Paper 164-2014

Nitty Gritty Data Set Attributes

Diane Olson, SAS Institute Inc., Cary, NC

ABSTRACT

Most programmers are familiar with the directive, "Know your data." But not everyone knows about all the metadata stored in a SAS® data set or understands what to do with this information. This paper covers the majority of these attributes, how to obtain them, why they are important, and what you can do with them.

For example, data sets that have been around for a while might have an inordinate number of deleted observations that you are carrying around unnecessarily. Or you might be able to quickly check to determine whether the data set is indexed—and if so, by what variables—to increase your program's performance. Also, engine-dependent data such as owner name and file size is found in PROC CONTENTS output, which is useful for understanding and managing your data. You can also use ODS OUTPUT to use the values of these many attributes programmatically. This paper shows you how.

INTRODUCTION

You can view data set attributes in several ways. In this paper we use the output of PROC CONTENTS from SAS 9.4. The version is important because the number of reported attributes increases as features are added. If you are using a version prior to SAS 9.4, you might not have access to some attributes as yet.

PROC CONTENTS divides output into three main categories: general attributes, engine/host attributes, and variable attributes. This paper covers the first two categories. For some attributes, PROC CONTENTS shows you whether the attribute is on or off or what its value is, no matter what that value might be. For other attributes, PROC CONTENTS output shows the information only if the attribute is in use. Encryption is one example of this. If a data set is not encrypted, *Encryption* does not appear in the CONTENTS output.

This paper uses ODS OUTPUT to create data sets that contain attributes of interest. You can also specify an OUT= option on the PROC CONTENTS statement. However, with the advent of ODS OUTPUT, the OUT= data set has not been updated with new attributes. ODS OUTPUT data sets are more flexible and contain all the information we need. An OUT2= data set also contains useful information about indexes and integrity constraints. DICTIONARY tables also include attribute information, but this paper concentrates on the attributes from PROC CONTENTS.

You might want to choose a favorite data set, get the PROC CONTENTS output, and follow along.

GENERAL DATA SET ATTRIBUTES

In *italics* in this section is a list of all general attributes with information about each. While some attributes seem obvious—for example, *Name* and *Created*—there are still important considerations. Attributes with an asterisk (*) have their own attributes, not discussed in this paper.

Name: If your data set name is not compatible with SAS 7 naming conventions, the data set is not accessible unless you set the VALIDMEMNAME option to EXTEND. (The default value for this option is COMPAT.) Knowing what files exist in the physical location of your libname lets you determine whether you are processing the entire library.

```
/* Set VALIDMEMNAME to EXTEND to create data set with special characters. */
options validmemname=extend;
data 'bed&breakfast'n;s=3;run;

/* Set VALIDMEMNAME to COMPAT and look at listing of files in directory. */
options validmemname=compat;
proc datasets;run;quit;

/* Set VALIDMEMNAME to EXTEND and look at listing again. */
options validmemname=extend;
proc datasets;run;quit;
```

Assuming that the WORK library is empty before you run the above example, here is what the log shows after the first PROC DATASETS statement:

```
WARNING: No matching members in directory.
```

The second PROC DATASETS statement shows that the bed&breakfast data set exists, because VALIDMEMNAME is set to EXTEND. If the data set name includes national characters, you would see the same results.

Created and *Last Modified*: By default, the *Created* attribute holds the DATETIME when SAS created the data set. By default, *Last Modified* holds the DATETIME when SAS last modified it. A modified DATETIME can indicate addition of an index or other modification of metadata, as well as data modifications.

These DATETIMEs might not hold the same value that the operating system lists for the created or modified DATETIME. When you use SAS to modify the data set, the internal *Last Modified* DATETIME changes. However, if you use operating system commands to move or update the file (not recommended), the internal DATETIME is not modified.

You can change the internal DATETIMEs in other ways. PROC DATASETS lets you set the *Created* date to a previous DATETIME using the DTC option on the MODIFY statement. This does not change operating system DATETIME.

```
/* Set MYDATA’s Created datetime to Groundhog Day. */
proc datasets lib=work nolist;
   modify mydata / dtc='02FEB2014:00:00:00'dt;run;
quit;
```

PROC COPY, PROC CPORT, and PROC UPLOAD allow internal DATETIMEs to be retained on a copy of a data set by using the DATECOPY option. PROC MIGRATE does this automatically. None of these affect operating system DATETIMEs.

