

What's New in Versions 7 and 8 for SAS Files?

Jane Stroupe, SAS Institute, Chicago, IL

ABSTRACT

The Nashville Release of the SAS® System provides several new features and enhancements to SAS files and their structure. As you begin to use the Nashville release, an understanding of compatibility between Version 6 SAS files and Nashville SAS files, along with the knowledge of what's new in the Nashville release, can assist in making the decision of when to make the change. This paper is an introduction to those features.

INTRODUCTION

Though there have been many enhancements to the SAS System in the Nashville Release, this paper concentrates on those base SAS enhancements to SAS files.

- New naming conventions
- Longer character values
- Library and catalog concatenation
- Generation Data sets
- Integrity Constraints
- Audit Trails
- Index enhancements
- Compression enhancements
- Creation of SAS tables using the Output Delivery System.

These features can motivate you to convert your SAS files to Nashville files; however, Version 6 SAS files and Nashville SAS files are very compatible so that the conversion does not have to be a massive, all-at-once conversion.

Unless otherwise noted, the features covered in the paper apply to both Version 7 and Version 8.

VERSION 6 AND NASHVILLE COMPATIBILITY

A Version 6 SAS session can not read, write, nor update any SAS file created as a Nashville SAS file.

On the other hand, a Nashville SAS session can access most Version 6 SAS files as shown in the following table.

Version 6 SAS File	Nashville Access
Data Files	Read/Write/Update
Data Step Views	None (V7)/ Read (V8)
SQL Views	Read
SAS/ACCESS® Views	Read/Write/Update
Catalogs	Read

Version 6 can still access any Version 6 SAS file that is read, created or updated by a Nashville SAS session.

OS/390 SAS FILES

A mainframe SAS data library can contain all Version 6 SAS files or all Nashville files, but not both. When you use a LIBNAME statement or a DD card, SAS determines whether the files were created by Version 6 or Nashville by checking the characteristics of the mainframe file.

DIRECTORY-BASED OPERATING SYSTEMS SAS FILES

In the directory-based systems, a directory can contain both Version 6 and Nashville files even if they have the same file name. The extensions in the directory-based systems are different for Nashville files.

Selected Nashville extensions	Type of file
.sas7bdat	Table
.sas7bview	Data view
.sas7bndx	Data set index
.sas7bcatalog	Catalog

Since both Version 6 and Nashville files can reside in the same directory, the library assignment is quite important.

You can use an engine assignment on the LIBNAME statement. For example,

```
libname sasdata v6 'pathname';
```

If you do not specify an engine, SAS assigns a default engine.

Directory Contains	Default Engine
All Version 6 SAS files	V6
All Nashville SAS files	V7
Both Nashville and Version 6 SAS files	V7
No SAS files	V7

When the directory contains all Version 6 files, by default SAS uses the Version 6 engine to read, write or update them even if you are using a Nashville SAS session. The SAS files remain Version 6; therefore, you can not use any Nashville enhancements (such as long variable names).

If the directory contains no SAS files or all Nashville SAS files, by default any file created in that directory by a Nashville SAS session is a Nashville SAS file. In this case, the LIBNAME statements

```
libname sasdata 'pathname';
```

or

```
libname sasdata v7 'pathname';
```

allow you to create Nashville SAS files or to read from Nashville SAS files. You can use the V7 engine or V8 engine assignment. Either results in the same structure of the file.

If you have both Version 6 and Nashville files in the same directory, you need to use an engine specification on the LIBNAME statement in order to access the Version 6 files with a Nashville SAS session.

```
libname sasdata v6 'pathname';
```

If you do not specify an engine, SAS defaults to the V7 engine. For example, if you submit this LIBNAME statement:

```
libname sasdata 'pathname';
```

you can read any Nashville SAS file, but not any Version 6 file.

NEW NAMING CONVENTIONS

WHAT'S NOT NEW?

The following names remain 8 characters maximum:

- Library references
- File references
- Format names

- Informat names.

WHAT'S NEW?

