Paper 69-27

What's In A Name; Describing SAS® File Types

Randall Cates, Technical Training Specialist 3
SAS Institute Inc., St. Louis Training Center

ABSTRACT

Are you confused by the different types of SAS files in your libraries? Need a scorecard? If a file has the type .SAS7BDAT that's a SAS dataset, right? So how is a type .SAS7BVEW different? Or is it different? If you can send a SAS dataset (.SAS7bdat) to another SAS programmer, why can't you do the same with a .SAS7BVEW? What's a MDDB? An Item Store file?

There are a number of different types of SAS files available to the SAS programmer. The SAS dataset is the most familiar, but many programmers also use SAS Views, SAS Catalogs, But each one stores something different. You can possibly find many if not all file types in your company's SAS libraries. This paper will review for the beginning SAS programmer all the different file types, how to use them to best effect, and will present some ideas for better file management.

First we will discuss each different file type, starting with the SAS dataset, and talk about what it stores, and how you can access and use it. For example, the SAS catalog has the file type of .SAS7BCAT, and it can store a number of items including sas code(both compiled and text) macros, and formats. Depending on what it stores you might want to keep each catalog in a specific library, or all catalogs in one.

INTRODUCTION

The SAS® system contains a veritable rainbow of different file types. However, the average SAS programmer only uses four or five primary file types; the SAS program, SAS dataset, SAS log, SAS Listing, with an occasional stab at a SAS catalog or a SAS index file. This paper will present an overview of most of the different files that are available, where they exist, and how they are used in SAS programming.

SAS file types discussed are:

- Program files
- Log files
- Listing files
- Datasets
- Data views
- Catalogs
- Indexes
- Audit files
- Stored programs
- Dictionary tables
- Item stores
- Multidimensional databases.

SAS Libraries and SAS I/O engines will also be reviewed.

In addition, certain other SAS products use different file structures. For example, IT Service Vision® uses Performance Databases (PDBs), CFO Vision® uses Financial Databases (FDBs), and Enterprise Guide® uses Project files. This paper will not cover these specialty file structures.

This presentation is intended as an Introduction to SAS file types and will not attempt to cover all aspects of any particular file type. Focus will be on files as they exist on MS-Windows® operating systems.

SAS FILE TYPES

SAS file types can be separated into two types of files, whether they are visible in a SAS library or not.

Files that are visible in a library are: datasets, data views, catalogs, stored programs, dictionary tables, and multidimensional databases.

Files that are not visible in a library are: program files, log files, listing files, engines, indexes, audit files, and item stores.

We will begin our tour by looking at SAS libraries.

SAS LIBRARIES

A SAS data library is defined as "a collection of one or more SAS files that are recognized by the SAS system and that are referenced and stored as a unit." Therefore, a SAS library is not a file. Instead it's a way of looking at a bunch of SAS files as a group, instead of a lot of individual files. Usually a library corresponds to all of the appropriate SAS files in a particular subdirectory.

The SAS system uses the LIBNAME statement to set up a SAS library and define the pathname(s) of the SAS files. For example:

```
LIBNAME mydata 'C:\My Data\SUGI27';
```

Now, the programmer or program has access to any SAS files at that location just by using the library reference, in this case "mydata".

```
PROC CONTENTS DATA=mydata._all_ nods;
Run;
```

The above code prints out a report of the names and file types of all SAS files in the Mydata library.

When the SAS system is started, either through Batch processing or Interactive Windowing modes, it automatically sets up a set of libnames; the WORK, SASHELP, and SASUSER libnames. These libraries each have special functions.

The SASHELP library consists of a combination of numerous subdirectories and contains links to catalogs and other SAS files. Some of these files contain information used to control various aspects of the SAS session. This library is also where you can find the files that the Help menu accesses. The defaults stored in this library are for everyone using SAS at your installation.