Also, the directory listing through PROC DATASETS or PROC CONTENTS DATA=_ALL_ display operating system DATETIMEs, not internal DATETIMEs.

Protection: This attribute indicates whether access to this data set requires any passwords. The kinds of passwords available are READ, WRITE, and ALTER, which are also the values that appear in the *Protection* field.

On a related topic, if you do not want to put clear-text passwords in your programs, you can use PROC PWENCODE to obtain an encoded password. Of course, if someone copies or pastes that encoded password, they can still access the data set, but someone likely cannot remember it only by viewing it over your shoulder. You can also write the encoded password to a file and then use it from that file.

```
/* Create an encoded password. */
proc pwencode in='my password';run;
```

From the above example, here is the encoded password that is returned in the SAS log:

```
{SAS002}DBCC571245AD0B31433834F80BD2B99E16B3C969
```

Type: This value is set using the data set TYPE= option. The value is blank for most data sets. You can use the type field to indicate how the data set is structured. Some SAS/STAT® procedures check the value to determine whether the data set can be used in a particular way. Some values include CORR, COV, and FACTOR, although there are others. Refer to the *SAS/STAT® User’s Guide* to determine if you can use this attribute.

Encryption: Until SAS 9.4, SASProprietary encryption, which the ENCRYPT=YES data set option enables, was the only type of encryption that SAS supported for Base SAS. Now Advanced Encryption Standard (AES) encryption is also supported with the ENCRYPT=AES data set option specification. AES produces a strong encryption using a key value of up to 64-bytes long. You specify the key value with the ENCRYPTKEY= data set option. The key value is not stored on the data set. You must specify the key value any time you access the data set, although you can replace or delete the data set without the key value. If you lose the key value, you cannot access the data set. This note is written to the log when a data set is created with AES encryption:

```
NOTE: If you lose or forget the ENCRYPTKEY, there will be no way to open the file
and recover the data.
```

Generations: In a generation group, all data sets have the same root member name but different version numbers. Generation groups are available only with the Base SAS engine. In a generation group, the base version has two generational attributes: The *maximum number of generations* states how many generations of the group can exist at one time, while the *next generation* gives the number of the next generation that would be created. The generations themselves have an attribute that lists their *generation number*. The number that is listed is the absolute generation number rather than a relative one. You can access a particular version of a generation group by specifying the generation number when using the GENNUM= data set option.

```
/* View CONTENTS information for the second version of MYDATA. */
proc contents data=mydata(gennum=2);run;
```

Managing generation groups through operating system commands instead of through SAS can render your generation group unusable.

Label: A data set label holds up to 256 bytes. It typically contains a description of the data set, although the contents are determined by the programmer. By default, the label is blank. The label automatically prints in the DATASETS procedure output when the DETAILS system option is set. See the section below about using DETAILS.

Audit Trail: An audit trail logs modifications on a data set and is available only with the Base SAS engine. Four different audit trail attributes exist. All are customizable when the audit trail is created with the AUDIT statement of the DATASETS procedure. *Admin_Image* indicates whether administrative events such as creation, suspension and resumption of the audit trail are logged. *Before_Image* indicates whether the observation is logged to the audit trail before any updates are made to it. *After_Image* indicates a value of an observation after it is added, deleted, or updated in the data set. *Error_Image* indicates the value of an observation that failed to be added or updated due to an ERROR. In such cases, the audit trail observation also holds the error message in the _ATMESSAGE_ value. Keep in mind that you can also view the audit trail data set itself using the TYPE=AUDIT data set option. Here is an example:

```
proc print data=mylib.mydata(type=audit);run;
```

The number of audit trail variables automatically print in the DATASETS procedure output when the DETAILS system option is set. See the section below about using DETAILS.

