THE POWER TO KNOW.

SAS® 9.3 LIBNAME Engine for DataFlux® Federation Server

User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2012. *SAS® 9.3 LIBNAME Engine for DataFlux® Federation Server: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.3 LIBNAME Engine for DataFlux® Federation Server: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, August 2012

Contents

Accessibility Features of the LIBNAME Engine for DataFlux Federation Server

Overview

The LIBNAME engine for DataFlux Federation Server has a command-line-only interface that is accessible using a keyboard or alternative keyboard assistive technologies. For this release, no accessibility testing was done and no additional features were added to address accessibility. If you have specific questions about the accessibility of SAS products, send them to **`accessibility@sas.com`** or call SAS Technical Support.

Recommended Reading

- *DataFlux Federation Server: Administrator's Guide*
- *SAS Language Reference: Concepts*
- *Base SAS Procedures Guide*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

Part 1

Introduction

Chapter 1
Using This Document

About the Document

This document explains how to use the SAS LIBNAME engine for DataFlux Federation Server, which has the engine name FEDSVR.

The LIBNAME engine is new functionality in the second maintenance release of SAS 9.3 software.

- Part 1 covers basic information about this document and introduces the FEDSVR engine.
- Part 2 describes how to use the FEDSVR engine.
- Part 3 describes the LIBNAME statement syntax and provides reference information about system, LIBNAME, and data set options that are supported by the FEDSVR engine. Not all options are available for all data sources.
- The appendix contains information about how to submit a fully specified data source connection string to the DataFlux Federation Server, instead of a DSN.

Intended Audience

This document is intended for applications programmers who

- are familiar with the basics of their data sources
- know how to use their operating environment
- can use basic SAS commands and statements
- have access to a DataFlux Federation Server

Database administrators might also want to read this document to understand how the FEDSVR engine is implemented and administered.

See Also

Throughout this document, you are referred to the following documentation:

DataFlux Federation Server Administrator's Guide
provides an overview of the functionality and components of the DataFlux Federation Server and itemizes the requirements for installing the server and configuring its drivers, data services, users, and DSNs. A data source reference within the guide describes requirements for connecting to each supported data source, and the database functionality that is supported through that data source driver.

SAS Language Reference: Concepts
Documents essential concepts for the DATA step, SAS features, and SAS files.

Chapter 2

Introduction to the LIBNAME Engine for DataFlux Federation Server

Understand the LIBNAME Engine

Overview of the FEDSVR Engine

The LIBNAME engine for DataFlux Federation Server provides a bridge from legacy SAS data access services to the data access technology that is provided by the DataFlux Federation Server. From a Base SAS session, the LIBNAME engine enables a SAS application such as a SAS procedure or a SAS DATA step to process data using the DataFlux Federation Server data access technology. For example, using the LIBNAME engine, you can process data sources such as a SAS data set, and the third-party databases that are supported by the DataFlux Federation Server.

Overview of the DataFlux Federation Server

The DataFlux Federation Server is a data server that provides scalable, threaded, multi-user, standards-based data access technology in order to process and seamlessly integrate data from multiple data sources. The server acts as a hub that provides clients with data by accessing, managing, and sharing SAS data as well as data from several third-party databases.

The DataFlux Federation Server provides access to several types of data. The initial release of the LIBNAME engine for DataFlux Federation Server supports SAS data sets, and third-party relational databases, including DB2, Greenplum, MySQL, Oracle, Teradata, and ODBC databases (such as dBASE, Microsoft Access, and Microsoft SQL Server).

Structured Query Language (SQL) is the data access language for the server. Clients typically connect to the DataFlux Federation Server and submit requests in the form of DataFlux FedSQL statements. The DataFlux FedSQL language is the implementation of SQL used by the DataFlux Federation Server to access relational databases. FedSQL provides a subset of the ANSI SQL standard SQL:1999 core compliant syntax as well as extra extensions. For applications, FedSQL provides a common SQL syntax across all data sources.

For complete information about the DataFlux Federation Server and its data access services, see the *DataFlux Federation Server Administrator's Guide*.

Using the LIBNAME Engine for DataFlux Federation Server

To use the LIBNAME engine, you must submit a LIBNAME statement that associates a SAS libref with the data that will be processed by the DataFlux Federation Server. The name of the LIBNAME engine for DataFlux Federation Server is FEDSVR. The remainder of the LIBNAME statement specifies connection information in order to connect to the Federation Server and to the data source. For more information, see Chapter 3, "Establishing a Connection to the DataFlux Federation Server," on page 9. Once a libref is assigned, you use that libref in the SAS session wherever a libref is valid.

Part 2

Usage

Chapter 3

Establishing a Connection to the DataFlux Federation Server

Components of the LIBNAME Statement

The LIBNAME statement for the LIBNAME engine for DataFlux Federation Server has the following components:

- FEDSVR engine name
- server connection arguments
- data source connection arguments
- data source processing options.

Server Connection

Server Connection Arguments

To connect to a data source with the DataFlux Federation Server, you must first connect to the DataFlux Federation Server. To connect to the DataFlux Federation Server, you must specify server connection arguments in the LIBNAME statement.

The DataFlux Federation Server process can be running on the local computer or on a remote computer. The SAS program connects as a client of the Federation Server. The data source connection arguments identify the computer on which the server is running, the port number which is used to access the server, and user authentication credentials.

The following is an example of the code necessary to connect to a DataFlux Federation Server and access a SAS data set.

```
libname lib1 fedsvr server="d1234.us.company.com"
   port=2171 user="myid" password=mypwd
   dsn="BaseDSN";

proc print data=lib3.table1;
run;
```

In the example, FEDSVR is the name of the LIBNAME engine and SERVER=, PORT=, USER= and PASSWORD= are server connection arguments.

SERVER="d1234.us.company.com"
identifies the computer on which the DataFlux Federation Server is running.

PORT=2171
specifies the TCP port that the DataFlux Federation Server is listening to for connections. 2171 is the default port number defined for a DataFlux Federation Server at installation.

USER="myid"
specifies the user ID for logging on to the server. Alias: UID=.

PASSWORD=mypwd
specifies the password that corresponds to the user ID for the server. Alias: PWD=.

DSN= is a data source connection argument. For more information, see "Data Source Connection" on page 11.

Authentication Requirements

Access to the DataFlux Federation Server and its data services is controlled by the DataFlux Authentication Server. The users and groups who can access the server and its data services are defined and maintained on the Authentication Server by the DataFlux Federation Server administrator. To obtain a valid login for the DataFlux Federation Server, contact the DataFlux Federation Server administrator.

AUTHDOMAIN= Option

If the FEDSVR engine is invoked in a metadata-aware environment, then an AUTHDOMAIN= option can be used to supply user credentials instead of the USER= and PASSWORD= arguments. A metadata-aware environment is a SAS session in which the METASERVER=, METAPORT=, METAUSER=, METAPASS= and METAREPOSITORY= system options have been set and have successfully established a connection to a SAS Metadata Server. The AUTHDOMAIN= argument specifies the name of an authentication domain metadata object that is defined in a SAS Metadata Repository.

Specifying an authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign on. Metadata server connection system options can be specified in a configuration file as well as in the OPTIONS statement.

The AUTHDOMAIN= option does not bypass authentication by the DataFlux Authentication Server; it also routes authentication through the SAS Metadata Server. For more information about authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*. For an example of how AUTHDOMAIN= is used, see "Authenticate with an Authentication Domain" on page 35.

Data Source Connection

The LIBNAME engine to the DataFlux Federation Server accesses data by referencing a DataFlux data source name (DSN) definition.

What is a DSN?

A DSN is an object that encapsulates data source connection information, such as the table driver name, physical location of the data, and any necessary authentication information that is required to retrieve data. A DSN is created in the DataFlux Federation Server Manager by a DataFlux Federation Server administrator. To obtain the name of a DSN definition, contact a DataFlux Federation Server administrator.

By requiring users to reference a DSN, the DataFlux Federation Server administrator can set authorization enforcement on data access and make the DSN available only to authorized users.

A DSN is referenced by specifying the DSN definition name in the **`DSN=`** argument of the LIBNAME statement.

DSN=*dsn-definition*
specifies the name of DSN definition that is defined on the DataFlux Federation Server.

Note: If the DSN definition name is not a valid SAS name, enclose the name in quotation marks. If the name contains single quotation marks, enclose the name in double quotation marks, or use two single quotation marks in the name and enclose the name in single quotation marks.

DSNUSER=*userid*
specifies a user ID or shared login that is defined for the DSN definition. SAS data sets do not support a user ID or password for a DSN definition. Alias: DSNUID=.

DSNPASSWORD=*password*
specifies the password that is associated with the DSNUSER= user ID. Alias: DSNPWD=.

Within the LIBNAME statement, the **`DSN=`** arguments are specified after the server connection arguments and before data source processing options, as follows:

```
libname lib3 fedsvr server="d1234.us.company.com"
   port=2171 user="myid" pwd=mypwd
   dsn="oradsn" preserve_col_names="yes" dbgen_name="dbms";
```

Federated DSNs

The DataFlux Federation Server supports DSN definitions that reference multiple data sources. The data sources can be on the same or on different DBMS. A DSN definition that references multiple data sources is referred to as a “federated DSN”.

The FEDSVR LIBNAME engine supports accessing one data source at a time. When you use a DSN that specifies multiple data sources, the engine references the first data source in the DSN by default. To access data from a subsequent data source in the DSN, you must reference the data source by specifying the catalog or schema name in the QUALIFIER or SCHEMA= option. QUALIFIER= and SCHEMA= can be specified as data source processing options, or as data set options. It is not necessary to specify both options unless they are required to uniquely identify the data source. Using these options, you will be able to read and write data from each of the data sources in the DSN. For an example of how the options are specified, see "Using the SCHEMA= Data Set Option to Reference a Subsequent Data Source in a Federated DSN" on page 36.

Data Source Processing Options

A SAS LIBNAME statement supports options to define how SAS processes DBMS objects. Some LIBNAME options enhance data access performance. Others determine locking and naming behavior. Options are also available to pass data source commands, SAS functions and to specify joins.

The LIBNAME statement options that you can submit depend on the data source. The supported statement options are listed in Chapter 12, "Data Source Processing Options for the FEDSVR Engine," on page 65.

Chapter 4
Security

About the Security on the LIBNAME Engine

When you use the LIBNAME engine for DataFlux Federation Server, you can connect only to data sources and locations as defined through the server administration. A non-administrative user must have Connect privilege on the DataFlux DSN or the data service in order to connect to the DSN.