Most SAS names can now be up to 32 characters. The list of those includes:

- Variable names
- Data set names
- Macro variable names
- Macro names
- Catalog names
- Catalog entry names.

Variable names can be stored in mixed case, though they are normalized for look-ups.

For example, the variable CUSTOMER would be stored in mixed case as Customer.

```
data work.NewAccounts;
    length Customer $ 20;
    infile rawdata;
    input CUSTOMER $1 - 20;
run;
```

In the following example, the variable CUSTOMER would be compared correctly regardless of the case that it was stored in the data.

```
proc sql;
    select NewAccounts.customer, address,
           city, amountdue
    from work.NewAccounts,
         work.ListOfAccounts
    where NewAccounts.CUSomer=
           ListOfAccounts.CUSTOMER;
```

Data set names are not case sensitive. In most instances, you can now use *'paths specification/membername'* in place of *libref.membername*.

For example, instead of

```
libname sasdata 'c:\account\data';
data sasdata.NewCustomers;
    data step syntax
proc print data=sasdata.NewCustomers;
```

you could use

```
data 'c:\account\data\newcustomers';
    data step syntax
proc print
    data='c:\account\data\newcustomers';
```

By default, the only characters you can use in a name are letters, digits, and underscores. However, in Version 8 you can use a new system option, VALIDVARNAME=ANY and "varname"n specification to use other characters for variable names.

For example,

```
options validvarname=any;
data work.NewAccounts;
    set sasdata.AllAccountInformation;
    where "Account#"n = 'WEC1456';
run;
```

The values for the VALIDVARNAME= option are

- V7 (default) Allows mixed case, 32 character variable names. Valid characters you can use are alpha, numeric and underscores.
- UPCASE Allows 32 character variable names, but uppercases the variable names. Valid characters you can use are alpha, numeric and underscores.

- V6 Version 6 rules apply.
- ANY Allows variable names to contain special characters. You can use a maximum of 32 characters, stored in mixed case.

LONGER CHARACTER VALUES AND LABELS

Character data can be up to 32K in length. The default length used in situations like list input is still 8 bytes.

Labels for both variables and data sets are now 256.

LIBRARY AND CATALOG CONCATENATION

LIBRARY CONCATENATION

In all operating systems, you can concatenate libraries to assign one logical name to several libraries. This allows you to simplify your programming tasks.

You can list either paths or other *librefs* in the concatenation; however, to specify an engine, you must use a *libref*. For example,

```
libname sasdata ('c:\project1' 'c:\project2'
                'c:\project3');
or
libname sasdata1 'c:\project1';
libname sasdata2 oracle user=sam pw=sampw
                path=orapath;
libname data (sasdata1 sasdata2);
or
libname newdata (sasdata2, 'c:\project2');
```

When SAS reads a file from a library, it locates the first file with that file name in the concatenation. When SAS outputs a SAS file to a library, it writes the SAS file to the first library in the concatenation.

Library concatenation is useful for referring to both Version 6 and Nashville files with the same *libref* when you are converting Version 6 files to Nashville files.

For example,

```
libname data v6 'SAS-data-library';
libname sasdata v7 'SAS-data-library';
libname both (sasdata data);
```

The SAS data sets in both the version 6 library and the Nashville library can be referred to with the *libref* of BOTH. Furthermore, if BOTH.CUSTOMERS is a Version 6 data set, the syntax

```
data both.Newcustomers;
    set both.Customers;
    where year(date)=1999;
```

run;

would create a Nashville table named BOTH.NEWCUSTOMERS.

CATALOG CONCATENATION

You can logically combine two or more catalogs into one by using catalog concatenation. There are two types of catalog concatenation, implicit and explicit:

Implicit concatenation occurs when you concatenate two or more libraries which have like-named catalogs. The catalog members are automatically concatenated.

For example, if the library reference, SASDATA, is a concatenation of two libraries, each of which has a FORMATS catalog, there is one logical FORMATS catalog in SASDATA containing a set of unique entries from both the FORMATS catalogs involved in the concatenation.

