.VCNTABU via CONSTRAINT_TABLE_USAGE</td>
<td>CONSTRAINT_NAME</td>
<td>TABLE_CATALOG</td>
<td>TABLE_NAME</td>
<td></td>
</tr>
<tr>
<td>SASHELP. VCNCOLU via CONSTRAINT_COLUMN_USAGE</td>
<td>CONSTRAINT_NAME</td>
<td>TABLE_CATALOG</td>
<td>TABLE_NAME</td>
<td>COLUMN_NAME</td>
</tr>
</tbody>
</table>

The first view, VCHKCON returned the list of CHECK constraints along with the syntax shown in the WHERE variable of Figure 23. The constraint names were found in the CONSTRAINT_NAME field. There were no fields identifying the library or data set for the constraint. Few fields were returned by the second and third views and very little information was found regarding their purpose. However, the fourth view, VCNCOLU offered an explicit link between the constraints and the variable(s) to which the constraints were applied. Despite the unfortunate naming, the TABLE_CATALOG field identified the library, the TABLE_NAME field the name of the data set, and the COLUMN_NAME the name of the variables associated with the constraint. The following syntax used the VCNCOLU view to return observations from the CONSTRAINT_COLUMN_USAGE table of DICTIONARY (Figure 24).

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

Figure 24. Listing Check Constraints returned by SASHELP.VCNCOLU  

<table>
<thead>
<tr>
<th>table_ catalog</th>
<th>table_ name</th>
<th>column_ name</th>
<th>constraint_ name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_returns</td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_sales</td>
<td></td>
</tr>
</tbody>
</table>

The following syntax was used to merge the metadata sources from PROC CONTENTS OUT= and OUT2= and SASHELP.VCNCOLU to create a list of variables and labels along with the constraints (Figure 25).

```sas
data vcncolu (keep=libname memname name constraint_name);
set sashelp.vcncolu;
rename table_catalog = libname;
rename table_name = memname;
rename column_name = name;
```

The first view, VCHKCON returned the list of CHECK constraints along with the syntax shown in the WHERE variable of Figure 23. The constraint names were found in the CONSTRAINT_NAME field. There were no fields identifying the library or data set for the constraint. Few fields were returned by the second and third views and very little information was found regarding their purpose. However, the fourth view, VCNCOLU offered an explicit link between the constraints and the variable(s) to which the constraints were applied. Despite the unfortunate naming, the TABLE_CATALOG field identified the library, the TABLE_NAME field the name of the data set, and the COLUMN_NAME the name of the variables associated with the constraint. The following syntax used the VCNCOLU view to return observations from the CONSTRAINT_COLUMN_USAGE table of DICTIONARY (Figure 24).

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

Figure 24. Listing Check Constraints returned by SASHELP.VCNCOLU  

<table>
<thead>
<tr>
<th>table_ catalog</th>
<th>table_ name</th>
<th>column_ name</th>
<th>constraint_ name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_returns</td>
<td></td>
</tr>
<tr>
<td>WORK</td>
<td>SHOES</td>
<td>c_sales</td>
<td></td>
</tr>
</tbody>
</table>

The following syntax was used to merge the metadata sources from PROC CONTENTS OUT= and OUT2= and SASHELP.VCNCOLU to create a list of variables and labels along with the constraints (Figure 25).

```sas
data vcncolu (keep=libname memname name constraint_name);
set sashelp.vcncolu;
rename table_catalog = libname;
rename table_name = memname;
rename column_name = name;
```