A DSN definition configures the authorization that is enforced for data access. The available authorization process depends on the data source. Third-party databases enforce data source security and can also be configured for Federation Server authorization. A DSN for SAS data sets can be configured to use legacy SAS file passwords or DataFlux Federation Server authorization. A DSN for SAS data sets cannot use both SAS passwords and Federation Server authorization.

SAS File Passwords

When you use the LIBNAME engine for DataFlux Federation Server, SAS passwords are supported only for unsecured DSNs. An unsecured DSN is a DSN that does not have Federation Server authorization enabled. If you specify a SAS password to create or read a SAS data set when Federation Server authorization is enabled, the server will return an error.

Base SAS software enables you to restrict access to SAS data sets by assigning SAS passwords when you create the data sets. You can specify three levels of protection: read, write, and alter. Later, to read or update the data set, you specify the appropriate password.

To assign or specify a password, submit the ALTER=, PW=, READ=, and WRITE= password options as follows:

```
libname myfiles fedsvr server="d1234.us.company.com"
   port=2171 uid="myid" pwd="mypwd"
   dsn=basedsn;
```

```
data myfiles.newds (pw=luke);
   set sashelp.class;
run;
```

In the preceding example, the DATA step assigns the password Luke to the new SAS data set.

For more information about SAS passwords, see *SAS Language Reference: Concepts*.

DataFlux Federation Server Security

DataFlux Federation Server supports authentication, encryption, data source authorization, and Federation Server authorization. Federation Server authorization is a security scheme in which the DataFlux Federation Server defines and enforces permissions that particular users have for particular resources. Authorization either permits or denies a specific action on a specific resource, based on the user's identity and on group memberships. For more information about the security services provided through DataFlux Federation Server, see the *DataFlux Federation Server Administrator's Guide*.

Chapter 5

Data Type Support

About the DataFlux Federation Server Data Type Support

The LIBNAME engine supports the DataFlux Federation Server data types by translating their definition to and from predetermined legacy SAS data types, which are SAS numeric and SAS character. For example, when you use the LIBNAME engine, the CONTENTS procedure reports the DataFlux Federation Server DATE data type as a SAS numeric.

A SAS numeric is a DOUBLE data type, which stores a 64-bit double-precision, floating point number. A SAS character is a CHAR data type, which stores a fixed-length character string from 1 to 32,767 bytes.

For complete information about the DataFlux Federation Server data types and the data types that are available for data storage in each data source, see the *DataFlux Federation Server Administrator's Guide*.

Translation of the DataFlux Federation Server Data Types

The DataFlux Federation Server data types are translated to and from the following predetermined legacy SAS data types:

Table 5.1 *DataFlux Federation Server Data Type Translation*

DataFlux Federation Server Data Type	Legacy SAS Data Type	Description
BIGINT	SAS numeric	Applies the SAS format 20. Because a SAS numeric is a DOUBLE, which is an approximate numeric data type rather than an exact numeric data type, there is potential for loss of precision.
CHAR(*n*)	SAS character	Applies the SAS format $n.
DATE	SAS numeric	Applies the SAS format DATE9. Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside the SAS date range are not supported and are treated as invalid dates.
DOUBLE	SAS numeric	
FLOAT(*p*)	SAS numeric	
INTEGER	SAS numeric	Applies the SAS format 11.
NCHAR(*n*)	SAS character	Applies the SAS format $n.
NVARCHAR(*n*)	SAS character	Applies the SAS format $n.
REAL	SAS numeric	
SMALLINT	SAS numeric	Applies the SAS format 6.
TIME(*p*)	SAS numeric	Applies the SAS format TIME8. LIBNAME engine does not support fractions of seconds for time values.
TIMESTAMP(*p*)	SAS numeric	Applies the SAS format DATETIME19.2.
TINYINT	SAS numeric	Applies the SAS format 4.
VARBINARY(*n*)	SAS character	Applies the SAS format $n.
VARCHAR(*n*)	SAS character	Applies the SAS format $n.

Override Default Legacy SAS Data Type

To override the predetermined legacy SAS data type, you can specify the DBSASTYPE= option in order to specify the data type to which to convert. Note that some data types are not supported.

For example, in the following code, DBSASTYPE= specifies a data type to use for the column MYCOLUMN:

```
proc print data=mylib.mytable (dbsastype=(mycolumn=char(20)'));
run;
```

If a conversion is not supported, an error occurs. For information, see "DBSASTYPE= Data Set Option" on page 117.

Chapter 6

Null Values

About Null Values

A null value indicates the absence of information. A null value means that a real value is unknown or nonexistent. That is, no data is assigned to the column in that specific row. A null value is not a zero or a blank, and it does not behave like a zero or a blank.

A null value is represented by the DataFlux Federation Server either as a SAS missing value or an ANSI SQL null value:

SAS missing value
: The SAS missing value indicators (. , ._, .A-.Z, and ' ') are known values that indicate nonexistent data. A SAS missing value is interpreted as its internal floating-point representation. By default, SAS prints a missing numeric value as a single period (.) and a missing character value as a blank space. A SAS data set represents null values with SAS missing values.

ANSI SQL null value
: Table data with an ANSI null has no real data value; it is metadata that indicates an unknown value. Third-party relational databases represent null values with ANSI SQL null values.

Null Value Processing Modes

The processing behavior of a null value depends on the mode, which is determined by how you connect to the DataFlux Federation Server.

- A client application that connects to the DataFlux Federation Server with a client-side driver processes data by default by using ANSI SQL null value behavior.
- When you use the SAS LIBNAME engine for DataFlux Federation Server, null values are processed by default with SAS missing value behavior.

In some ways a SAS missing value is analogous to an ANSI SQL null value; however, the processing behavior can be different. Therefore, if an application is processing data with SAS missing value behavior rather than ANSI SQL null behavior, then you need to be aware of processing differences.

- You can sort a SAS missing value and evaluate it with standard comparison operators.
- You cannot sort an ANSI SQL null value or evaluate it with standard comparison operators, because there is no data on which to operate. For example, to test for a null value, you cannot use arithmetic comparison operators such as = or <.
- SAS missing values in a SAS data set are translated to ANSI SQL null values when the data is copied to a data source that processes in the ANSI SQL null mode (for example, an Oracle database).
- Many relational databases, such as Oracle and DB2, implement ANSI SQL null values. Therefore, the concept of ANSI SQL null values with the DataFlux Federation Server languages is the same as with the Oracle SQL language.

Note: Because the SAS data set does not physically store null indicators, the DataFlux Federation Server languages emulate ANSI SQL null values for the data source.

Potential Result Set Differences

When the data contains null values, you might get different result sets depending on whether the processing is done in SAS missing value mode or in ANSI SQL null value mode. Although in many cases this does not present a problem, it is important to understand how these differences occur.

Processing SAS missing values is different than processing ANSI SQL null values and has significant implications in these situations:

- when filtering data (for example, in a WHERE clause, a HAVING clause, or an outer join ON clause). SAS mode interprets null values as SAS missing values, which are known values, whereas ANSI mode interprets null values as unknown values.
- when submitting outer joins where internal processing might generate nulls for intermediate result sets.
- when comparing a blank character, SAS mode interprets the blank character as a missing value. In ANSI mode, a blank character is a blank character; it has no special meaning.

Change Null Processing Modes

When using the LIBNAME engine for DataFlux Federation Server, the default processing mode for null values is SAS missing value behavior. The following are instances of when you might want to change the mode to ANSI SQL null value behavior:

- an application processes a third-party database and wants the same results as on the DBMS, which processes data with ANSI SQL null value behavior.

- an application processes data and wants the same results as a client application that connects to the DataFlux Federation Server with a client-side driver, which processes data with ANSI SQL null value behavior

To change the null value processing behavior for the LIBNAME engine to have an ANSI SQL null value inserted into a character column instead of a blank, use the "NULLCHAR= Data Set Option" on page 132.

Chapter 7
SAS Names

About SAS Names

A SAS name is a name that is assigned to items such as columns and tables. For most SAS names, the first character must be a letter or an underscore. Subsequent characters can be letters, numbers, or underscores. Blanks and special characters (except the underscore) are not allowed. The maximum length of a SAS name depends on the language element to which it is assigned. Many SAS names, such as column names, can be 32 characters long. Other SAS names, such as librefs, have a maximum length of eight characters. For more information about SAS names, see *SAS Language Reference: Concepts*.

Because the DataFlux Federation Server languages include naming restrictions, such as reserved keywords, and some third-party databases allow case-sensitive names and names with special characters, you must show special consideration when you use the names for tables and columns. This section presents default naming behaviors and options that can modify naming behavior.

Support for Column Names with Special Characters

When the LIBNAME engine reads a column name that contains characters that are not allowed in standard SAS names, the default behavior is to replace an unsupported character with an underscore (_). For example, the database column name **`Amount Budgeted$`** becomes the SAS name **`Amount_Budgeted_`**.

Note: Nonstandard names include those with blank spaces or special characters (such as @, #, %) that are not allowed in SAS names unless the VALIDVARNAME=ANY option is set. When a SAS data set contains special characters in column names and the data set is accessed with VALIDVARNAME=ANY not set, the LIBNAME engine replaces the special characters with underscores, but no message is issued.

When SAS encounters a DBMS name that exceeds 32 characters, SAS truncates the name. After it has modified or truncated a DBMS column name, SAS appends a number to the name, if necessary, to preserve uniqueness. For example, DBMS column names **`MY$DEPT`**, **`My$Dept`**, and **`my$dept`** become SAS names **`MY_DEPT`**, **`MY_Dept0`**, and **`my_dept1`**.

To change how SAS handles case-sensitive or nonstandard table and column names, specify one or more of the following options:

PRESERVE_COL_NAMES=YES
: is a LIBNAME statement and data set option. When set to YES, this option preserves blank spaces, special characters, and mixed case in column names. See "PRESERVE_COL_NAMES= LIBNAME Statement Option" on page 83 for more information about this option.

PRESERVE_TAB_NAMES=YES
: is a LIBNAME statement option. When set to YES, this option preserves blank spaces, special characters, and mixed case in table names. See "PRESERVE_TAB_NAMES= LIBNAME Statement Option" on page 85 for more information about this option.

VALIDVARNAME=ANY
: is a SAS global system option that can override the SAS naming conventions. See "VALIDVARNAME= System Option" on page 62.