The SASUSER library contains SAS files that enable each programmer to tailor their session to their own separate requirements. For example, if you wish to set up and use a different set of Fkey commands than the default set, your list of commands are stored in a catalog file here. You may also want to store some files that you use all the time here. The SASUSER library can also be used to store user-defined formats that you access frequently but that are not part of your company's collection of valid formats. Formats will be discussed further in the Catalogs section.

The WORK library is a temporary scratch repository. It stores any temporary files that you create and any temporary files that SAS creates during the processing of the session. The SAS system creates a new directory for this each time a SAS session is initiated, and is deleted at the end of the SAS session. Whenever you create a SAS data set using just one name, it defaults to the Work library. For example:

```
DATA test;
```

Is interpreted by the SAS system as:

```
DATA WORK.test;
```

If you would like to specify a Permanent library to be the default repository you can set up a USER library using a standard LIBNAME statement. For example:

```
DATA USER 'D:\My Temp Files';
```

After this statement, whenever a one-level file is mentioned in the code, the SAS system will interpret it to mean the USER library. The WORK library is still available, but you now need to explicitly specify it.

SAS I/O ENGINES

Engines are not strictly files, but are sets of internal instructions that SAS uses to read from and write to files. Engines form a access path between the SAS system and the data.

For example, you can use the Oracle LIBNAME engine to transparently access data stored in an Oracle database. The LIBNAME statement would look like this:

```
LIBNAME myoracle ORACLE user=me pw=mypasswd
     Path=dbmssrv schema=special;
```

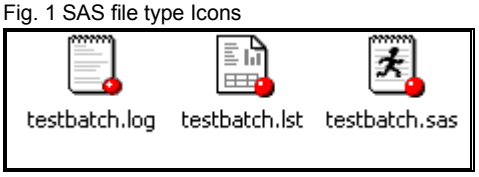
The name of the library is Myoracle. Everything else in the LIBNAME statement are instructions to the SAS system to control access to the Oracle database. The word ORACL tells SAS to use the Oracle engine. The engine has all the instructions inside it enabling it to locate the files and bring the data into the SAS workspace in a clear and usable form. The files that enable this particular engine belong to the SAS/ACCESS to ORACLE product.

When accessing directories containing SAS data files, usually you do not have to specify what types of SAS files are contained there. The SAS system will choose the appropriate engine for you. When you submit the LIBNAME statement accessing (for example) a directory containing SAS Version 8 files, the SAS system identifies the files there as V8 files and accesses the V8 engine. The Log will contain a note that specifies engine the SAS system has assigned to the library.

SAS PROGRAMS, LOGS, AND LISTINGS

Program, log, and listing files are probably the most frequently used files in the SAS system. They are all structured as text files, in that programmers can use any Text Editor to open, peruse, and edit them. They are, however separated from other text files by their extensions. Program files usually have a .SAS extension, Log files have a .LOG extension, and Listing files have a .LST extension. Figure 1 shows how these three files look in MS-Windows Explorer. These types of files will not be seen through a library, even if the library accesses a directory containing these files.

Fig. 1 SAS file type Icons



Programmers create program files. As the name implies, the program file contains SAS code that directs SAS to fulfill some data processing task(s).

Log files are created as a result of a program file being submitted to the SAS system for execution, most usually by the Batch programming method. However, when a programmer working in interactive SAS saves the contents of the Log Window, the default type of the file is also .LOG. The log file contains the program code that was submitted as well as notes, error messages and warning messages that describe how the SAS system processed the program code.

Listing files are also created as a result of a program file being submitted to the SAS system. However, the listing file is only created as a result of program code that generates some type of report. If the code doesn't request (for example) a PROC PRINT, or worse yet, has errors that prevent a report from being created, the listing file is not created.

SAS DATA SETS

The SAS data file (or dataset) stores data. Probably for most SAS programmers the SAS dataset is the storage option of choice. The SAS dataset has a .SAS7BDAT extension (See Fig.2).

Fig. 2 SAS Data set icon

The SAS dataset stores character and numeric data in the form of a 2-dimensional table of discrete observations and variables. The observations correspond to the rows of a table and consist of all the stored information about one "entity" of choice. The variables correspond to the columns of a table and each consists of all discrete data points for one item of interest.