If you have several catalogs in different libraries or catalogs in the same library but with different catalog names, you can use the CATNAME statement to create a logical catalog name for the concatenation. For example:

```
catname work.allprojects (sasdata.project1
                        sasdata.project2 data.project3);
```

The search order for a catalog concatenation is similar to that for library concatenation. When you create a catalog entry, the entry is written to the first catalog listed in the concatenation. When you read from a catalog, only the first entry with that name is retrieved.

GENERATION DATA SETS

By default, when you replace a SAS data set, the original data set is removed from the library. If you want to keep several versions of the data set, you can create the SAS data set with the capability of automatically storing historical versions. These historical versions along with the current version are generation data sets.

The GENMAX= data set option tells SAS how many copies of the data set to store.

For example,

```
data sasdata.employeeinfo(genmax=4);
    data step code
run;
```

allows SAS to keep the current version of SASDATA.EMPLOYEEINFO (also referred to as the base version) along with a maximum of 3 historical copies.

You can create a new version of a SAS data set using any replacement process such as a data step with a SET or MERGE statement or a PROC SORT step. As you create a new version, SAS assigns an absolute generation number to the latest historical version and reassigns relative generation numbers for all the historical versions. The relative generation number is a negative number that indicates the relationship between the current (base) version and the historical version.

For example, if you had created SASDATA.EMPLOYEEINFO five times, there would be four copies of the table in the SASDATA data library. The names and generation numbers of these tables would be as follows:

Table Name	Absolute Generation Number	Relative Generation Number
sasdata.employeeinfo	5	0
sasdata.employeeinfo#004	4	-1
sasdata.employeeinfo#003	3	-2
sasdata.employeeinfo#002	2	-3

At this point, the absolute generation number of 1 has been retired and the table named sasdata.employeeinfo#001 has been deleted.

You can refer to the historical versions of the data by using the GENNUM= data set option.

In the previous example, if you wanted to print the second version of the data, you could use the syntax:

```
proc print
    data=sasdata.employeeinfo(gennum=2);
run;
```

or

```
proc print
```

```
data=sasdata.employeeinfo(gennum=-3);
run;
```

INTEGRITY CONSTRAINTS

You can preserve the consistency and correctness of your data by creating integrity constraints on the table. The integrity constraints are rules that the data must adhere to when they are updated in place by editing in a window such as the VIEWTABLE window or by a submitting a step such as the DATA STEP with the MODIFY statement, PROC SQL with the INSERT INTO statement or SET statement, or PROC FSEDIT.

There are two types of integrity constraints, general and referential. The general constraints are

- NOT NULL constraint non-missing values
- UNIQUE constraint unique values
- CHECK constraint similar to a WHERE clause.

The referential constraints are

- PRIMARY KEY insures that the column(s) on which the primary key constraint is created are both unique and non-missing.
- FOREIGN KEY links tables together.

You can create integrity constraints with PROC SQL or with PROC DATASETS, both of which basically use the ANSI standard syntax. The advantage of using PROC DATASETS is that you can assign a message that appears during the editing process when a data value violates the constraint. A default message automatically appears regardless of techniques used to create the integrity constraint.

For example, if you wanted to create a NOT NULL constraint, a CHECK constraint, and a PRIMARY KEY constraint on the table SASDATA.CUSTOMERLIST, the syntax would be:

```
proc datasets lib=sasdata;
    modify CustomerList;
    integrity constraint create
        ❶ needaddress=not null(CustomerAddress) ❷
            message= 'You must specify an
                address';
    integrity constraint create
        ❸ ok_state=check(when=(state in
            ('GA','FL','SC','AL','MS')))
            message= 'State must be GA, FL,
                SC, AL, MS';
    integrity constraint create
        ❹ pkcustid=primary key(CustomerIdNumber)❺
            message='You need to provide a
                unique, non-missing
                customer ID number';
```

- ❶ NEEDADDRESS is the name of the constraint.
- ❷ CUSTOMERADDRESS is the column name.
- ❸ OK_STATE is the name of the constraint.
- ❹ PKCUSTID is the name of the constraint.
- ❺ CUSTOMERIDNUMBER is the column name.