The availability of these options and their default settings are data source specific. See the naming conventions documentation for each data source reference in the *DataFlux Federation Server Administrator's Guide*.

Reserved Language Keywords

To use SQL language reserved keywords as table or column names, you must identify them in the program syntax. When using the LIBNAME engine, use the LIBNAME statement options PRESERVE_TAB_NAMES=YES or PRESERVE_COL_NAMES=YES in the LIBNAME statement.

SQL reserved keywords include, for example, ANALYZE, CREATE, MEMBER, SELECT, and WHERE.

To use SELECT as a table name when using the LIBNAME engine, specify the PRESERVE_TAB_NAMES=YES option.

```
libname mylib fedsvr server="d1234.us.company.com"
    port=2171 uid="myid" pwd="mypwd"
    dsn=oradsn preserve_tab_names=yes preserve_col_names=yes;

proc sql;
   create table mylib.select (row INT);
   insert into mylib.select values (100);
   select row from mylib.select;
quit;
```

Chapter 8

SAS Functionality Available Through the Engine

Data Source Access and Processing

The LIBNAME engine for DataFlux Federation Server supports the following processing:

- The engine supports input (read), output (create), and update (modify, add, and delete) processing. You can submit most SAS language procedures, including the SQL procedure, perform WHERE and BY-Group processing, and submit the DATA step with the DataFlux Federation Server data services to read, create, update, and delete data.
- You can list a data source's tables with the DATASETS procedure, and you can list the metadata attributes of a table with the CONTENTS procedure. However, the engine does not support utility processing to modify metadata attributes such as renaming or adding a column or changing a label. The MODIFY statement for PROC DATASETS is not supported because it requires the file to be opened in utility mode.

- The engine is a sequential engine with limited random access. If you request processing that requires random access that is not supported, a message in the SAS log notifies you that the processing is not valid for sequential access.
- The engine does not support threaded application processing. Therefore, SAS procedures such as the SORT and SUMMARY procedures cannot perform threaded processing.
- The engine does not support DATA step views and PROC SQL views.
- The engine cannot access a damaged file and therefore provides no means for repairing a damaged file.

Some SAS functionality is restricted by the data source itself. For example, when you use the engine to access a SAS data set, some SAS features such as audit trail, referential integrity constraints, generation data sets, and Cross-Environment Data Access (CEDA) are not supported. For data source functionality details, see the data source reference in *DataFlux Federation Server Administrator's Guide*.

LIBNAME Statement Processing Options

The LIBNAME statement provides several statement options that control data source processing. For example, you can process multiple rows by using the READBUFF= statement option.

However, the statement options that you can submit depend on the data source. The supported statement options are listed in Chapter 12, "Data Source Processing Options for the FEDSVR Engine," on page 65.

Nonexistent Values

For nonexistent values, the engine reads, writes, and updates SAS missing values for the data sources that implement SAS missing values, which is the SAS data set. For data sources that do not implement SAS missing values, such as the third-party databases, a nonexistent value is

- mapped to an ANSI SQL null value when writing to the file
- converted from an ANSI SQL null value to a SAS missing value when reading from the file.

Numeric Column Length

A column's length refers to the number of bytes used to store each of the column's values in a file. For example, a column that is defined as a DOUBLE has a storage size of 8 bytes.

In SAS, you can control the length with the LENGTH statement in the DATA step. However, the LIBNAME engine does not support specifying a shorter numeric storage size with the LENGTH statement. When you use the engine, you can read a legacy SAS file with a shorter numeric storage size, but you cannot specify a shorter numeric storage size or update the data.

SAS Data Set Options

You can apply SAS data set options on a table when you access a data source. A data set option applies only to the table on which it is specified, and it remains in effect for the duration of the DATA step or SAS procedure. For example, you can process a segment of data with SAS data set options such as FIRSTOBS= and OBS=.

However, the data set options that you can use depend on the data source. The supported data set options are listed in Chapter 13, “Data Set Options for FEDSVR Engine,” on page 97.

SAS Formats and Informats

You can submit DATA steps and PROC steps to do the following:

- store and apply SAS formats and informats for a SAS data set.

  Note: The engine does not support file replacement, which means that you cannot overwrite an existing file to change a stored SAS format. However, you can submit the FORMAT statement to display the data with a different format.

- apply SAS formats and informats for DB2, Greenplum, MySQL, ODBC databases, Oracle, and Teradata. You cannot store SAS formats or informats for those data sources.

When a table is created, the engine validates a specified SAS format or informat name. Both SAS supplied and user-defined formats and informats can be used.

SAS Indexes

The following SAS functionality for SAS indexes is not supported:

- You cannot create an index with a DATA step or a PROC step.
- The CONTENTS procedure does not display index information.
- For the SET and MODIFY statements, the KEY= option, which enables you to specify an index to retrieve particular observations, is not supported. The POINT= option, which enables you to specify a variable whose value is the observation number to be read, and UNIQUE= option, which causes a KEY= search always to begin at the top of the index for the data file, are also not supported for SET and MODIFY statements.
- The MODIFY statement in the DATASETS procedure is not supported, which means that for a SAS data set you cannot submit the INDEX CENTILES statement to request that centiles be refreshed or change how often centiles are refreshed.
- For a SAS data set, the fast-append method to append to an indexed data set is not supported.

SAS Names and Support for DBMS Names

When the engine reads DBMS column names that contain characters that are not standard in SAS names, the default behavior is to replace an unsupported character with an underscore (_). For example, the DBMS column name Amount Budgeted$ becomes the SAS name Amount_Budgeted_.

Note: Nonstandard names include those with blank spaces or special characters (such as @, #, %) that are not allowed in SAS names.

When the engine encounters a DBMS name that exceeds 32 characters, it truncates the name. After it has modified or truncated a DBMS column name, SAS appends a number to the variable name, if necessary, to preserve uniqueness. For example, DBMS column names MY$DEPT, My$Dept, and my$dept become SAS names MY_DEPT, MY_Dept0, and my_dept1.

SAS Passwords

When using an unsecured DSN, you can assign and specify SAS passwords by submitting the ALTER=, PW=, READ=, and WRITE= data set options.

You cannot submit the DATASETS procedure to manipulate SAS passwords on a SAS data set.

SAS System Options

SAS System options are instructions that affect your SAS session. They control how operations such as SAS System initialization, hardware and software interfacing, and the input, processing, and output of jobs and files, are performed. The availability and behavior of these options are data-source specific. The supported system options are listed in Chapter 11, “System Options for the FEDSVR Engine,” on page 45.

SAS Procedures

COMPARE Procedure

You can submit the COMPARE procedure to compare the contents of two tables, selected columns in different tables, or columns within the same table.

If you compare the contents of columns from different data sources, PROC COMPARE might report numeric precision differences due to a data source's implementation of a data type.

To compare columns from different data sources, you can specify equality criterion to control how values are compared. Two options for the PROC COMPARE statement determine the equality criterion for comparing values:

CRITERION=
: specifies the criterion for judging the equality of numeric values.

METHOD=
: specifies the method for judging the equality of numeric values.

For more information about these options, see the COMPARE procedure in *Base SAS Procedures Guide*.

CONTENTS Procedure

You can submit the CONTENTS procedure (or the CONTENTS statement in the DATASETS procedure) to list the attributes of a table (such as the date when the table was created, the date when data was last modified, and so on) or the contents of a directory.

However, there are some limitations for the CONTENTS procedure:

- Index information is not displayed.
- The number of observations (rows) are not displayed for tables that have row-level permission.
- The results of a PROC CONTENTS depend on the data source. For example, some of the attributes that list for a SAS data set do not list for a third-party database table.

SORT Procedure

When you use the SORT procedure, you will observe behavior differences:

- The engine does not support file replacement. That is, you cannot overwrite an existing table that has the same name. For example, you must specify the OUT= option for the SORT procedure:

  ```
  libname myfiles fedsvr server="d1234.us.company.com"
     port=2171 user="myid" pwd="mypwd"
     dsn=basedsn;

  proc sort data=myfiles.table1 out=myfiles.table1_sorted;
     by column1;
  run;
  ```

- There might be behavior differences based on the accessed data source. For example, you can sort the rows of a SAS data set and store the output to another SAS data set. However, for a third-party DBMS, sorting data often has no effect on how it is stored. When you access third-party DBMS tables, they are stored in the format of the database, which differs from the format of SAS files, such as a SAS data set.

  Because you cannot depend on your data to be sorted in the database, you must sort the data at the time of query. Furthermore, when you sort DBMS data, the results might vary depending on whether the DBMS places data with null values (which are translated to SAS missing values) at the beginning or the end of the result set.

  When the output table is written, the sorted data is written to the output table in the sorted order. However, when an application subsequently reads the output table from the DBMS, the DBMS sends the data back in an order unrelated to the sort.

A solution might be to specify a SAS file such as a SAS data set as the output table in the OUT= option as shown in the following code:

```
libname oradata fedsvr server="d1234.us.company.com"
   port=2171 user="myid" pwd="mypwd" dsn=oradsn;

proc sort data=oradata.table1 out=work.table1_sorted;
   by column1 column2;
run;

proc print data=work.table1_sorted;
run;
```

SQL Procedure

The engine supports the SQL procedure so that you can retrieve and manipulate data, create tables, and add or modify data.

You can also use the PROC SQL pass-through facility to connect to a data source, and submit DataFlux FedSQL statements. However, you cannot use the PROC SQL pass-through facility to send DBMS-specific SQL statements directly to a DBMS for execution.

Chapter 9

Examples of Using the Engine

Connect to the DataFlux Federation Server and Reference a DSN Definition

Details

This example accesses a SAS data set by specifying server connection arguments to connect to the DataFlux Federation Server and by referencing a DataFlux DSN with the DSN= data source connection argument. The DataFlux DSN provides the information to access the data source.

Program

```
libname mylib fedsvr 1
   server="d1234.us.company.com" port=2171 user="myid" pwd=mypwd 2
   dsn=BaseDSN; 3

proc print data=mylib.MyTable; 4
run;
```

Program Description

1. The LIBNAME statement assigns the libref MYLIB and specifies the FEDSVR engine.
2. The LIBNAME statement server connection arguments specify how to connect to the DataFlux Federation Server. The arguments identify the computer on which the server is running, the port number that is used to access the server, and user authentication information.
3. The DSN= connection argument references a DataFlux DSN, which is defined on the DataFlux Federation Server and provides the information to access the data source. The BaseDSN encapsulates the SAS data set connection information, such as the BASE table driver name and the physical location of the data.
4. The PRINT procedure specifies the name of the SAS data set that is accessed by the DataFlux DSN and prints the SAS data set.