Observations: This attribute shows the number of observations in your data set. When performance would be impacted to indicate the number of observations, such as with the TAPE engine and some SAS/ACCESS® engines, the value will be a missing value. For the Base SAS engine, if there is a missing value for this attribute, the number of observations may have gone over the maximum observation count limit. For more information on this and an important related topic, see EXTENDOBSCOUNTER below in the Base SAS engine attribute section.

The number of observations will automatically print in the DATASETS procedure output when you set the DETAILS system option. See the section below about using DETAILS.

*Variables**: The general attribute indicates the number of variables in this data set. All variables in a data set and their related attributes are also part of a data set's attributes. In short, a variable has these attributes: Name, Length, Type (character or numeric), Format, and Informat.

The number of variables automatically prints in the DATASETS procedure output when the DETAILS system option is set. See the section below about using DETAILS.

*Indexes**: The general attribute indicates the number of indexes (simple and composite combined) on the data set. There are many reasons that an index might make your program execute more quickly, but you must balance that possibility with resources that are used to maintain the index. Several online white papers describe when and how to use indexes. However, one tip is to pay attention to the centiles (cumulative percentiles) for Base SAS engine data sets. You can see this attribute by specifying the CENTILES option on PROC CONTENTS. The centiles hold the distribution of the index values. Along with other information, SAS uses these centile values to determine when an index should be used. By default, centiles values are updated when 5% of the data set has been changed. You can set that percentage using the PROC DATASETS INDEX UPDATECENTILES option if you want to update more or less frequently.

Indexes are not used when a data set is accessed through Cross Environment Data Access (CEDA).

The number of indexes automatically prints in the DATASETS procedure output when the DETAILS system option is set. See the section below about using DETAILS.

*Integrity Constraints**: Integrity constraints are a set of data validation rules that govern which data values can exist in a particular data set. General and referential integrity constraints are available with the Base SAS engine.

Referential integrity constraints connect two or more data sets, linking data values in the data sets. Like indexes, this topic is involved, but online white papers exist that can help you learn about integrity constraints. Here are a couple of important things to know: Although it is never a best practice to use operating system commands to manage SAS files, it is especially dangerous when referential integrity constraints are involved. Referential integrity constraints will no longer function properly if operating system commands move the data sets that are involved. Also, updating data sets that are linked by referential constraints becomes slower with each additional data set. In such conditions, SAS has to open multiple data sets and test the validity of any updated value with the referencing data set(s).

PROC CONTENTS can produce an entire data set of information on the indexes and integrity constraints on a data set by using the OUT2= option. This data set contains many pieces of information, including a variable that contains the SAS code to recreate each index or integrity constraint.

Integrity constraints are not used when a data set is being accessed through CEDA.

Observation Length: The length of an observation might not be the sum of the total of the variables' lengths. There can be padding depending on the type of the variables and the alignment on the particular host. Indeed, the observation length might differ from one host or engine to another, even if the data is the same.

Deleted Observations: This attribute shows how many observations are marked for deletion in the data set. If an observation is deleted, SAS marks that observation as no longer existing—it is not actually removed from the data set at that time. If it were, the data set would be rewritten every time an observation was deleted. Instead, the data can no longer be accessed.

If your data set has a large number of deleted observations, it uses disk space unnecessarily and SAS might take longer to access your active observations. Deleted observations must still be read from the disk and logic must be used to skip them when returning active data. To remove the deleted observations from the data set, you can use a method such as PROC COPY to copy the data set and retain other data set attributes that you need.

Use this code to determine how many deleted observations are in each data set in a library:

```
ods output attributes=deleted(keep=member label2 nvalue2
   where=(label2 contains 'Deleted' and nvalue2 > 0)
   );
proc contents data=lib1._all_;run;
proc print data=deleted;
   format nvalue2 1.;
run;
```