If you add an invalid state, a missing customer address, or a non-unique or missing customer id number, the message appears.

For a detailed table SASDATA.ACCOUNTS, you could create a foreign key on the column ID. This foreign key links ID in SASDATA.ACCOUNTS to CUSTOMERIDNUMBER in SASDATA.CUSTOMERLIST. This parent/child relationship insures that the tables stay in sync when you add observations to the SASDATA.ACCOUNTS data, or delete rows from SASDATA.CUSTOMERLIST.

When you create the foreign key, you can specify what action to take if edits or deletions occur.

- **ON UPDATE RESTRICT**
prevents an ID value from being added to SASDATA.ACCOUNTS, if there is not a corresponding CUSTOMERIDNUMBER in the table SASDATA.CUSTOMERLIST.
- **ON DELETE RESTRICT**
requires that you delete all corresponding rows in SASDATA.ACCOUNTS before you delete an observation from SASDATA.CUSTOMERLIST.
- **ON DELETE NULL**
converts the corresponding ID values to missing in SASDATA.ACCOUNTS if you delete an observation from SASDATA.CUSTOMERLIST.

To create a foreign key, use

```
proc datasets lib=sasdata;
  modify Accounts;
  integrity constraint create
    ❶ fkidnum=foreign key(ID) ❷
      references sasdata.customerlist ❸
      on delete restrict ❹
      on update restrict ❺
      message= 'This ID number is not
        correct.';
```

- ❶ FKIDNUM is the name of the constraint.
- ❷ ID is the column name in SASDATA.ACCOUNTS that is the foreign key.
- ❸ SASDATA.CUSTOMERLIST is the name of the table containing a primary key. You do not need to specify the primary key since a table can have at most one primary key.
- ❹ Before you delete a row from SASDATA.CUSTOMERLIST, you must delete rows containing that ID in SASDATA.ACCOUNTS.
- ❺ You can not add a row to SASDATA.ACCOUNTS unless the corresponding ID can be found in SASDATA.CUSTOMERLIST.

AUDIT TRAILS

An Audit Trail is an optional SAS file that documents additions, deletions, and updates to a SAS data file. It provides a log of rejected records that can be useful especially if you are using integrity constraints.

Audit trails are experimental in Version 7 but production in Version 8.

In an audit trail file, there are three types of variables that store information about who made a modification to the SAS table, what was modified, and when the modification was made. These variable types are

- **Data file variables** that replicate the variables in the SAS table
- **Automatic variables** (called _AT*_ variables) that store modifications.
- **USER defined variables** (called USER_VAR variables) that let you provide additional information to the audit trail.

To create an audit trail file, use PROC DATASETS. The audit trail file is created in the same SAS data library as the SAS data file and has a member type of AUDIT.

For example,

```
proc datasets lib=sasdata;
  audit employeeinfo;
  initiate;
```

```
user_var reason $50 label='Why we made an
                        update.';
user_var whodidit $25 label='Who made the
                        update';
```

run;

Now add an observation to this SASDATA.EMPLOYEEINFO.

```
proc sql;
  insert into sasdata.employeeinfo
  set idnumber='1227',
      jobclassification='Manager',
      salary=150000,
      reason='New manager',
      whodidit='Me';
```

quit;

Only the values for IDNUMBER, JOBCLASSIFICATION, and SALARY are added to SASDATA.EMPLOYEE. All five variables are added to the audit trail.

To report the data in the audit trail file, use PROC PRINT with the TYPE= data set option.

```
proc print data=sasdata.employeeinfo
          (type=audit);
```

run;

produces the following report:

Obs	Job Classification	Id Number	Salary	reason
1	Manager	1227	\$150,000	New manager

Obs	whodidit	_ATDATETIME_	_ATOBSNO_	_ATRETURNCODE_
1	Me	23JUN99:15:31:19	4	.