Control Data Source Processing with a LIBNAME Statement Option

Details

The FEDSVR engine supports several LIBNAME statement options that control data source processing. The options that you can submit depend on the data source. This example shows how to preserve an Oracle table name that is not a valid SAS name.

Program

```
libname oralib fedsvr 1
   server="d1234.us.company.com" port=2171 user="myid" pwd=mypwd 2
dsn="OracleDSN" 3 preserve_tab_names="yes" 4;

proc sql dquote=ansi; 5
   select * from oralib."My Table";
```

Program Description

1. The LIBNAME statement assigns the libref ORALIB and specifies the FEDSVR engine.
2. The LIBNAME statement server connection arguments specify how to connect to the DataFlux Federation Server.
3. The data source connection arguments reference a DataFlux DSN, which provides the information to access an Oracle database.
4. The PRESERVE_TAB_NAMES=YES option controls table names by preserving blank spaces, special characters, reserved words, and mixed case. The default behavior is that SAS normalizes the DBMS table name according to SAS naming conventions.
5. The SQL procedure reads the table name from the data source as My Table.

Apply a SAS Data Set Option

Details

When you access a data source by using the FEDSVR engine, you can specify SAS data set options in a DATA step or SAS procedure. SAS data set options apply only to the table on which they are specified. The SAS data set options that you can use depend on the data source. Only a subset of the data set options that are provided by SAS are supported for the FEDSVR engine.

This example shows how to apply the DBCOMMIT= data set option. The DBCOMMIT= data set option causes an automatic commit (a permanent writing of data to the DBMS) after a specified number of rows have been processed.

Program

```
libname db2 fedsvr 1
   server="d1234.us.company.com" port=2171 user="myid" pwd=mypwd 2
   dsn=DB2DSN; 3

data db2.dept (dbcommit=10); 4
   ...
run;
```

Program Description

1. The LIBNAME statement assigns the libref DB2 and specifies the FEDSVR engine.
2. The LIBNAME statement server connection arguments specify how to connect to the DataFlux Federation Server.
3. The data source connection arguments reference a DataFlux DSN, which provides the information to access a DB2 database.
4. The DBCOMMIT= data set option in the DATA statement causes a commit to be issued after every 10 rows are processed.

Control Data Source Processing with a SAS System Option and a LIBNAME Statement Option

Details

SAS System options are instructions that affect your SAS session. They control how operations are performed. The availability and behavior of system options depend on the data source. Only a subset of the system options that are provided by SAS are supported for the FEDSVR engine. In addition, the FEDSVR engine supports several LIBNAME statement options that control data source processing. Like SAS system options, the LIBNAME statement options that you can submit depend on the data source.

This example shows how to control the rules for valid SAS column names by using a SAS system option along with a LIBNAME statement option.

Program

```
options validvarname=any; 1

libname mylib fedsvr 2
   server="d1234.us.company.com" port=2171 user="myid" pwd="mypwd" 3
   dsn=BaseDSN 4
   preserve_col_names=yes 5;

proc sql dquote=ansi;
   create table mylib.mytable ("my$column" int);
```

Program Description

1. The OPTIONS statement includes the VALIDVARNAME=ANY system option, which specifies that SAS column names can be up to 32 characters in length, can begin with or contain any characters, including blanks, and can contain mixed-case letters.
2. The LIBNAME statement assigns the libref MYLIB and specifies the FEDSVR engine.

3. The LIBNAME statement server connection arguments specify how to connect to the DataFlux Federation Server.
4. The data source connection arguments reference a DataFlux DSN, which provides the information to access a SAS data set.
5. The PRESERVE_COL_NAMES=YES option controls the column names by preserving blank spaces, special characters, and mixed case when a table is created. Because the VALIDVARNAME=ANY system option is specified, the PRESERVE_COL_NAMES=YES LIBNAME statement option is required.
6. The SQL procedure creates a table with a column named **`my$column`**.

Authenticate with an Authentication Domain

Details

The following is an example of how the AUTHDOMAIN= server connection option is used in the FEDSVR LIBNAME statement. The AUTHDOMAIN= server connection option enables a program to use metadata server connection system options to resolve the calling identity instead of specifying USER= and PASSWORD= values in the LIBNAME statement.

Program

```
options metaserver="d5678.us.company.com"
      metaport=8561
      metauser="myid"
      metapass="mypassword"
     metarepository=foundation; 1

libname lib1 fedsvr server="d1234.us.company.com" port="2171"
authdomain="fedauth1" 2 dsn=BASEDSN1;

proc datasets lib=lib1; run;
quit;
```

Program Description

1. The OPTIONS statement specifies SAS Metadata Server connection system options. The system options assign the session context to the user identified in METAUSER= and METAPASS= and establish a connection to the SAS Metadata Server on this user’s behalf. For more information about metadata server connection system options, see the *SAS Language Interfaces to Metadata*. These system options can be specified in a configuration file, as well as in the OPTIONS statement.
2. The AUTHDOMAIN= server connection argument specifies the name of an authentication domain that is defined on the SAS Metadata Server.

   If the specified authentication domain, FEDAUTH1, is a valid authentication domain for the user identified in METAUSER= and METAPASS=, and this user ID also authenticates to the DataFlux Authentication Server, then SAS permits the Federation Server connection.

The AUTHDOMAIN= server connection option does not bypass authentication by the DataFlux Authentication Server; it also routes authentication through the SAS Metadata Server. For more information about authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

Using the SCHEMA= Data Set Option to Reference a Subsequent Data Source in a Federated DSN

Details

The DataFlux Federation Server supports DSN definitions that reference multiple data sources. The data sources can be on the same DBMS or from different DBMS. A DSN definition that references multiple data sources is referred to as a “federated DSN”.

The FEDSVR LIBNAME engine supports accessing one data source at a time. This example shows how to use the SCHEMA= data set option to access data with a DSN that references two BASE SAS schemas.

Note: You must obtain appropriate schema names from the DataFlux Federation Server administrator in order to perform this procedure.

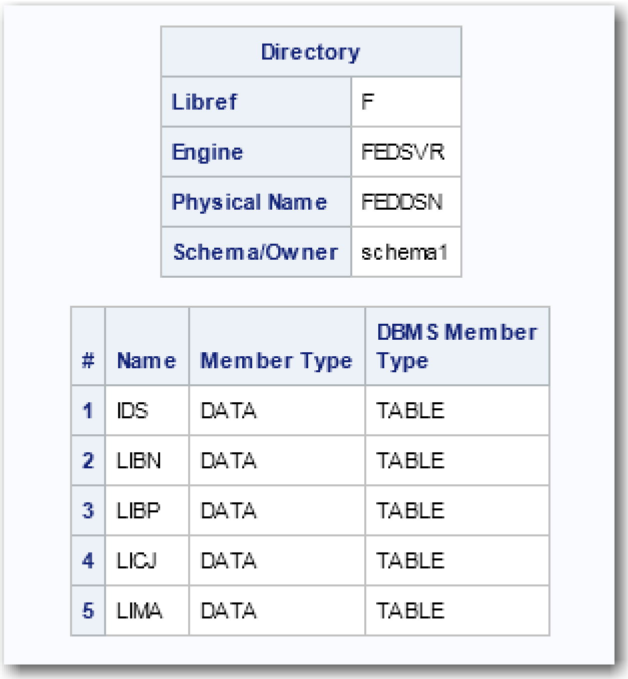
Program

```
libname f fedsvr server="d1234.us.company.com" port=2171
user="myid" password="mypwd" dsn=feddsn; 1

proc datasets lib=f; run; quit; 2

proc print data=f.ids; run; 3

proc print data=f.class2(SCHEMA=schema2); run; 4
```

Program Description

1. The LIBNAME statement assigns libref F to a DSN definition named FEDDSN, which defines access to two BASE SAS schemas: SCHEMA1 and SCHEMA2. The default behavior of the engine is to reference the first data source defined in the DSN, which is SCHEMA1. The SCHEMA= data source processing option could be specified in the LIBNAME statement to reference SCHEMA2 instead, but we do not do that here.
2. PROC DATASETS lists the tables that are available in SCHEMA1.

Display 9.1 *PROC DATASETS listing of SCHEMA1*



Directory	
Libref	F
Engine	FEDSVR
Physical Name	FEDDSN
Schema/Owner	schema1

#	Name	Member Type	DBMS Member Type
1	IDS	DATA	TABLE
2	LIBN	DATA	TABLE
3	LIBP	DATA	TABLE
4	LICJ	DATA	TABLE
5	LIMA	DATA	TABLE

3. The first PROC PRINT request specifies to print table F.IDS from SCHEMA1. The schema indicated in the LIBNAME statement is the active schema, unless the SCHEMA= data set option is used.

4. We have been told SCHEMA2 has a table in it named CLASS2. The second PROC PRINT specifies to print table F.CLASS2. The SCHEMA= data set option is used to change the focus of the FEDSVR engine from SCHEMA1 to SCHEMA2.

Using the SCHEMA= Data Source Processing Option

Details

This example shows how to use the SCHEMA= data source processing option to reference the second schema in a DSN that references two BASE SAS schemas.

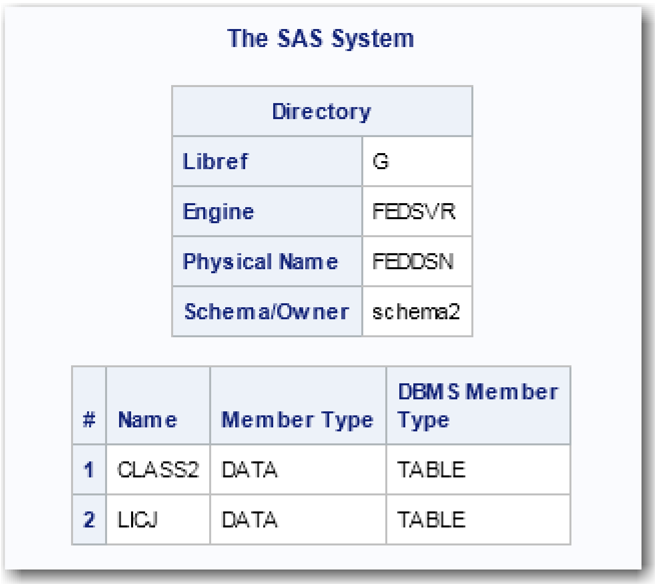
Program

```
libname g fedsvr server='1234.us.company.com' port=2171
user="myid" pwd="mypwd" DSN=FEDDSN schema=schema2;1

proc datasets lib=g; run; quit;2

proc print data=g.class2;run;3

proc print data=g.ids(schema=schema2); run;4
```

Program Description

1. The LIBNAME statement assigns libref G to the DSN definition named FEDDSN, which defines access to two BASE SAS schemas: SCHEMA1 and SCHEMA2. The SCHEMA= data source processing option is specified in the LIBNAME statement to reference SCHEMA2.
2. PROC DATASETS lists the tables that are available in SCHEMA2.