The ODS object name, Attributes, is used above. It is the ODS object table name for the first section of information that PROC CONTENTS generates. Using the ODS OUTPUT statement for Attributes creates a data set containing the first section of PROC CONTENTS information. These variables are created by default: member, label1, cvalue1, nvalue1, label2, cvalue2, and nvalue2.

```
Obs           Member            Label1                cValue1

 1    WORK.MYDATA (gennum=2)    Data Set Name         WORK.MYDATA (gennum=2)
 2    WORK.MYDATA (gennum=2)    Member Type           DATA
 3    WORK.MYDATA (gennum=2)    Engine                V9
 4    WORK.MYDATA (gennum=2)    Created               02/02/2014 11:13:40
 5    WORK.MYDATA (gennum=2)    Last Modified         02/02/2014 11:13:40
 6    WORK.MYDATA (gennum=2)    Protection
 7    WORK.MYDATA (gennum=2)    Data Set Type
 8    WORK.MYDATA (gennum=2)    Generation Number     2
 9    WORK.MYDATA (gennum=2)    Label
10    WORK.MYDATA (gennum=2)    Data Representation   WINDOWS_32
11    WORK.MYDATA (gennum=2)    Encoding              wlatin1  Western (Windows)


Obs        nValue1    Label2                cValue2        nValue2

 1               .    Observations             1          1.000000
 2               .    Variables                1          1.000000
 3               .    Indexes                  0                 0
 4      1706958820    Observation Length       8          8.000000
 5      1706958820    Deleted Observations     0                 0
 6               .    Compressed               NO                .
 7               .    Sorted                   NO                .
 8        2.000000                                               0
 9               .                                               0
10               .                                               0
11               .                                               0
```

Output 1. PROC PRINT output from ODS Output Attributes Table from PROC CONTENTS

Member holds the libname and member name of the particular data set. Remember that there are two columns of information that PROC CONTENTS generates in the listing. All left-hand column information is held in the Label1, cValue1, and nValue1 variables. Right-hand column information is in the Label2, cValue2, and nValue2 variables. The Label1 and Label2 variables hold the attribute name. Character values of the attributes are held in the cValue1 and cValue2 variables, while the numeric values are held in the nValue1 and nValue2 variables.

The Deleted Observations value is printed on the right side in the listing, so the data set named ‘deleted’ has the variables Label2, cValue2, and nValue2 populated. The label holds the name of the attribute. In our case, Label2 holds ‘Deleted Observations’. In this case, our attribute is a numeric attribute. We are interested in the value for nValue2, specifically when it is greater than 0, meaning that a particular data set has more than zero deleted observations. Using the libname of LIB1 plus _ALL_ causes all the members in the library to be processed.

With deleted observations, compressed data sets can work differently. If the REUSE=YES system or data set option is set, a compressed data set reuses the space of a deleted observation by writing a new observation into that space. Therefore, the *Deleted Observations* attribute for a compressed data set with REUSE=YES holds a missing value. The REUSE= option is available only with the Base SAS engine.

Compressed: Two options compress a data set on disk: CHAR (or YES) or BINARY. CHAR uses Run Length Encoding compression, and BINARY compression uses the Ross Data Compression algorithm. Compression can decrease the amount of disk space required to store a data set. Compressing data means that fewer reads or writes are required to get or put the data from or to the disk. However, you must balance this with the increased processing that is necessary to decompress the observations for use. Results are data-dependent, so you should test with your own data to determine which is best in your case. Here is a simple example of creating a BINARY compressed data set:

```
data binCompress(compress=binary);
...
run;
```

Reuse: This attribute applies only to compressed Base SAS engine data sets. If the option value is set to YES, any space a deleted observation leaves may be filled with a new observation, instead of adding the new observation to the end of the data set. If you care what order your observations are written in the data set, i.e. you want the first observation to be observation #1 in the data set, then you should set the option value to NO.

Point to Observations: This attribute reflects the setting of the POINTOBS option for Base SAS engine compressed data sets. The default value, YES, is to allow random access. If you do not require random access to the observations (pointing to observations) and can use sequential access, POINTOBS=NO can improve performance when updating or adding observations to your data set. REUSE=YES always has precedence over POINTOBS=YES, as these two are mutually exclusive.

*Sorted**: Sorting a data set might speed up processing of your data, depending on what you are doing. A number of valuable online papers about PROC SORT can help you determine what is best in your case. An important point to remember is that if you use the linguistic collation capabilities, your data remains sorted when moved across operating systems. Linguistic collation is indicated by using the SORTSEQ=LINGUISTIC keywords with the SORT procedure.