The first view, VCHKCON returned the list of CHECK constraints along with the syntax shown in the WHERE variable of Figure 23. The constraint names were found in the CONSTRAINT_NAME field. There were no fields identifying the library or data set for the constraint. Few fields were returned by the second and third views and very little information was found regarding their purpose. However, the fourth view, VCNCOLU offered an explicit link between the constraints and the variable(s) to which the constraints were applied. Despite the unfortunate naming, the TABLE_CATALOG field identified the library, the TABLE_NAME field the name of the data set, and the COLUMN_NAME the name of the variables associated with the constraint. The following syntax used the VCNCOLU view to return observations from the CONSTRAINT_COLUMN_USAGE table of DICTIONARY (Figure 24).

```sas
proc print data=sashelp.vcncolu noobs;
var table_catalog table_name column_name constraint_name;
run;
```
run;
proc contents data=shoes out=m_contents_columns (keep=libname memname name label type format) out2=m_contents_constraints (keep=libname member name type message where rename=(member=memname name=constraint_name type=constraint_type)) noprint;
run;
proc sort data=vcncolu; by libname memname constraint_name; run;
proc sort data=m_contents_constraints; by libname memname constraint_name; run;
data m_constraints;
    merge vcncolu(in=v) m_contents_constraints;
    by libname memname constraint_name;
    if v;
run;
proc sort data=m_contents_columns; by libname memname name; run;
proc sort data=m_constraints; by libname memname name; run;
data m_dictionary;
    merge m_contents_columns (in=vv) m_constraints;
    by libname memname name;
    if vv;
    if label = "" then label = name;
run;
title1 "table of variables and constraints";
proc report data= m_dictionary nowd nocenter missing;
columns libname memname name label constraint_name where;
define libname/ group noprint;
define memname/ group noprint;
define name / group width=12;
define label/ group width=20 flow;
define constraint_name/ group width=10 flow;
define where/ group width=40 flow;
compute before _page_;
    line @1 "library: " libname $8. " data set:" memname $32.;
endcomp;
break after name /skip;
run;

Figure 25. Listing Variables and CHECK Constraints

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Set:SHOES</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory</td>
<td>Total Inventory</td>
<td>Total Returns</td>
</tr>
<tr>
<td>Product</td>
<td>Product</td>
<td>c_returns1 (Returns&gt;= and Returns&lt;=$20,000)</td>
</tr>
<tr>
<td>Region</td>
<td>Region</td>
<td>c_sales1 (Sales&gt;$20,000)</td>
</tr>
<tr>
<td>Sales</td>
<td>Total Sales</td>
<td>c_sales1 Sales&gt;Returs</td>
</tr>
<tr>
<td>Stores</td>
<td>Number Of Stores</td>
<td></td>
</tr>
<tr>
<td>Subsidiary</td>
<td>Subsidiary</td>
<td></td>
</tr>
</tbody>
</table>

PROC DATASETS was useful in generating constraints for documentation; however, the CHECK constraints effectively discarded observations that failed the data integrity criteria. It is often important to examine the observations that failed integrity criteria in order to correct the problem, or at least report those observations. For example, the following used data STEP syntax created the variable FLG_RETURNS that indicated when the value of RETURNS was greater than $20,000

proc format;
    value flg_returns 0 = "missing or $0-20,000"
        1 = "returns over $20,000";
run;
data shoes;
set sashelp.shoes;
PROC FREQ created a count of the number of observation that had RETURNS over $20,000 and PROC PRINT listed those 4 observations. The following is an extension of PROC DATASETS that accomplished the same task while providing even more valuable metadata to account for observations that failed the constraints.

```sas
proc datasets library=work nolist;
modify shoes;
ic create c_returns1=check(where=(returns >=. and returns <= 20000)) message = "returns over $20,000" msgtype=user;
ic create c_sales1=check(where=(sales > returns)) message = "sales <= returns" msgtype=user;
run;
delete shoes_audit;
append base=shoes_audit data=shoes(type=audit here=(_atopcode_="EA"));
audit shoes; terminate;
append base=shoes_audit data=shoes force;
quit;
```

Syntax was added to the previous PROC DATASET syntax to enable SAS AUDIT capability. When AUDIT is enabled SAS will monitor aspects of the audited data set and save various metadata tables.