Display 9.2 *PROC DATASETS listing of SCHEMA2*



The SAS System

Directory	
Libref	G
Engine	FEDSVR
Physical Name	FEDDSN
Schema/Owner	schema2

#	Name	Member Type	DBMS Member Type
1	CLASS2	DATA	TABLE
2	LICJ	DATA	TABLE

3. The first PROC PRINT request specifies to print table G.CLASS2 from SCHEMA2. The schema indicated in the LIBNAME statement is the active schema.
4. The second PROC PRINT specifies to print table G.IDS from SCHEMA1. The SCHEMA= data set option is used to change the focus of the FEDSVR engine from SCHEMA2 to SCHEMA1.

Part 3

LIBNAME Statement Syntax and Options: Reference

Chapter 10
LIBNAME Statement for the FEDSVR Engine

Dictionary

LIBNAME Statement Syntax

Associates a SAS libref with data to be processed by the DataFlux Federation Server data access services.

Valid in:	Anywhere
Requirement:	The FEDSVR engine must be used with DataFlux Federation Server 3.1 or later.
Interactions:	The FEDSVR LIBNAME engine supports accessing only one data source at a time. For more information, see “Federated DSNs” on page 11. The engine processes nonexistent values as SAS missing values rather than ANSI SQL null values. For behavior differences, see “Null Values” on page 19.

Syntax

LIBNAME *libref* **FEDSVR**
server-connection-arguments
data-source-connection-arguments
<*data-source-processing-options*> ;

Arguments

The LIBNAME statement takes the following arguments:

libref
a valid SAS name that serves as an alias (shortcut) to the aggregate storage location of the data source or data sources. The data sources can be SAS data sets or third-party relational databases. A libref cannot exceed eight characters.

FEDSVR
the engine name that connects to the DataFlux Federation Server data access services that provide scalable, threaded, high-performance, and standards-based data access

technology. The engine establishes a remote connection, such that the current SAS session is a client to the DataFlux Federation Server.

Requirement: The engine name is required.

server-connection-arguments
provide connection information to the DataFlux Federation Server. For a list of server connection arguments, see "Server Connection Arguments" on page 42.

data-source-connection-arguments
provide connection information for accessing the data source or data sources. For the data source connection arguments, see "Data Source Connection Arguments" on page 43.

Requirement: You must specify data source connection information.

data-source-processing-options
define how data sources are processed. Some options enhance performance; others determine locking or naming behavior. The availability and default behavior of many of these options are data source specific. For a list of data source processing options, see "About the FEDSVR Engine LIBNAME Statement Options" on page 66.

CLEAR
disassociates one or more currently assigned librefs. Specify *libref* to disassociate a single libref. Specify _ALL_ CLEAR to disassociate all currently assigned librefs.

LIST
writes the attributes of one or more libraries to the SAS log. Specify *libref* to list the attributes of a single library. Specify _ALL_ LIST to list the attributes of all libraries that have librefs in your current SAS session.

Server Connection Arguments

The server connection arguments specify how to connect to the DataFlux Federation Server. The server connection arguments are as follows:

AUTHDOMAIN=*authentication-domain*
specifies the name of an authentication domain, which is a metadata object that manages the credentials (user ID and password) that are associated with the specified domain. Specifying an authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign on. An example is authdomain=FedServerAuth.

A SAS metadata administrator creates authentication domain definitions while creating a user definition with the User Manager in SAS Management Console. The authentication domain is associated with one or more login metadata objects that provide access to the DataFlux Federation Server. The domain name is resolved by the engine calling the metadata server and returning the authentication credentials.

Requirements:
The authentication domain and the associated login definition must be stored in a SAS Metadata Repository, and the SAS Metadata Server must be running to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

Interaction: If you specify AUTHDOMAIN=, do not specify USER= and PASSWORD=.

See: For complete information about creating and using authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

PASSWORD=*password*

specifies the password that corresponds to the user ID for the DataFlux Federation Server. The maximum length is 512 characters.

Alias: PWD=

Interaction: If the password is not specified and you are running interactively, SAS displays a dialog box to acquire the password for the session.

Tip: To specify an encoded password, use the PWENCODE procedure to disguise the text string, and then enter the encoded password for PASSWORD=. See the PWENCODE procedure in *Base SAS Procedures Guide*.

PORT=*number*

specifies the TCP port that the DataFlux Federation Server is listening to for connections. The default port number used in DataFlux Federation Server installations is 2171. The connection uses the SAS Bridge protocol. An example is `port=2171`.

Range: 0 - 65535

SERVER='*hostname* | *IPaddress*'

specifies either the host name or IP (Internet Protocol) address of the computer that hosts the DataFlux Federation Server. Here is an example: `server="d1234.us.company.com"` or `server="123.45.67.890"`. The maximum length is 256 characters.

Requirements:

You must specify the computer that hosts the DataFlux Federation Server, and you must enclose the host name or IP address in single or double quotation marks.

The DataFlux Federation Server must be running.

You must also specify DSN=.

Interactions:

Specifying SERVER= invokes the REMTS driver. The driver acts as a conduit between a client, which is the current SAS session, and a data source that is associated with the DataFlux Federation Server.

The REMTS driver is platform independent, which means that it can communicate from a SAS session on any platform with the DataFlux Federation Server on any platform.

USER=*ID*

specifies the user ID for logging on to the DataFlux Federation Server. The maximum length is 256 characters.

Alias: UID=

Requirement: The user ID must be capable of authenticating to the DataFlux Authentication Server. Users are registered on the DataFlux Authentication Server by the DataFlux Federation Server administrator.

Interaction: If the user ID is not specified and you are running interactively, SAS displays a dialog box to acquire the user ID for the session.

Data Source Connection Arguments

The data source connection arguments provide the connection information to access the data source. To access the data, you reference a DataFlux DSN, which is the name of an object that encapsulates the information needed to connect to the data source. The DSN contains the table driver name, the physical location of the data, plus any necessary authentication information that is required to retrieve data.

The data source connection arguments to reference a DataFlux DSN are as follows:

DSN=*dsn-definition*

references a DataFlux DSN, which encapsulates all of the information that is necessary to connect to a particular data source, configures the authorization that is enforced for data access, and identifies the SQL dialect that the application submits to the data source.

Requirements:

A DataFlux Federation Server administrator must have created the DSN definition.

If the DSN definition name is not a valid SAS name, enclose the DSN definition name in quotation marks. If the name contains single quotation marks, use double quotation marks around the name. Otherwise, use two single quotation marks in the name and enclose the name in single quotation marks.

If a user ID and password are defined for the DSN definition, you must specify DSNUSER= and DSNPASSWORD= with the DSN= definition name. SAS data sets do not support a user ID or password for a DSN definition.

To resolve the DSN, you or the user ID specified in DSNUSER= must have Connect privilege to the DSN definition, the data service that it references, or to the DataFlux Federation Server. For more information, see the *DataFlux Federation Server Administrator's Guide*.

Note: In its initial release, the FEDSVR engine supports only fetch and insert functionality for Greenplum data. Update and delete operations are not supported for Greenplum.

DSNPASSWORD="*password*"

specifies the password that corresponds to the user ID for the DataFlux DSN. The maximum length is data source dependent.

Alias: DSNPWD=

Requirements:

The password must be enclosed in double quotation marks.

To specify DSNPASSWORD=, you must specify DSN= and DSNUSER=.

Tip: You can use an encoded password in place of a plain-text password. For a method to encode your password, see the PWENCODE procedure in *Base SAS Procedures Guide*.

DSNUSER="*ID*"

enables you to access a DataFlux DSN with a user ID that is different from the default ID.

Alias: DSNUID=

Default: If DSNUSER= is not specified, the current user is assumed.

Requirements:

To specify DSNUSER=, you must specify DSN= and DSNPASSWORD=.

The user ID must be enclosed in double quotation marks.

Data Source Processing Options

The LIBNAME statement data source processing options control how a data source is processed. For example, some options enhance performance, whereas others determine locking or naming behavior. The availability and default behavior of many of these options are data source specific. For a list of the statement options, see Chapter 12, "Data Source Processing Options for the FEDSVR Engine," on page 65.

Chapter 11

System Options for the FEDSVR Engine

About SAS System Options for the FEDSVR Engine

System options are instructions that affect your SAS session. They control the way that operations are performed such as SAS System initialization, hardware and software interfacing, and the input, processing, and output of jobs and files.

Only a subset of SAS system options affect the operations of the FEDSVR engine. In addition, the availability and behavior of these options are data source specific.

Dictionary

BUFNO= System Option

Specifies the number of buffers to be allocated for processing SAS data sets.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Supports:	SAS data set

Syntax

BUFNO=*n*| *n*K | *n*M | *n*G | *n*T |*hex*X | MIN | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G | *n*T**
specifies the number of buffers to be allocated in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of `8` specifies 8 bytes, and a value of `3m` specifies 3,145,728 bytes.

***hex*X**
specifies the number of buffers as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value `2dx` specifies 45 buffers.

MIN
sets the minimum number of buffers to 0, which causes SAS to use the minimum optimal value for the operating environment. This is the default.

MAX
sets the number of buffers to the maximum possible number in your operating environment, up to the largest four-byte, signed integer which is 2^{31}-1, or approximately 2 billion.

CAUTION:
The recommended maximum for this option is 10.

Details

The number of buffers is not a permanent attribute of the data set; it is valid only for the current SAS session or job.

BUFNO= applies to SAS data sets that are opened for input, output, or update.

Using BUFNO= can improve execution time by limiting the number of input/output operations that are required for a particular SAS data set. The improvement in execution time, however, comes at the expense of increased memory consumption.

Operating Environment Information
The syntax that is shown here applies to the OPTIONS statement. In the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

Comparisons

You can override the BUFNO= system option by using the BUFNO= data set option.