For example, one Human Resources dataset would contain information about employees. Each record would then consist of all of the information about one employee. Example variables would then consist of; Name, Age, JobTitle, etc. Since the data exist in a tabular form all records will contain every variable, but for some records there can be variables that contain nothing, if that information is missing.

Datasets can be created in a number of ways. Programmers use the DATA step to create and modify dataset. For example: the following DATA step creates a new dataset called One in the Work library from an existing data file called Two, also in the Work library. Since nothing else has been specified, all observations and all variables from Work.Two have been copied into Work.One.

```
DATA work.one;
     SET work.two;
RUN;
```

Datasets can also be created by reading in raw text files, and files from other types of applications, for example from Excel files.

There are 2 parts to the dataset structure, the Descriptor portion and the Data portion.

The Descriptor portion contains Metadata, that is; all of the information that describes the structure of the dataset, also called the Attributes of the dataset. Contained in the Descriptor portion is: the name of the dataset, the number of observations, number of variables, variable names, variable types, variable lengths, and so on. To view this information you can go to the SAS Explorer Window, find the appropriate library, for example Work, double-click on the library, which will open it and show you all SAS files currently there. Then Right-Mouse click on the appropriate dataset and choose Properties.

Or you can use PROC CONTENTS to obtain a report of the Descriptor portion.

```
PROC CONTENTS Data=work.one;
RUN;
```

The Data portion contains the data. To view the data, similarly you can Double-click on the dataset in SAS Explorer, or Right Mouse click on the dataset and choose Open. This will show you a Viewtable of the data.

Or you can use PROC PRINT to print the data to the Output.

```
PROC PRINT Data=work.one;
RUN;
```

Some advanced concepts about SAS datasets concern compression of data files, password protection, indexing (See SAS Index files section), and SAS Audit trails (See SAS Audit Trails section).

If data storage space is a concern at your site, you might consider compressing your SAS datasets to save space. Compressing datasets essentially takes out trailing blanks from Character variables. To do this; specify (Compress=Yes) in the DATA step.

```
DATA Work.One(compress=yes);
     SET work.two;
RUN;
```

This option only works well with datasets that have character variables containing lots of trailing blanks. If the dataset contains primarily numeric variables, and/or the character variables do not have many trailing blanks, the compressed dataset could end up larger than the uncompressed dataset.

Compressed datasets can be used exactly the same as uncompressed datasets, however, there will be a small processing increase since records are uncompressed whenever they are accessed.

SAS datasets are somewhat protected from inappropriate use since only the SAS system can read them. However, if data security is a concern at your job, you can add password protection to your datasets. Additionally you can set up three different levels of protection; read-protection, write-protection, or alter protection. Read protection allows you to read but not modify the dataset. Write protection allows you to read the data and to edit or update the dataset, but not to modify the structure of the dataset Alter protection allows full access to the dataset including adding, deleting or renaming variables, and deleting the dataset in code.

To protect a dataset using DATA step code you use the specify the level of password protection you need using the READ=, ALTER=, or WRITE= dataset option on the new dataset. Not all options are necessary.

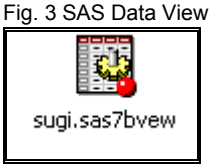
```
DATA work.one(READ=green, WRITE=red,
ALTER=blue);
     SET work.two;
RUN;
```

If you attempt to access a protected data file without supplying the appropriate password, the SAS system will prompt you for a password in a prompt window. If you do not supply the correct password you will be unable to access the data.

SAS DATA VIEWS

The SAS Data View looks and acts just like a SAS Dataset. However the Data View contains no data whatsoever. Instead the Data view contains code. The SAS Data View file has a .SAS7BVEW extension (See Figure 3).

Fig. 3 SAS Data View icon



You can use a data view as a data source in a DATA step (SET, MERGE, UPDATE), a PROC PRINT, or other procedures. And the results generally are the same as if it were a regular SAS dataset, containing data.