If you are not using linguistic collation, copying your data set can result in a data set that is sorted for the original operating system. For example, imagine the situation where your data resides on z/OS.

```
Sort Information

Sortedby       x
Validated      YES
Character Set  EBCDIC
```

Output 2. Output from a PROC CONTENTS on a data set sorted on z/OS

The Character Set is EBCDIC, which makes sense for z/OS. Use PROC COPY or PROC CPORT and PROC CIMPORT to move your data to a UNIX platform and again perform a PROC CONTENTS. The character set is still EBCDIC, and the data remains sorted that way. The actual data will have been converted to ASCII, however. In this case, the data set must be sorted again to be used as you would expect from a sorted data set with an ASCII character set.

Data Representation: By default, SAS creates a data set with the data representation of the CPU that is running SAS. Different operating environments use different standards for storing data. For example, a bit-by-bit comparison of the same data in a data set on Windows versus Solaris would show distinct differences. The *Data Representation* identifies which operating system created the data set. CEDA uses the information to read the data set on a non-native operating system.

CEDA is used automatically when needed. Unless NONOTES is in effect, you will see this NOTE when a non-native data file is accessed:

```
NOTE: Data file WORK.B.DATA is in a format that is native to another host, or the file encoding
      does not match the session encoding. Cross Environment Data Access will be used, which
      might require additional CPU resources and might reduce performance.
```

As this note indicates, CEDA uses additional CPU resources. Restrictions for accessing a data set through CEDA include no update access, no indexes or integrity constraints, and possible loss of numeric precision. For more details, search for “SAS® File Processing with CEDA” in *SAS® Language Reference: Concepts* on support.sas.com.

Encoding: The *Encoding* of a library member indicates the encoding map type that is used to represent a character set. By default, SAS sets the session encoding based on the operating system on which you are running. Using different maps result in different *code points* to represent characters. For example, the capital letter A is represented in ASCII as a hexadecimal 41. The same letter in EBCDIC is represented with a hexadecimal C1. Knowing the encoding allows the code points to be interpreted correctly.

Much like the *Data Representation*, CEDA must be used to read a data set when the *Encoding* differs from the SAS session encoding. Additional CPU resources are used, and the same limitations for CEDA are in effect as mentioned above.

*Information Maps**: Information maps are metadata definitions of data sources, working somewhat like a data set view. Instead of WHERE clauses, information maps have filters that identify particular pieces of data. The filter names are displayed in PROC CONTENTS. Be aware that the data set variable names displayed in the filter names might have been altered based on the setting of the VALIDVARNAME setting. Because the original names might have a length greater than allowed or contain special characters, the names might not fit the current VALIDVARNAME settings. For display purposes, names too long for the VALIDVARNAME setting are uniquely truncated with an appended number, if necessary. Special characters in names that the current VALIDVARNAME setting does not allow are replaced by underscores in the PROC CONTENTS output.

*Extended Attributes**: Extended Attributes are customizable metadata for your Base SAS engine data set. Like indexes and integrity constraints, extended attributes exist only if you define them on your data set. They can hold any attribute that you want to store: They are attributes unique to your data that you decide are important to keep. Some engines do not support Extended Attributes, so be aware of that when copying data to other engines. Unlike attributes such as indexes and integrity constraints, extended attributes can still be used if accessing the data through CEDA.

THE DETAILS OPTION

The DETAILS option specified as a system option or on the PROC DATASETS statement allows number of observations, number of variables, number of indexes, number of audit trail observations, and number of audit trail variables from the data set to be printed. However, this information comes with a price. Normally, PROC DATASETS does not open all data sets in a library. To retrieve the information for DETAILS requests, it must. Opening each data set impacts performance.

PRESERVING YOUR ATTRIBUTES

If you want to create a copy of your data set, including the data and the data set attributes, be aware that the DATA step might not keep the data set attributes. However , there are procedures that focus on keeping your data set attributes and your data: PROC MIGRATE, PROC COPY, PROC CPORT, PROC CIMPORT, PROC UPLOAD and PROC DOWNLOAD. PROC MIGRATE also retains integrity constraints by default, though the remaining procedures require use of CONSTRAINT=YES. PROC COPY with the default CLONE option and CONSTRAINT=YES clones your attributes. Adding NOCLONE causes PROC COPY to act somewhat differently. See PROC COPY documentation for more information.