BUFSIZE= System Option

Specifies the permanent buffer page size for output SAS data sets.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Supports:	SAS data set

Syntax

BUFSIZE=*n*| *n*K | *n*M | *n*G | *n*T |*hex*X | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G | *n*T**

specifies the page size in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of `8` specifies 8 bytes, and a value of `3m` specifies 3,145,728 bytes.

The default is 0, which causes SAS to use the minimum optimal page size for the operating environment.

***hex*X**

specifies the page size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value `2dx` sets the page size to 45 bytes.

MAX

sets the page size to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 2^{31}-1, or approximately 2 billion bytes.

Details

The page size is the amount of data that can be transferred from a single input/output operation to one buffer. The page size is a permanent attribute of the data set and is used when the data set is processed.

A larger page size can improve execution time by reducing the number of times SAS has to read from or write to the storage medium. However, the improvement in execution time comes at the expense of increased memory consumption.

To change the page size, use a DATA step to copy the data set and either specify a new page or use the SAS default.

Note: If you use the COPY procedure to copy a data set to another library that is allocated with a different engine, the specified page size of the data set is not retained.

Operating Environment Information

The default value for BUFSIZE= is determined by your operating environment and is set to optimize sequential access. To improve performance for direct (random) access, you should change the value for BUFSIZE=. For the default setting and possible settings for direct access, see the BUFSIZE= system option in the SAS documentation for your operating environment.

Operating Environment Information

The syntax that is shown here applies to the OPTIONS statement. In the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

Comparisons

The BUFSIZE= system option can be overridden by the BUFSIZE= data set option, unless the data set option specifies 0. Then, the system option overrides the data set option. The BUFSIZE= system option must be set to 0 to reset the page size to the default value for the operating environment.

COMPRESS= System Option

Specifies the compression of rows in output tables.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Supports:	SAS data set

Syntax

COMPRESS= NO | YES | CHAR | BINARY

Syntax Description

NO
specifies that the rows in a newly created table be uncompressed (fixed-length records).

YES|CHAR
specifies that the rows in a newly created table be compressed (variable-length records) using RLE (Run Length Encoding). RLE compresses rows by reducing repeated consecutive characters (including blanks) to two-byte or three-byte representations.

Tip: Use this compression algorithm for character data.

BINARY
specifies that the rows in a newly created table be compressed (variable-length records) using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file.

Tip: This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (numeric columns). Because the compression function operates on a single record at a time, the record length needs to be several hundred bytes or larger for effective compression.

Details

Compressing a file is a process that reduces the number of bytes required to represent each observation. Advantages of compressing a file include reduced storage requirements for the file and fewer I/O operations necessary to read or write to the data during processing. However, more CPU resources are required to read a compressed file (because of the overhead of uncompressing each observation), and there are situations when the resulting file size might increase rather than decrease.

Use the COMPRESS= system option to compress all output data sets that are created during a SAS session.

After a file is compressed, the setting is a permanent attribute of the file, which means that to change the setting, you must re-create the file. That is, to uncompress a file, specify COMPRESS=NO for a DATA step that copies the compressed file.

Note: For the COPY procedure, the default value CLONE uses the compression attribute from the input data set for the output data set. If the engine for the input data set does not support the compression attribute, then PROC COPY uses the current value of the COMPRESS= system option. For more information about CLONE and NOCLONE, see the COPY Statement options, DATASETS Procedure, in the *Base SAS Procedures Guide*.

Comparisons

The COMPRESS= system option can be overridden by the COMPRESS= connection string option, the COMPRESS= LIBNAME statement option, and the COMPRESS= data set option.

When you create a compressed file, you can also specify REUSE=YES (as a data set option or system option) in order to track and reuse space. With REUSE=YES, new observations are inserted in space freed when other observations are updated or deleted. When the default REUSE=NO is in effect, new observations are appended to the existing file.

FIRSTOBS= System Option

Specifies the row number or external file record that SAS processes first.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Supports:	All

Syntax

FIRSTOBS=*n* | *n*K | *n*M | *n*G | *n*T | *hex*X | MIN | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G | *n*T**

specifies the number of the first row or external file record to process in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of `8` specifies 8 bytes, and a value of `3m` specifies 3,145,728 bytes.

***hex*X**

specifies the number of the first row or the external file record to process as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value `2dx` specifies the 45th row.

MIN
sets the number of the first row or external file record to process to 1. This is the default.

MAX
sets the number of the first row to process to the maximum number of rows in the data sets or records in the external file, up to the largest eight-byte, signed integer, which is 2^{63}-1, or approximately 9.2 quintillion rows.

Details

The FIRSTOBS= system option is valid for all steps for the duration of your current SAS session or until you change the setting. To affect any single SAS data set, use the FIRSTOBS= data set option.

You can apply FIRSTOBS= processing to WHERE processing. For details about processing a segment of data that is conditionally selected, see the *SAS Language Reference: Concepts*.

Operating Environment Information
The syntax that is shown here applies to the OPTIONS statement. In the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the documentation for your operating environment.

Comparisons

- You can override the FIRSTOBS= system option by using the FIRSTOBS= data set option and by using the FIRSTOBS= option as a part of the INFILE statement.
- The FIRSTOBS= system option specifies a starting point for processing. The OBS= system option specifies an ending point.

Example: Specifying FIRSTOBS

If you specify FIRSTOBS=50, SAS processes the 50th row of the data set first.

This option applies to every input data set that is used in a program or a SAS process. In this example, SAS begins reading at the eleventh row in the data sets OLD, A, and B:

```
options firstobs=11;
libname myfiles fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn;
data myfiles.a;
   set myfiles.old; /* 100 rows */
run;
data myfiles.b;
   set myfiles.a;
run;
data myfiles.c;
   set myfiles.b;
run;
```

Data set OLD has 100 rows, data set A has 90, B has 80, and C has 70. To avoid decreasing the number of rows in successive data sets, use the FIRSTOBS= data set option in the SET statement. You can also reset FIRSTOBS=1 between a DATA step and a PROC step.

OBS= System Option

Specifies the last record to process.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Supports:	All

Syntax

OBS=*n* | *n*K | *n*M | *n*G | *n*T | *hex*X | MIN | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G | *n*T**

specifies a number to indicate when to stop processing, with *n* being an integer. Using one of the letter notations results in multiplying the integer by a specific value. That is, specifying K (kilobytes) multiplies the integer by 1,024; M (megabytes) multiplies by 1,048,576; G (gigabytes) multiplies by 1,073,741,824; or T (terabytes) multiplies by 1,099,511,627,776. For example, a value of **`20`** specifies 20 rows or records. A value of **`3m`** specifies 3,145,728 rows or records.

***hex*X**

specifies a number to indicate when to stop processing as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the hexadecimal value F8 must be specified as **`0F8x`** to specify the decimal equivalent of 248. The value **`2dx`** specifies the decimal equivalent of 45.

MIN

sets the number to 0 to indicate when to stop processing.

Interaction: When OBS=0 and the NOREPLACE option is in effect, SAS can still take certain actions because it actually executes each DATA and PROC step in the program, using no rows. For example, SAS executes procedures, such as CONTENTS and DATASETS, that process libraries or SAS data sets. External files are also opened and closed. Therefore, even if you specify OBS=0, when your program writes to an external file with a PUT statement, an end-of-file mark is written, and any existing data in the file is deleted.

MAX

sets the number to indicate when to stop processing to the maximum number of rows or records, up to the largest eight-byte, signed integer, which is 2^{63}-1, or approximately 9.2 quintillion. This is the default.

Details

OBS= tells SAS when to stop processing rows or records. For example, if OBS=10 is specified, the result is 10 rows or records.

OBS= is valid for all steps during your current SAS session or until you change the setting. You can also use OBS= to control analysis of SAS data sets in PROC steps.

In WHERE processing, SAS first subsets the data, and then applies OBS= to the subset. The FEDSVR engine does not have the concept of observation numbering from the original data set. It sends back the number of rows requested, numbered chronologically, regardless of where they occur in the data set.

If SAS is processing a raw data file, OBS= specifies the last line of data to read. SAS counts a line of input data as one row, even if the raw data for several SAS data set rows is on a single line.

Operating Environment Information

The syntax that is shown here applies to the OPTIONS statement. In the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

Comparisons

- An OBS= specification from either a data set option or an INFILE statement option takes precedence over the OBS= system option.
- The OBS= system option specifies an ending point for processing. The FIRSTOBS= system option specifies a starting point.

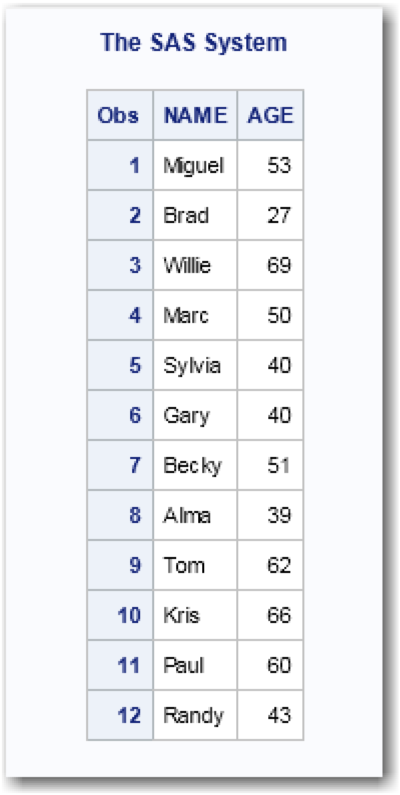
Examples

Example 1: Using OBS= to Specify When to Stop Processing Rows

This example illustrates the result of using OBS= to tell SAS when to stop processing rows. This example creates a SAS data set that contains 15 rows, executes the OPTIONS statement by specifying OBS=12, and then executes the PRINT procedure. The result is 12 rows.

```
libname myfiles fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn;
data myfiles.Ages;
   input Name $ Age;
   datalines;
Miguel 53
Brad 27
Willie 69
Marc 50
Sylvia 40
Gary 40
Becky 51
Alma 39
Tom 62
Kris 66
Paul 60
Randy 43
Barbara 52
Virginia 72
Arun 25
;
options obs=12;
proc print data=myfiles.Ages;
run;
```