The data view actually contains stored, compiled SAS code that, when accessed, can read data immediately from a variety of sources, including, raw text files, SAS datasets, SQL tables, and tables from other applications.

Advantages to SAS Data Views include being able to have "virtual" tables that mimic extremely large datasets without being actual copies of them, thus saving storage space. The data view takes up very few bytes of storage space. Also, if you have information that changes regularly, you can set up a data view that will always give you up-to-date information each time you access it. If security is an issue, you can set up a data view to the data so that users will only access "just" the information that they need, without letting them see the rest of the data.

There are a number of limitations to data views. You cannot copy a view to a floppy disk or email and send it to a colleague. The data view must reside with it's data source(s). Every time the data view is accessed the records are created new which means that processing time goes up. Procedures that require multiple passes through the data may experience a degradation in accuracy.

Also, even though SAS datasets and SAS data views are different, you can't create a view in a library with an existing dataset of the same name, and vice/versa.

There are a number of ways of creating a view; the DATA step, PROC SQL, and SAS/ACCESS. We will look at just the first 2.

To create a view In the DATA step, on the DATA statement add the View= option after the final dataset name. This option tells

the SAS system to compile, but not execute the step and to store the compiled code in the view named. The view name must be the same as one of the dataset names, and you can only create one data view in one DATA step.

```
DATA work.one / VIEW=work.one;
     SET work.two;
RUN;
```

To create a view in PROC SQL, specify VIEW in the CREATE clause of an SQL statement.

```
PROC SQL;
     CREATE VIEW work.one AS
     SELECT *
     FROM work.two;
QUIT;
```

Using a data view is the same as with a SAS dataset. You can use them as a source in another DATA step.

```
DATA work.new;
     SET work.one; * This is a View;
RUN;
```

When the DATA step is compiled and run, the data view is accessed, each record is created in memory using the compiled code in the view, any other processing is done that is required, and the record is output to the new dataset.

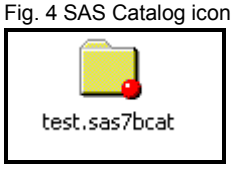
Where statements and other requirements may be placed upon views successfully.

```
PROC PRINT data=work.one
     Where id='RDU';
RUN;
```

SAS CATALOGS

Catalog files do not store data. But they do store many other different kinds of information, some of which can relate to your data. For example, catalogs can be used to store PROC REPORT templates, user-defined formats, completed SAS/GRAPH charts, and Compiled Macros. The catalog file has a .SAS7BCAT extension (See Figure 4).

Fig. 4 SAS Catalog icon



The SASHELP library has a large number of catalogs containing SAS system information. You can create your own catalogs .

The structure of a catalog looks very much like a Windows directory with files of different types coexisting together. Catalogs store information in the form of discrete units called entries. Each has it's own entry type that identifies it's purpose to the SAS system. When you wish to access or create a particular entry, you use standard SAS naming conventions. For example, if you wanted to create a PROC REPORT template called Report9 in your Reports catalog in the Mylib library, the PROC REPORT statement would look like this:

```
PROC REPORT DATA = Mylib.one
     OUTREPT = Mylib.Reports.Report9.Rept;
```

You can use PROC FORMAT to store your User-Defined formats in a catalog in the Work library called FORMAT or in a calalog of your naming in a permanent library.

```
PROC FORMAT;
     VALUE $myfmt    'A'='ALPHA'
                     'B'='BETA'
                     'C'='KAPPA';
     RUN;
```

With this code, an entry called $myfmt is made into the FORMAT catalog. However, this is still temporary, stored in the WORK library. To make it permanent, add the LIB= option to the PROC FPRMAT statement, specifying a permanent library and optionally a catslog name(if no catalog name is specified, SAS names it FORMATS). A new FORMAT catalog will be created there with each new user-defined format stored as a separate entry.

```
PROC FORMAT LIB=Mylib.Myformats;
     VALUE $myfmt    'A'='ALPHA'
                     'B'='BETA'
                     'C'='KAPPA';
RUN;
```

The SAS system doesn't automatically check your permanent libraries for Format catalogues. The FMTSEARCH= option identifies other libraries for the SAS system to search for permanent format cstalogs.

```
OPTIONS FMTSEARCH=(Mylib.Myformats);
```

You may specify multiple libraries and catalogs in the FMTSEARCH list. Then, when SAS encounters a FORMAT statement, the SAS system will search each catalog starting with SAS's own formats, any temporary (WORK) format catalogs, then your permanent format catalogs.