ENGINE/HOST-DEPENDENT DATA SET ATTRIBUTES

Although general data set attributes are available for all hosts and engines, there is also information particular to the engine, host, or both that is displayed by PROC CONTENTS.

BASE SAS ENGINE

Several pieces of information are printed for the Base SAS engine, as shown in Output 3. The pieces of information we are focusing on are the number of times the data set has been repaired and the EXTENDOBSCOUNTER information.

```
              Data Set Page Size            65536
              Number of Data Set Pages      1
              First Data Page               1
              Max Obs per Page              8061
              Obs in First Data Page        100
              Number of Data Set Repairs    1
              Last Repair                   14:15 Saturday, January 25, 2014
```

Output 3. Partial output from a PROC CONTENTS on a Base SAS engine data set

NUMBER OF DATA SET REPAIRS

Perhaps you have encountered a situation when you ran out of disk space while writing a data set or found that you got an ERROR when writing over a bad sector on disk.

In such situations, you would have seen an ERROR that indicates the problem, and your data set would have been marked as damaged. The DLDMGACTION option setting indicates the action that is taken when a library member is damaged: FAIL, ABORT, REPAIR, NOINDEX, or PROMPT. If your DLDMGACTION option is set to the REPAIR default, you might have seen the following message when a data set is automatically repaired:

```
NOTE: File FOO.FILE1.DATA is damaged.
NOTE: Data set FOO.FILE1 contained no structural errors, however some changes during last
      update may not have been written to disk.
```

Or if your DLDMGACTION option is set to FAIL, perhaps you have run PROC DATASETS with the REPAIR option to repair a data set. If a data set continues to become damaged over and over, corrective action at the operating-system level might be necessary. For example, for disk-full conditions, perhaps the data set needs to be written to a physical location that is large enough to contain it.

Taking note of when the value of “Number of Data Set Repairs” is greater than zero is a best-practice recommendation. To perform that check, you can use the ODS OUTPUT functionality. Below is a short program that outputs an observation when a data set in the library has been repaired at least once. If the ‘repaired’ data set is created, you know that there is at least one data set that fits the criteria and you might need to investigate. Using the _ALL_ construct with PROC CONTENTS checks all data sets in the library, such as MYLIB in this example.

```
ods output enginehost=repaired(
   drop=cvalue1
   where=(label1 contains 'Repairs' and nvalue1> 0)
);

proc contents data= mylib._all_ ;run;
proc print data=repaired label;
   var member nvalue1;
   label nvalue1='# of Repairs';
   format nvalue1 1.;
run;
```

```
                            # of
     Obs       Member       Repairs

      1     MYLIB.DATA1       1
      2     MYLIB.FILE1       2
```

Output 4. Identifying data sets that have been repaired in a library

EXTENDOBSCOUNTER

The number of observations is kept in a long integer in a SAS data set. In an operating environment with a 32-bit long integer, such as 64-bit Windows, a count of up to 2G-1 observations can be stored in the data set header. The main concern with exceeding the observation count limit is that you can no longer have indexes or integrity constraints on that data set. If the data set is indexed or has an integrity constraint, you would see this warning in the log when the observation count limit is exceeded:

WARNING: File TEMPDATA.BIGONE contains 2G - 1 observations and cannot hold more because it contains an index or an Integrity Constraint that uses an index.

To continue using the index or integrity constraint, you must extend the observation count limit. To do this, you re-create the data set with the EXTENDOBSCOUNTER=YES option. This produces a file format that lets the observation count limit match that of a 64-bit long integer. This was available in SAS 9.3, but the default value for the system option was NO.

In SAS 9.4, the default value of the EXTENDOBSCOUNTER system option became YES. By default, any data sets created have the enhanced file format. By default, data sets created in an operating environment with a 32-bit long integer with SAS 9.4 cannot be read by releases before SAS 9.3. If your site requires access to these new data sets from sessions prior to SAS 9.3, you need to set the system option EXTENDOBSCOUNTER=NO. If you deliver data sets to other locations, you might also need to set the system option to NO.