Output 11.1 *PROC PRINT Output By Using OBS=*



The SAS System

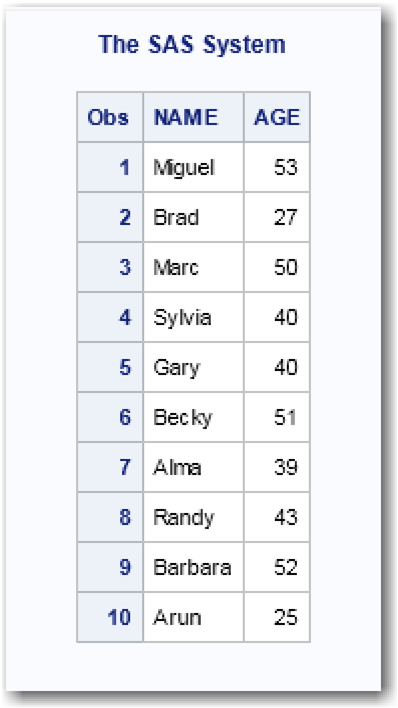
Obs	NAME	AGE
1	Miguel	53
2	Brad	27
3	Willie	69
4	Marc	50
5	Sylvia	40
6	Gary	40
7	Becky	51
8	Alma	39
9	Tom	62
10	Kris	66
11	Paul	60
12	Randy	43

Example 2: Using OBS= with WHERE Processing

This example illustrates the result of using OBS= along with WHERE processing. The example uses the data set that was created in Example 1, which contains 15 rows, and assumes that the SAS session has been reset to the defaults FIRSTOBS=1 and OBS=MAX. This example returns the first 10 rows that meet the WHERE criteria.

```
libname myfiles fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn;

options obs=10;
proc print data=myfiles.Ages;
  where Age LT 60;
run;
```

Output 11.2 *PROC PRINT Using a WHERE Statement and OBS=*


The SAS System

Obs	NAME	AGE
1	Miguel	53
2	Brad	27
3	Marc	50
4	Sylvia	40
5	Gary	40
6	Becky	51
7	Alma	39
8	Randy	43
9	Barbara	52
10	Arun	25

SASTRACE= System Option

Generates trace information about a data source.

Valid in:	OPTIONS statement, configuration file, SAS invocation
Default:	NONE
Supports:	All

Syntax

SASTRACE= ',,,d' | ' ,,d,' | ' d,' | ' ,,,s' | ' ,,,sa'

Syntax Description

',,,d'

specifies that all SQL statements sent to the data source are sent to the log. These statements include the following:

SELECT	DELETE
CREATE	SYSTEM CATALOG
DROP	COMMIT
INSERT	ROLLBACK
UPDATE	

For those data sources that do not generate SQL statements, the API calls, including all parameters, are sent to the log.

',,d,'
specifies that all routine calls are sent to the log. When this option is selected, all function enters and exits, as well as pertinent parameters and return codes, are traced. The information, however, will vary for each data source.

This option is most useful if you are having a problem and need to send a SAS log to SAS Technical Support for troubleshooting.

'd,'
specifies that all data source calls, such as API and Client calls, connection information, column bindings, column error information, and row processing are sent to the log. However, this information will vary for each data source.

This option is most useful if you are having a problem and need to send a SAS log to SAS Technical Support for troubleshooting.

',,,s'
specifies that a summary of timing information for calls made to the data source is sent to the log.

',,,sa'
specifies that timing information for each call made to the data source, along with a summary, is sent to the log.

Details

SASTRACE= is a powerful tool to use when you want to see the commands that are sent to your data source. SASTRACE= output is data source specific. However, most data sources will show you statements like SELECT or COMMIT as the data source processes them for the SAS application. The following details will help you manage SASTRACE= output:

- When using SASTRACE= on PC platforms, you must also specify the SASTRACELOC= system option.
- To turn SAS tracing off, you can specify the following option:

  ```
  options sastrace=off;
  ```
- Log output is much easier to read if you specify NOSTSUFFIX.

  Note: NOSTSUFFIX is not supported on z/OS.

The following code is entered without specifying the option, and the resulting log is longer and harder to decipher.

```
options sastrace=',,,d' sastraceloc=saslog;

libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;

proc print data=mydblib.snow_birthdays;
run;
```

The resulting log is as follows:

```
  17 1544121286 du_prep 825 PRINT
SASTSE_5: Prepared: on connection 0 18 1544121286 du_prep 825 PRINT
SELECT * FROM SNOW_BIRTHDAYS 19 1544121286 du_prep 825 PRINT
  20 1544121286 du_prep 825 PRINT
  21 1544121286 du_exec 1795 PRINT
SASTSE_6: Executed: on connection 0 22 1544121286 du_exec 1795 PRINT
Prepared statement SASTSE_5 23 1544121286 du_exec 1795 PRINT
  24 1544121286 du_exec 1795 PRINT
```

However, by using NOSTSUFFIX, the log file becomes much easier to read:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;

libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;

proc print data=mydblib.snow_birthdays;
run;
```

The resulting log is as follows:

```
SASTSE_1: Prepared: on connection 0
SELECT * FROM SNOW_BIRTHDAYS

8          proc print data=mydblib.snow_birthdays; run;


SASTSE_2: Executed: on connection 0
Prepared statement SASTSE_1
```

Examples

Example 1: SQL Trace ',,,d'

The examples in this section are based on the following table, and are shown using NOSTSUFFIX and SASTRACELOC=SASLOG.

```
data work.winter_birthdays;
   input empid birthdat date9. lastname $18.;
   format birthdat date9.;
datalines;
678999 28DEC1966 PAVEO            JULIANA        3451
456788 12JAN1977 SHIPTON          TIFFANY        3468
890123 20FEB1973 THORSTAD         EDVARD         3329
;
run;
```

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;
```

The output is written to the SAS log, as specified by the SASTRACELOC=SASLOG system option.

Log 11.1 *SAS Log Output from the SASTRACE=',,,d' System Option*

```
10   options sastrace=',,,d' sastraceloc=saslog nostsuffix;
SASTSE: AUTOCOMMIT is NO for connection 0
11         options sastrace=',,,d' sastraceloc=saslog nostsuffix;
12         libname mydblib fedsvr
12       !
connect_string=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXX
12       ! XXXXXXXXXXXXXXXXXXX;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:        FEDSVR
      Physical Name:
13
14         proc delete data=mydblib.snow_birthdays;run;
SASTSE: AUTOCOMMIT is NO for connection 1
SASTSE: AUTOCOMMIT turned ON for connection id 1
SASTSE_1: Executed: on connection 1
DROP  TABLE SNOW_BIRTHDAYS
SASTSE: 0 row(s) affected by INSERT/UPDATE/DELETE or other statement.
NOTE: Deleting MYDBLIB.SNOW_BIRTHDAYS (memtype=DATA).
NOTE: PROCEDURE DELETE used (Total process time):
      real time           0.28 seconds
      cpu time            0.04 seconds
15
16         data mydblib.snow_birthdays;
17         set work.winter_birthdays;
18         run;
SASTSE_2: Prepared: on connection 1
SELECT * FROM SNOW_BIRTHDAYS WHERE 0=1
SASTSE: AUTOCOMMIT is NO for connection 2
SASTSE_3: Executed: on connection 2
CREATE TABLE SNOW_BIRTHDAYS (empid DOUBLE,birthdat DATE,lastname VARCHAR(18))
SASTSE: 0 row(s) affected by INSERT/UPDATE/DELETE or other statement.
SASTSE: COMMIT performed on connection 2.
SASTSE_4: Prepared: on connection 2
INSERT INTO SNOW_BIRTHDAYS (empid,birthdat,lastname)  VALUES ( ? , ? , ? )
```

```
SASTSE_5: Executed: on connection 2
Prepared statement SASTSE_4
SASTSE_6: Executed: on connection 2
2                                         The SAS System            11:39 Friday,
December 5, 2008
Prepared statement SASTSE_4
SASTSE_7: Executed: on connection 2
Prepared statement SASTSE_4
NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
SASTSE: COMMIT performed on connection 2.
NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables.
SASTSE: COMMIT performed on connection 2.
SASTSE: COMMIT performed on connection 2.
NOTE: DATA statement used (Total process time):
      real time           0.29 seconds
      cpu time            0.01 seconds
19
SASTSE: AUTOCOMMIT turned ON for connection id 0
SASTSE_8: Prepared: on connection 0
SELECT * FROM SNOW_BIRTHDAYS
20         proc print data=mydblib.snow_birthdays;run;
SASTSE_9: Executed: on connection 0
Prepared statement SASTSE_8
NOTE: There were 3 observations read from the data set MYDBLIB.SNOW_BIRTHDAYS.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           3.04 seconds
      cpu time            0.34 seconds
21
22
23         libname mydblib clear;
NOTE: Libref MYDBLIB has been deassigned.
```

Example 2: Log Trace ',,d,'

```
options sastrace=',,d,' sastraceloc=saslog nostsuffix;
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;
```

The output is written to the SAS log, as specified by the SASTRACELOC=SASLOG system option.