SAS INDEXES

The index file stores indexes. It is very closely aligned with the SAS data set, and SAS treats it as part of the dataset. If you add or delete observations or modify values, the index is automatically updated. The index file has a .SAS7BNDX extension (See Figure 5). Even though the index file is connected with the SAS data set, the index file is not visible in the SAS Explorer window, but can be seen in Windows Explorer or in PROC DATASETS output.

Fig. 5 SAS Index file icon

A dataset's index works like the index of a book, giving the reader the ability to find a particular topic immediately rather than starting at the beginning of the book, and checking each page. The index stores the individual values of a chosen key variable or combination of variables, along with the positions of the corresponding observations. So you can look up any particular observation quickly by specifying it's key variable's specific value. The index identifies the exact location of the observation in the dataset.

There are a number of benefits associated with indexes. The SAS system can use indexes to improve the performance of WHERE statements, BY statements, and for SET and MODIFY statements (using the KEY= option). The programmer can save time since a PROC SORT is not required. Also, after a dataset is set up with one or more indexes, any additions, deletions, or modifications of data are automatically reflected to the appropriate index, saving the programmer from re-creating the indexes each

time.

However, indexes aren't perfect. Creating and maintaining indexes take CPU time. And the more indexes there are, the more CPU time will be taken per data transaction to update them. Also, using an index to read observations may actually increase the number of I/O (input/output) requests.

There are 2 primary types of indexes, simple (one key variable) and composite (multiple key variables), but for one dataset, all indexes are stored in just one index file. There are several ways of creating an index. One way is in the DATA step.

```
DATA Work.one (index=(ssn));
     Set Work.two;
RUN;
```

Here a simple index has been created using the ssn variable as a key variable. Since there is only one key variable for this particular variable, the name of the index is the name of the variable, ssn. When creating a composite index, you can not take the name of one of the variables, but must specify a separate name.

```
DATA Work.one (index=(Myindex=(last first));
     Set Work.two;
RUN;
```

Some uses of indexes are transparent to the programmer. For example, if you try to print an unsorted dataset with a PROC PRINT step and a BY statement, it will fail.

```
PROC PRINT DATA=Work.one;
     BY ssn;
RUN;
```

PROC PRINT needs a sorted dataset to do this. However, if the dataset has an index using the BY variable as a key variable, PROC PRINT will be able to use that to print the dataset. Similarly, a WHERE statement can use an index to more efficiently access the requested observations.

```
PROC PRINT DATA=Work.one;
     WHERE ssn GT '001-20-3456';
RUN;
```

Since a SAS data view contains no data, it cannot have index files associated with it.

SAS AUDIT FILES

The audit file is another optional SAS file that you can create to record changes that take place in a SAS dataset. Every time the dataset is opened for modification, addition, or deletion, one or more records are added to the audit file recording who did what and when. The audit file has a .SAS7BAUD extension (See Figure 6). Even though the audit file is connected with the SAS data set, the audit file is not visible in the SAS Explorer window. However, audit files can be seen in Windows Explorer or in PROC DATASETS output. Audit files were new with Version 7.

Fig. 6 SAS Audit file icon

Many businesses require some type of audit trail. The audit file in SAS allows programmers to maintain a historical record of changes to a dataset showing every change that has happened to observations from the time it is entered into the dataset until it is deleted. Additionally, the audit file is the only place that stores observations rejected from inclusion during updates if Integrity Constraints are in place.