1. The AUDIT statement identified the empty data set SHOES as the data set to audit.
2. The INITIATE statement started the audit.
3. The LOG statement instructed on the types of metadata filed to create and maintain during the audit process. As a result of the option ERROR_IMAGE=YES, SAS was instructed to save a record of any observations that could not be added to SHOES due to failures on constraints. 4. As in the previous syntax APPEND was used to attempt to add all the original SASHELP.SHOES into the empty SHOES data set. With AUDIT initiated, the 4 observations that could not be appended to SHOES due to constraint failure were added to a SHOES metadata file of type AUDIT. This data set does not exist after the audit capability is terminated, so the next statements were used to save the audit metadata to a permanent data set called SHOES_AUDIT.
5. The DELETE statement was used to make sure there was no data set in WORK called SHOES_AUDIT. 6. The APPEND statement created the BASE data set SHOES_AUDIT from the SAS AUDIT file. The AUDIT file was accessed through the data set option TYPE=AUDIT. The WHERE statement was used to make sure that only metadata data records in the audit file as a result of the error "EA", or "observations added failed" were appended to the SHOES_AUDIT data set. 7. The SHOES audit process was terminated. 8. The SHOES data set containing the 391 observations that passed the constraints was appended to the SHOES_AUDIT data set containing the 4 observations that failed the constraints, thus SHOES_AUDIT contained all the original SASHELP.SHOES data. The option FORCE was used because the SHOES_AUDIT data set contained additional variables, or metadata fields, that SAS maintained as a result of the audit process. A PROC CONTENTS of SHOES_AUDIT revealed the following additional metadata fields:

- _ATDATETIME_ is a numeric field containing the data and time, with a format of DATETIME19.
- _ATMESSAGE_ is a character field containing the error message
- _ATOBSNO_ is a numeric field containing the observation number of the record with the error
- _ATOPCODE_ is a character variable that records the type of error (e.g. EA)
- _ATRETURNCODE_ is a number for which the use is unclear
- _ATUSERID_ is the operating system user known to SAS

The following syntax used the SHOES_AUDIT metadata table to summarize and list observations failing the constraints (Figure 26).

```sas
proc format;
  value $atmess (default=60) ' ' = 'no error found';
run;
```
proc freq data=shoes_audit;
    table _atmessage_ /missing nocum;
    format _atmessage_ $atmess.);
run;
proc print data=shoes_audit;
where _atopcode_ = "ea";
run;

Figure 26. PROC FREQ and PRINT of Observations Failing the Constraints

<table>
<thead>
<tr>
<th><em>ATMESSAGE</em></th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>No error found</td>
<td>391</td>
<td>98.99</td>
</tr>
<tr>
<td>ERROR: Returns over $20,000</td>
<td>4</td>
<td>1.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Region</th>
<th>Product</th>
<th>Subsidiary</th>
<th>Stores</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Canada</td>
<td>Men's Casual</td>
<td>Vancouver</td>
<td>25</td>
<td>$353,351</td>
</tr>
<tr>
<td>2</td>
<td>Canada</td>
<td>Slipper</td>
<td>Vancouver</td>
<td>27</td>
<td>$700,513</td>
</tr>
<tr>
<td>3</td>
<td>Central America/Caribbean</td>
<td>Men's Casual</td>
<td>Kingston</td>
<td>28</td>
<td>$576,112</td>
</tr>
<tr>
<td>4</td>
<td>Middle East</td>
<td>Men's Casual</td>
<td>Tel Aviv</td>
<td>11</td>
<td>$1,258,717</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>ATUSERID</em></th>
<th><em>ATOPCODE</em></th>
<th>Returns</th>
<th><em>ATDATE</em></th>
<th><em>ATERROR</em></th>
<th><em>ATRETURN</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pthornton</td>
<td>EA</td>
<td>Returns over $20,000</td>
<td>2007-06-13</td>
<td>07:04:10</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>pthornton</td>
<td>EA</td>
<td>Returns over $20,000</td>
<td>2007-06-13</td>
<td>07:04:10</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>pthornton</td>
<td>EA</td>
<td>Returns over $20,000</td>
<td>2007-06-13</td>
<td>07:04:10</td>
<td>106</td>
</tr>
<tr>
<td>4</td>
<td>pthornton</td>
<td>EA</td>
<td>Returns over $20,000</td>
<td>2007-06-13</td>
<td>07:04:10</td>
<td>184</td>
</tr>
</tbody>
</table>

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


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


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


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


ACKNOWLEDGMENTS
Thanks to Cyndi Williamson and Mary McCracken for their time in reviewing the paper.

RECOMMENDED READING


International Conference, Long Beach, FL


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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.