Obs	_ATUSERID_	_ATOPCODE_	_ATMESSAGE_
1	sasjss	DA	

The _AT*_ variables are providing the following information:

ATDATETIME	Stores the date and time of a modification
ATUSERID	Stores the logon userid associated with a modification
ATOBSNO	Stores the observation number affected by the modification, except when REUSE=YES because the observation number is always 0
ATRETURNCODE	Stores the event return code
ATMESSAGE	Stores the SAS log message at the time of the modification
ATOPCODE	Stores a code describing the type of operation.

Consult the SAS OnLineDoc® for the various codes returned by _ATOPCODE_.

INDEXING ENHANCEMENTS

Improvements to indexing include:

- Not sorting the table when creating an index provided the table is already sorted
- Storing distribution information to use when determining whether to read the data using the index or sequential read
- Pattern matching with the LIKE operator
- Using composite indexes for compound optimization
- Waiting to update an index until all the values have been appended when appending data.

In addition there are two new data set options, IDXWHERE= and

IDXNAME=, that allow you to determine if an index is to be used or which index to use.

For example, the following program forces SAS to use an index to be used to read the data.

```
proc print data=sasdata.accountinformation
      (idxwhere = yes);
      where accountnumber='31234' or
            accountnumber='14356';
run;
```

COMPRESSION ENHANCEMENTS

You can now specify a compression algorithm that best suits your data.

For example, use the following code if the table has a lot of numeric data.

```
data sasdata.compress1 (compress=binary);
      data step syntax
run;
```

Use the following code if the table has a lot of character data. (The COMPRESS=YES option gives the same result.)

```
data sasdata.compress2 (compress=char);
      data step syntax
run;
```

In addition, you can use the POINT= option with compressed data to create a random sample.

CREATION OF A SAS TABLE USING ODS

In the Nashville Release, procedure output is created as output objects. These output objects separate the data generated from the template used to create the report. The Output Delivery System (ODS) allows you to get data containing every statistic created by almost all procedures.

Before you create a data set using ODS, you must know the names of the data components created.

Use the ODS TRACE statement to determine the names of the output objects generated by a procedure.

For example,

```
ODS TRACE ON;
proc univariate data=sasdata.employeeinfo;
      var salary;
run;
proc freq data=sasdata.employeeinfo;
      tables jobclassification*agegroup;
run;
ODS TRACE OFF;
```

This provides this information in your log:

For PROC UNIVARIATE:

```
Output Added:
-----
Name:      Moments
Label:     Moments
Template:  base.univariate.Moments
Path:      Univariate.Salary.Moments
-----
```

```
Output Added:
-----
Name:      BasicMeasures
Label:     Basic Measures of Location and
Variability
Template:  base.univariate.Measures
```

```
Path:      Univariate.Salary.BasicMeasures
-----
Output Added:
-----
Name:      TestsForLocation
Label:     Tests For Location
Template:  base.univariate.Location
Path:      Univariate.Salary.TestsForLocation
-----
```

```
Output Added:
-----
Name:      Quantiles
Label:     Quantiles
Template:  base.univariate.Quantiles
Path:      Univariate.Salary.Quantiles
-----
```

```
Output Added:
-----
Name:      ExtremeObs
Label:     Extreme Observations
Template:  base.univariate.ExtObs
Path:      Univariate.Salary.ExtremeObs
-----
```

For PROC FREQ

```
Output Added:
-----
Name:      CrossTabFreqs
Label:     Cross-Tabular Freq Table
Data Name:
Path:
Freq.JobClassification_by_agegroup.CrossTabFreqs
-----
```

Note that the ODS TRACE ON statement turns on the trace until you use the ODS TRACE OFF statement.

You can create tables for some or all of these output objects.

For example, to create tables for all of the output objects, you use:

```
ODS OUTPUT Moments=work.salarymoments
            BasicMeasures=work.salbmeasures
            TestsForLocation=work.salarytloc
            Quantiles=work.salaryquantiles
            ExtremeObs=work.ExtremeObs;
proc univariate data=sasdata.employeeinfo;
      var salary;
run;

ODS OUTPUT CrossTabFreqs=work.freqcounts;
proc freq data=sasdata.employeeinfo;
      tables jobclassification,agegroup;
run;
```

The scope of the ODS OUTPUT statement is for the procedure submitted. Therefore, you do not have to turn it off when the procedure is completed as you do with ODS TRACE.