The audit file must reside in the same SAS library as it's dataset, and there can only be one audit file per dataset. The audit trail is not recommended for datasets that are copied, moved, sorted in place, or transferred to another O.S. Also be aware that use of an audit file will increase CPU processing time and may take up significant storage space, depending on how many changes you make to the dataset. Finally, an audit trail can not be applied to a SAS data view.

Starting an audit trail is done with PROC DATASETS and the AUDIT statement.

```
PROC DATASETS LIB=Work;
     AUDIT One;
     INITIATE;
     USER_VAR reason_code $ 20;
QUIT;
```

This code opens a new audit file for the SAS dataset, Work.one, of the same name (Work.one), but with the .SAS7BAUD extension. The USER_VAR statement adds a variable, reason_code, to the audit trail for users to enter reasons for data changes.

Now, any time the dataset is opened for updates, be it through the DATA step, PROC SQL, or even a Data Entry Screen, a record is entered into the audit file for each change made. In addition, access is available to the audit file's variable "reason_code" as if it was a data set variable. The below PROC SQL code enters one record into the dataset and the audit file is updated automatically.

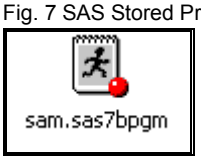
```
PROC SQL;
     INSERT INTO Work.one
         SET ssn = '002-03-4567'
             Last = 'Cates'
             First = 'Randall'
             Reason_code = 'Add new employee';
QUIT;
```

Since a SAS data view contains no data, it cannot have audit files associated with it.

SAS STORED PROGRAMS

The SAS stored program stores compiled DATA step code. This type of file can be stored in a SAS library and can be executed without having to be recompiled. The stored program has a .SAS7BPGM extension and is visible in the SAS Explorer window (See Figure 7).

Fig. 7 SAS Stored Program icon



Stored compiled programs are of advantage in production job settings, especially when the DATA step code is complex and/or contains many statements because the SAS system doesn't need to re-compile the code every time the files are executed. However, stored programs contain only DATA step code, no PROCs or Global statements. Stored programs will not work on different Operating Systems from where they were created.

To create a stored program, type your DATA step as usual, first making sure that the DATA step works correctly.

```
DATA work.one;
     SET Mylib.two;
     <other code>;
RUN;
```

Once you are assured that the code will work the way it's supposed to, add the PGM= option to the DATA statement, giving it a libname and filename. Then re-submit the code.

```
LIBNAME Programs 'C:\Workshop\Stores';
DATA work.one / PGM=Programs.onestep;
     SET Mylib.two;
     <other code>;
RUN;
```

At this point the DATA step is compiled and the resulting machine code is stored in the Programs library with the name of Onestep. The DATA step is not executed at this time.

To use the stored program, use it in a new DATA step. Be sure to submit appropriate LIBNAME statements first.

```
LIBNAME Programs 'C:\Workshop\Stores';
LIBNAME Mylib 'C:\Workshop\My Data';
DATA PGM=Programs.onestep;
RUN;
```

The stored program is recalled, and executed and the new data set is created. Note that in the Log window you only see a note of the results of the DATA step. The underlying DATA step code does not print to the log.

Optionally you can add the DESCRIBE statement which prints the underlying code to the log window. However, this will also stop the code from executing. So if you do add this statement you also need to add the EXECUTE statement which will execute the code.

```
LIBNAME Programs 'C:\Workshop\Stores';
LIBNAME Mylib 'C:\Workshop\My Data';
DATA PGM=Programs.onestep;
     DESCRIBE;
     EXECUTE;
RUN;
```

Once the stored program has been created, you may change the names of the Input and Output datasets, either through different LIBNAME statements or the REDIRECT statement at execution time.

```
LIBNAME Programs 'C:\Workshop\Stores';
LIBNAME Mylib 'C:\Some Other Path';
DATA PGM=Programs.onestep;
RUN;
```

With the above code, we have set the Library to a different path. However, we still need to access a dataset of the same name as the one stored (Mylib.two). To change either the Input dataset or the Output dataset names, use the REDIRECT statement.

```
LIBNAME Programs 'C:\Workshop\Stores';
LIBNAME Mylib 'C:\Workshop\My Data';
DATA PGM=Programs.onestep;
     REDIRECT INPUT Mylib.two=Mylib.three;
RUN;
```

Use the REDIRECT statement with caution, however. The new Input dataset must contain the same variables and attributes as the original dataset.