For a more detailed description, see “Extending the Observation Count in a SAS Data File” on support.sas.com.

SCALABLE PERFORMANCE DATA ENGINE (SPDE)

For SPDE, PROC CONTENTS surfaces details about how the parts that comprise the data set are stored, including metadata and data file names and the data partsize. It also lists index information if the data set is indexed and compression information if the data set is compressed. Because the data set is usually kept on disk in many different files for concurrent access, you might also need to list physical files that make up the logical data set. You can do this using the LISTFILES= data set option. LISTFILES= is a special data set option that is used only with PROC CONTENTS and an SPDE data set. The default setting is NO. Here is the example.

```
libname cens01 spde 'c:\main' datapath=('c:\data');
proc contents data=cens01.educ12(listfiles=yes);run;
```

Output 5 shows PROC CONTENTS SPDE information with LISTFILES=YES.

```
                    Engine/Host Dependent Information

             Blocking Factor (obs/block)                819
             Data Partsize                              33553920
             -   Alphabetic List of Index Info          -
             Index                                      AGE
             KeyValue (Min)                             0
             KeyValue (Max)                             100
             Number of discrete values                  101
             Index                                      STATE
             KeyValue (Min)                             AK
             KeyValue (Max)                             WY
             Number of discrete values                  51
             -   Metadata Files                         -
             c:\main\educ12.mdf.0.0.0.spds9             -
             -   Data Files                             -
             c:\data\educ12.dpf.03bf03f6.0.3.spds9      -
             c:\data\educ12.dpf.03bf03f6.1.3.spds9      -
             c:\data\educ12.dpf.03bf03f6.2.3.spds9      -
             -   Index Files                            -
             c:\main\educ12.idxstate.03bf03f6.0.3.spds9  -
             c:\main\educ12.hbxstate.03bf03f6.0.3.spds9  -
             c:\main\educ12.idxage.03bf03f6.0.3.spds9    -
             c:\main\educ12.hbxage.03bf03f6.0.3.spds9    -
```

Output 5. Output from a PROC CONTENTS for SPDE with the LISTFILES option.

UNIX HOSTS

UNIX hosts surface such information as which users have operating-system access permissions to the data set, the size of the file in bytes, and more. You can see the values in Output 6.

```
    Inode Number               5275560
    Access Permission          rw-r--r--
    Owner Name                 sasddo
    File Size (bytes)          131072
```

Output 6. Engine/Host PROC CONTENTS for UNIX Hosts

One use of this information would be to retain access permissions on copied data sets. For example, when you use PROC COPY or PROC MIGRATE, new library members contain the default access permission of whoever is running those procedures. If you want to retain the original permissions, then the access permission of the original data set can be obtained using the EngineHost ODS object table name and the ODS OUTPUT statement before the PROC CONTENTS call.

Z/OS HOST

Three different types of data sets exist on z/OS. As a result, you can see different EngineHost contents.

z/OS DIRECT ACCESS BOUND

```
Data Set Page Size          36864
Number of Data Set Pages    101
First Data Page             1
Max Obs per Page            83
Obs in First Data Page      77
Number of Data Set Repairs  0
Physical Name               MYUSERID.PTE.D100922.SASLIB
Release Created             9.0301B0
Release Last Modified       9.0301B0
Created by                  MYSASJOB
Last Modified by            MYSASJOB
Subextents                  1
Total Blocks Used           606
Percent of Lib Allocation   31.5%
```

Output 7. Engine/Host PROC CONTENTS for z/OS Direct-Access Bound Library member

On z/OS, SAS data libraries can reside in what we describe as a direct-access bound library, as is shown in this example. The term *bound* is used to indicate that all members are bound together into a single native MVS data set. A direct-access bound library is conceptually very similar to an MVS PDSE, but the format is proprietary to SAS.

Field Descriptions

Data set page size: The number of bytes occupied by a single page of the member. For direct-access bound libraries, the page size is always an integral multiple of the library block size.

Physical name: The physical name of the native MVS data set in which the library resides. No unique physical name exists for each member because all members reside in the SAS-bound library, which in turn resides in a single MVS data set.

Created by: The name of the MVS job or TSO session under which the member was created.