If you have several analysis variables or if you have BY variables, ODS creates several objects with the same name. By default, the ODS OUTPUT statement creates only one SAS table by default.

For example,

```
ODS OUTPUT Quantiles=work.quantiles;
proc univariate data=sasdata.employeeinfo;
      var salary bonusamount;
run;
```

creates one table containing all the quantile information for each

analysis variable.

All the quantiles data			
Obs	VarName	Quantile	Estimate
1	salary	100% Max	110241.0
2	salary	99%	108667.0
3	salary	95%	87306.0
4	salary	90%	71890.0
5	salary	75% Q3	46380.5
6	salary	50% Median	39873.5
7	salary	25% Q1	34958.0
8	salary	10%	31280.0
9	salary	5%	30041.0
10	salary	1%	26151.0
11	salary	0% Min	26149.0
12	bonusamount	100% Max	5512.1
13	bonusamount	99%	5433.4
14	bonusamount	95%	4365.3
15	bonusamount	90%	3594.5
16	bonusamount	75% Q3	2319.0
17	bonusamount	50% Median	1993.7
18	bonusamount	25% Q1	1747.9
19	bonusamount	10%	1564.0
20	bonusamount	5%	1502.1
21	bonusamount	1%	1307.6
22	bonusamount	0% Min	1307.5

You can use an ODS option, MATCH_ALL, to create an individual table containing information for just one analysis variable.

For example,

```
ODS OUTPUT Quantiles(MATCH_ALL)
      =work.quantiles;
proc univariate data=sasdata.employeeinfo;
  var salary bonusamount;
run;
```

creates a table named WORK.QUANTILES containing quantile information for the variable SALARY, and a table named WORK.QUANTILES1 containing quantile information for the variable BONUSAMOUNT.

Quantiles data			
Obs	Var Name	Quantile	Estimate
1	salary	100% Max	110241.0
2	salary	99%	108667.0
3	salary	95%	87306.0
4	salary	90%	71890.0
5	salary	75% Q3	46380.5
6	salary	50% Median	39873.5
7	salary	25% Q1	34958.0
8	salary	10%	31280.0
9	salary	5%	30041.0
10	salary	1%	26151.0
11	salary	0% Min	26149.0

Quantiles1 data			
Obs	VarName	Quantile	Estimate
1	bonusamount	100% Max	5512.05
2	bonusamount	99%	5433.35
3	bonusamount	95%	4365.30
4	bonusamount	90%	3594.50
5	bonusamount	75% Q3	2319.03
6	bonusamount	50% Median	1993.68
7	bonusamount	25% Q1	1747.90
8	bonusamount	10%	1564.00
9	bonusamount	5%	1502.05
10	bonusamount	1%	1307.55
11	bonusamount	0% Min	1307.45

CONCLUSION

As you convert your SAS files from Version 6 to Nashville, keep in mind that the conversion can take place in stages. Since a Nashville SAS session can read and write most Version 6 SAS files, you do not need to convert your Version 6 SAS files until

you want to take advantage of the new features for SAS files available in the Nashville release.

As you make the decision of when to convert from Version 6 to the Nashville release, note the many Nashville enhancements to the SAS System that benefit the way you store and retrieve your SAS files. Longer names and longer character values, the capability to store multiple copies of a SAS data set automatically (generation data sets), the ability to determine how edits to your data are evaluated (integrity constraints and audit trails) simplify your data storage tasks. Using ODS to create tables containing some or all the statistics generated from a procedure makes it easy to report on specific information contained in a table.

REFERENCES

SAS OnLineDoc®

Course notes for New Features of the Nashville Release of the SAS System.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Jane Stroupe

SAS Institute

Chicago IL

312/819-6800 ext 8823

Email: Jane.Stroupe@sas.com