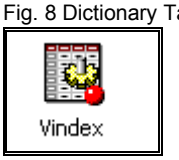
Changing the name of the Output dataset is similar to the last example. In the below example the Dataset is created in the permanent library (Mylib) rather than in the Work library.

```
LIBNAME Programs 'C:\Workshop\Stores';
LIBNAME Mylib 'C:\Workshop\My Data';
DATA PGM=Programs.onestep;
     REDIRECT OUTPUT Work.one=Mylib.one;
RUN;
```

DICTIONARY TABLES

Dictionary Tables are SAS Data Views. As such they have the same structure as regular DATA Views, and also have a .SAS7BVEW extension.

Fig. 8 Dictionary Table icon



Dictionary tables are special, however, because they contain lists of information related to the current SAS session. Some things that you can find out from dictionary tables are:

- All SAS Libraries available
- All SAS datasets available
- All SAS system options for the current session
- All SAS macros available
- All external files currently in use or available.

Dictionary tables can be seen in the SASHELP library and are usually accessed through the PROC SQL step. Using a DESCRIBE TABLE step you can look at the columns available in any one table. Be aware that some of these tables access a large amount of information, depending on how you use SAS. So it often helps to use WHERE clauses or statements to subset the report.

For example, There is a dictionary table called DICTIONARY.CATALOGS. The following code accesses and describes the columns within it.

```
PROC SQL;
     DESCRIBE TABLE dictionary.indexes;
```

The results (sent to the Log window) are shown below.

```
create table DICTIONARY.CATALOGS
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   objname char(32) label='Object Name',
   objtype char(8) label='Object Type',
   objdesc char(256) label='Object
     Description',
   created num format=DATETIME
     informat=DATETIME label='Date Created',
   modified num format=DATETIME
     informat=DATETIME label='Date Modified',
   alias char(8) label='Object Alias'
  );
```

These results show that the Catalogs data view accesses 8 different columns; libname, memname, memtype, objname, objtype, objdesc, created, modified and alias. And it shows the attributes of each.

Now we can use a SELECT statement to view information about all catalogs available in our SAS session. However, rather than look at all of the catalogs we can add a WHERE to limit our results.

```
PROC SQL;
     SELECT * FROM Dictionary.catalogs
     WHERE LIBNAME eq 'Mylib';
QUIT;
```

This code reduces the results to just the catalogs in one library (Mylib).

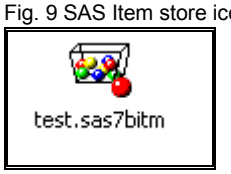
These tables are also available using SAS code. However the names are a bit more confusing. First, The libref DICTIONARY is only available in PROC SQL code. In SAS code you need to use the SASHELP libref. Then the names that you see in the SASHELP window are different from the names you use in PROC SQL. To access the Catalogs Dictionary View with PROC CONTENTS, you want to look at SASHELP.VSCATLG.

```
PROC CONTENTS DATA=sashelp.vscatlg;
RUN;
```

SAS ITEM STORES

The Item Store file was also introduced in Version 7. It is a member in a SAS library, though it is hidden from view in the SAS Explorer window. The item store file has a .SAS7BITM extension (See Figure 9). Use of the item stores is considered an advanced topic so we won't discuss ways of modifying them.

Fig. 9 SAS Item store icon



As the name suggests, the item store stores things. Currently there are 2 uses for item stores. The SAS registry is stored in 2 item stores (SASUSER.REGSTRY.ITEMSTOR and SASHELP.REGSTRY.ITEMSTOR), and ODS templates are stored in another (SASHELP.TMPLMST.ITEMSTOR). It is highly recommended that you leave the SAS registry alone, since it controls most everything about your SAS session.