Last modified by: The name of the MVS job or TSO session under which the member was last modified.

Subextents: The number of distinct ranges of contiguous blocks required to store the member. It indicates the degree to which the member is fragmented within the library space map.

Total blocks used: The total number of blocks occupied by the member. It equals the number of member pages multiplied by the number of library blocks required to store a single page.

Percent of lib allocation: The percentage of the total library space that is occupied by this member. It is computed by dividing *Total blocks used* (see above) by *Total library blocks* (found in the Directory statistics).

z/OS SEQUENTIAL BOUND

```
Data Set Page Size  32760
Physical Name       MYUSERID.SASLIB.V9TAPE
Release Created     9.0401M1
Created by          MYSASJOB
```

Output 8. Engine/Host PROC CONTENTS for z/OS Sequential-Bound Library member

On z/OS, SAS data libraries can reside in a sequential-access bound library, as is shown in this example. Multiple members can exist in these libraries, and the library exists in a single MVS data set which may reside on tape or on disk.

Field Descriptions

Physical name: The physical name of the native MVS data set in which the library resides. No unique physical name exists for each member because all members reside in the SAS bound library, which in turn resides in a single MVS data set.

Created by: The name of the MVS job or TSO session under which the member was created.

Due to the limitations of tape devices, members of a sequential library cannot be modified in place. Therefore, there are no modification statistics.

z/OS UNIX FILE SYSTEM

```
Data Set Page Size          16384
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            2013
Obs in First Data Page      1
Number of Data Set Repairs  0
ExtendObsCounter            YES
Physical Name               /u/myuserid/mylib/memb01.sas7bdat
Release Created             9.0401M0
Release Last Modified       9.0401M0
Created by                  MYUSERID
Last Modified by            MYUSERID
```

Output 9. Engine/Host PROC CONTENTS for z/OS UNIX file system member

On z/OS, SAS data libraries can reside in a UNIX file system directory. Each member of the library is stored in a separate UNIX file within the directory.

Field Descriptions

Physical name: The fully-qualified path or name for the UNIX file in which the member resides.

Created by: The name of the MVS job or TSO session under which the member was created. This information is available only for SAS files that were created by SAS on z/OS. It is omitted for files that SAS created on other host platforms.

Last modified by: The name of the MVS job or TSO session under which the member was last modified. This information is only available for SAS files that were created by SAS on z/OS. It is omitted for files created by SAS on other host platforms.

ODS OBJECT TABLE NAMES

PROC CONTENTS has a list of ODS object table names that indicates what name you must use in an ODS OUTPUT statement to generate a data set for a particular set of information.

Attributes	General data set attributes
Directory	General library information
EngineHost	Engine and operating system information
ExtendedAttributesDataSet	List of data set extended attributes
ExtendedAttributesDataSetShort	List of only data set extended attribute names
ExtendedAttributesVariable	List of variable extended attributes
ExtendedAttributesVariableShort	List of only variable extended attribute names
ICs	List of integrity constraints
ICsShort	List of only integrity constraint names
Indexes	List of indexes
IndexesShort	List of only index names
InfoMaps	List of information maps
InfoMapsShort	List of only information map filter names

Members	Library member information
Position	List variables by creation order in data set
PositionShort	List variable names only by creation order
Sortedby	Sort information
Sortedby Short	List of sorted variables
Variables	List of variables in alphabetical order
VariablesShort	List of only variable names in alphabetical order

CONCLUSION

Many data set attributes are associated with your actual data to form the data sets that you use on a daily basis. You can get more out of your SAS programming when you understand what attributes are available and how to use them to best advantage. In combination with an understanding of using the ODS OUTPUT statement, this knowledge can help you manage your data libraries and create more efficient SAS programs.

ACKNOWLEDGMENTS

I am a part of a great team at SAS. My thanks to Lewis King, who wrote the section on z/OS hosts. For their time, reviewing attention, and very intelligent suggestions, thanks to Jane Eslinger, Miguel Bamberger, Doug Ragsdale, Brad Richardson, and Bill McNeill.

CONTACT INFORMATION

I would love to answer any questions or comments. Please contact me by email: Diane.Olson@sas.com.