The structure of the Item store is very similar to the Unix operating system, with directories and subdirectories and members within them. However, you can't work with the members directly in SAS since the item store is only available from C programs, not SAS code. Having said that, you can work with item stores indirectly.

The SAS Registry stores configuration data about your SAS session and about various applications that are available to you. You can use the SAS Registry Editor window to view the contents of the registry, modify items, export and import items, etc. There is also a procedure, PROC REGISTRY, to accomplish these tasks.

The ODS template store is available from the SAS Results window (Not the Output Window). Click on the Results Window and there is a new Templates menu item added at the top of the View menu. You can click on that and SAS will open an Explorer style window showing you the TMPLMST item store and all items in there. Do not attempt to modify any templates stored there.

If you wish to create some different ODS templates, you can copy an existing template from the TMPLMST item store into a new item store and modify it there.

SAS MULTIDIMENSIONAL DATABASES (MDDB)

A multidimensional database is yet another specialized storage file that stores data from a data warehouse or other files in a multi-level format. Base SAS has no tools to create MDDBs. Other SAS products, such as SAS/Warehouse Administrator®, create MDDBs but you can still access and view them in the SAS Explorer window. The MDDB file has a .SASBMDB extension.

Fig. 10 SAS MDDB icon

the MDDB does not store it's data in a 2-dimensional table structure. Instead it consists of a Multidimensional model of data consisting of classification variables and summary statistics. The combinations of classification variables define **crossings**. A crossing is a unique combination of classification variable values and the statistics requested at that point.

For instance, suppose you have a database of personnel information. From this database you can choose some variables to be classification variables, perhaps jobcode, age, and Education. Also, there could be some numeric variables for which we want to view various statistics, perhaps Salary and Bonus. The MDDB can then be constructed by defining the classification variables and requesting various statistics(mean, median, standard deviation, etc.) for the numeric variables. Therefore a crossing would now be found in the MDDB for Jobcode(PILOT), Age(35-40), and Education(Grad School) and would have the summary statistics of Mean, Median and Standard Deviation for Salaries for all observations meeting that criterion.

While you can't create an MDDB in BASE SAS, you can access and view one. MDDBs are valid file types for SAS libraries, so you can see and click on one in the SAS Explorer Window. SAS will open up a Viewtable window showing one of the summary crossings in there. You can then right-click anywhere in the window and open up the MDDB layout window, which shows all classification variables make up the MDDB. At this point you can choose one or more combinations of classification variables to see another view of the data.

CONCLUSION

This has been a very short overview of SAS files available to the SAS programmer. Due to time and space constraints this paper has been only able to touch briefly on each file type. There is much more that could be said, perhaps enough for a full paper, or more on each file type.

A proper understanding of different SAS file types and when to use them and when not to use them can help in improving your use of the SAS system to get the job done.

There is a lot more information available for those who need to know more about any particular SAS file type. Check the References at the end of the paper for more information. Also, SAS Technical Support (919-677-8008 or email support@sas.com) can be invaluable to answer questions or to clear up confusions.

REFERENCES

SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Institute Inc., *SAS OnlineDoc®*, Version 8, Cary, NC: SAS Institute Inc., 1999.

SAS Institute, Inc., *The Complete Guide to the SAS® Output Delivery System, Version 8*, Cary, NC: SAS Institute, Inc., 1999. 310 pp.

Olson, Diane (2000), "Power Indexing: A Guide to Using Indexes Effectively in Nashville Releases", *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference CD*, Paper 124.

Riba, S. David (1999), "The DATA Statement: Efficiency Techniques", *Proceedings of the Twenty-Fourth Annual SAS ® Users Group International Conference*, Paper 71.

CONTACT INFORMATION

Your comments and questions are much valued and encouraged. Randall can be reached at:

Randall Cates
SAS Institute Inc.
MCI Building, Suite 550
100 South Fourth Street
St. Louis, MO 63102
314-421-6364 ext. 8506
email: Randall.Cates@SAS.com®