Log 11.2 *SAS Log Output from the SASTRACE=',,d,' System Option*

```
11   options sastrace=',,d,' sastraceloc=saslog nostsuffix;
12
ACCESS ENGINE:  Entering dbiconi.
ACCESS ENGINE:  Exiting dbiconi.  rc=0x00000000
ACCESS ENGINE: Entering DBICON
ACCESS ENGINE:  CONNECTION= SHAREDREAD
SASTSE: Successful connection made, connection id 0
ACCESS ENGINE: Successful physical conn id 0
ACCESS ENGINE: Number of connections is 1
ACCESS ENGINE: Exiting DBICON with  rc=0X00000000
13         options sastrace=',,d,' sastraceloc=saslog nostsuffix;
14
15         libname mydblib fedsvr
15       !
connect_string=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXX
15       ! XXXXXXXXXXXXXXXXXXXX;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:        FEDSVR
      Physical Name:
16
17         data mydblib.snow_birthdays;
18         set work.winter_birthdays;
19         run;
NOTE: Libref MYDBLIB has been deassigned.
ACCESS ENGINE:  Entering yoeopen
ACCESS ENGINE: Entering DBIEXST with table name being SNOW_BIRTHDAYS
ACCESS ENGINE: Using a utility connection for verifying table existence
ACCESS ENGINE: Entering DBICON
ACCESS ENGINE:  CONNECTION= SHAREDREAD
SASTSE: Successful connection made, connection id 1
ACCESS ENGINE: Successful physical conn id 1
ACCESS ENGINE: Number of connections is 2
ACCESS ENGINE: Exiting DBICON with  rc=0X00000000
ACCESS ENGINE: Entering dbiopen
SASTSE: Enter duopen, table is SNOW_BIRTHDAYS, openmode is INPUT, statement 0,
connection 1
SASTSE: Using FETCH for file SNOW_BIRTHDAYS on connection 1
SASTSE: Enter setconloc, table is SNOW_BIRTHDAYS, statement 0, connection 1
SASTSE: Exit duopen, rc = 0x00000000
ACCESS ENGINE: Successful dbiopen, open id 0, connect id 1
ACCESS ENGINE: Exit dbiopen with rc=0X00000000
SASTSE: Enter duexist, table is SNOW_BIRTHDAYS, statement 0, connection 1
SASTSE: Exit duexist, table DOES NOT exist
ACCESS ENGINE: Entering dbiclose
SASTSE: Enter duclose, table is SNOW_BIRTHDAYS, statement 0, connection 1
SASTSE: Exit duclose, rc = 0x00000000
ACCESS ENGINE: DBICLOSE open_id 0, connect_id 1
ACCESS ENGINE: Exiting dbiclos with rc=0X00000000
ACCESS ENGINE: Exit DBIEXST rc=0X00403809
ACCESS ENGINE:  Open Mode is XO_OUTPUT
ACCESS ENGINE:  Access Mode is XO_SEQ
ACCESS ENGINE:  Shr flag is XHSHRMEM
ACCESS ENGINE: Entering DBICON
ACCESS ENGINE:  CONNECTION= SHAREDREAD
SASTSE: Successful connection made, connection id 2
ACCESS ENGINE: Successful physical conn id 2
ACCESS ENGINE: Number of connections is 3
ACCESS ENGINE: Exiting DBICON with  rc=0X00000000
ACCESS ENGINE: Entering dbiopen
SASTSE: Enter duopen, table is SNOW_BIRTHDAYS, openmode is OUTPUT, statement
-99, connection 2
SASTSE: Using FETCH for file SNOW_BIRTHDAYS on connection 2
SASTSE: Enter setconloc, table is SNOW_BIRTHDAYS, statement -99, connection 2
SASTSE: Exit duopen, rc = 0x00000000
ACCESS ENGINE: Successful dbiopen, open id 0, connect id 2
ACCESS ENGINE: Exit dbiopen with rc=0X00000000
ACCESS ENGINE: Exit yoeopen with SUCCESS.
ACCESS ENGINE:  Begin yoeinfo
ACCESS ENGINE: Exit yoeinfo with SUCCESS.
SASTSE: Enter duoload, table is SNOW_BIRTHDAYS, statement 0, connection 2
ACCESS ENGINE: Entering dbrload with SQL Statement set to
          CREATE TABLE SNOW_BIRTHDAYS (empid DOUBLE,birthdat DATE,lastname
VARCHAR(18))
SASTSE: Enter duexec, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit duexec, rc = 0x00000000
SASTSE: Enter duforc, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit duforc, rc = 0x00000000
SASTSE: Exit duoload, rc = 0x00000000
SASTSE: Enter duins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Enter prepins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit prepins, rc = 0x00000000
SASTSE: Enter doins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit doins, rc = 0x00000000
SASTSE: Exit duins, rc = SUCCESS
SASTSE: Enter duins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Enter doins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit doins, rc = 0x00000000
SASTSE: Exit duins, rc = SUCCESS
SASTSE: Enter duins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Enter doins, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit doins, rc = 0x00000000
SASTSE: Exit duins, rc = SUCCESS
NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
SASTSE: Enter duforc, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit duforc, rc = 0x00000000
NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables.
ACCESS ENGINE:  Enter yoeclos
ACCESS ENGINE: Entering dbiclose
SASTSE: Enter duclose, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Enter duforc, table is SNOW_BIRTHDAYS, statement 0, connection 2
SASTSE: Exit duforc, rc = 0x00000000
SASTSE: Exit duclose, rc = 0x00000000
ACCESS ENGINE: DBICLOSE open_id 0, connect_id 2
ACCESS ENGINE: Exiting dbiclos with rc=0X00000000
ACCESS ENGINE: Entering DBIDCON
SASTSE: Successful disconnection, connection id 2
SASTSE: Successful CLI free environment from connection 1
ACCESS ENGINE: Physical disconnect on id = 1
ACCESS ENGINE: Exiting DBIDCON with rc=0X00000000, rc2=0X00000000
20
21   libname mydblib clear;
```

Example 3: Time Trace ',,,s'

```
options sastrace=',,,s' sastraceloc=saslog nostsuffix;
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
```

```
data mydblib.snow_birthdays;
   set work.winter_birthdays;
run;
libname mydblib clear;
```

The output is written to the SAS log, as specified by the SASTRACELOC=SASLOG system option.

Log 11.3 *SAS Log Output from the SASTRACE= ',,,s' System Option*

```
11         options sastrace=',,,s' sastraceloc=saslog nostsuffix;
12
13         libname mydblib fedsvr
connect_string=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXX;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:        FEDSVR
      Physical Name:
14
15         data mydblib.snow_birthdays;
16         set work.winter_birthdays;
17         run;
Summary Statistics for FEDSVR are:
Total SQL prepare seconds were:                     0.009349
Total seconds used by the FEDSVR ACCESS engine were     0.010446
NOTE: There were 3 observations read from the data set WORK.WINTER_BIRTHDAYS.
NOTE: The data set MYDBLIB.SNOW_BIRTHDAYS has 3 observations and 3 variables.
Summary Statistics for FEDSVR are:
Total SQL execution seconds were:                   0.092450
Total SQL prepare seconds were:                     0.026569
Total seconds used by the FEDSVR ACCESS engine were     0.127992
NOTE: DATA statement used (Total process time):
      real time           0.46 seconds
      cpu time            0.06 seconds
18
19         libname mydblib clear;
NOTE: Libref MYDBLIB has been deassigned.
```

SASTRACELOC= System Option

Writes trace information to a specified location.

Valid in:	OPTIONS statement, configuration file, SAS invocation
Default:	stdout
Supports:	All

Syntax

SASTRACELOC= stdout | SASLOG | FILE *'path-and-filename'*

Syntax Description

stdout
specifies to write the trace information to the default output location for your operating environment. This is the default setting.

SASLOG
specifies to write the trace information to the SAS log.

FILE '*path-and-filename*'
specifies to write the trace information to an external file. You must enclose the external file specification in quotation marks.

Details

SASTRACELOC= enables you to specify where to put the trace messages that are generated by SASTRACE=. By default, the output goes to the default output location for your operating environment. You can send the output to a SAS log by specifying SASTRACELOC=SASLOG.

Note: This option and its values might differ for each host.

Example: Specifying the SASTRACELOC= System Option

The following example on a PC platform writes the trace information to the TRACE.LOG file in the work directory on the C drive.

```
options sastrace=',,,d' sastraceloc=file 'c:\work\trace.log';
```

VALIDVARNAME= System Option

Specifies the rules for valid SAS column names that can be created and processed during a SAS session.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Default:	V7
Requirement:	You must also specify PRESERVE_COL_NAMES=YES LIBNAME statement data source processing option.
Supports:	All

Syntax

VALIDVARNAME=V7 | UPCASE | ANY

Syntax Description

V7
specifies that column names must follow these rules:

- can be up to 32 characters in length.
- must begin with a letter of the Latin alphabet (A - Z, a - z) or the underscore character. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- cannot contain blanks.

- cannot contain special characters except for the underscore.
- can contain mixed-case letters. SAS stores and writes the column name in the same case that is used in the first reference to the column. However, when SAS processes a column name, SAS internally converts it to uppercase. You cannot, therefore, use the same column name with a different combination of uppercase and lowercase letters to represent different columns. For example, `cat`, `Cat`, and `CAT` all represent the same column.
- cannot be assigned the names of special SAS automatic columns (such as _N_ and _ERROR_) or column list names (such as _NUMERIC_, _CHARACTER_, and _ALL_).

UPCASE

specifies that the column name follows the same rules as V7, except that the column name is uppercase, as in earlier versions of SAS.

ANY

specifies that SAS column names must follow these rules:

- can be up to 32 characters in length.
- can begin with or contain any characters, including blanks.

  Note: If you use any characters other than the ones that are valid when the VALIDVARNAME system option is set to V7 (letters of the Latin alphabet, numerals, or underscores), then you must express the column name as a *name literal* and you must set VALIDVARNAME=ANY. See “SAS Name Literals” and “Avoiding Errors When Using Name Literals” in *SAS Language Reference: Concepts*.
- can contain mixed-case letters. SAS stores and writes the column name in the same case that is used in the first reference to the column. However, when SAS processes a column name, SAS internally converts it to uppercase. You cannot, therefore, use the same column name with a different combination of uppercase and lowercase letters to represent different columns. For example, `cat`, `Cat`, and `CAT` all represent the same column.

Note: For more information about SAS naming, see *SAS Language Reference: Concepts*.

Chapter 12

Data Source Processing Options for the FEDSVR Engine

About the FEDSVR Engine LIBNAME Statement Options

LIBNAME statement options control how a data source is processed. For example, some options enhance performance, while others determine locking or naming behavior. Only a subset of LIBNAME statement options that are provided by SAS for relational databases are available for the FEDSVR engine. The availability and behavior of these options are data source specific.

Dictionary

ACCESS= LIBNAME Statement Option

Determines the access level with which a libref connection is opened.

Valid in:	LIBNAME statement
Default:	none
Supports:	All

Syntax

ACCESS=READONLY

Syntax Description

READONLY
specifies that tables can be read but not updated.

Details

Using this option prevents writing to the data source. If this option is omitted, tables can be read and updated if you have the necessary data privileges.

AUTOCOMMIT= LIBNAME Statement Option

Indicates whether updates are committed immediately after they are submitted.

Valid in:	LIBNAME statement
Supports:	DB2 UNIX/PC, Greenplum, ODBC, MySQL

Syntax

AUTOCOMMIT= YES | NO

Syntax Description

YES
specifies that all updates, deletes, and inserts are committed (that is, saved to a table) immediately after they are submitted, and no rollback is possible.

NO
specifies that the commit operation is automatically performed when processing reaches the DBCOMMIT= value, or the default number of rows if DBCOMMIT is not set.

Details

The default is NO if the data source supports transactions and the connection is used for updating data.

For MySQL, the default is YES.

COMPRESS= LIBNAME Statement Option

Specifies the compression of rows in output tables.

Valid in:	LIBNAME statement
Default:	No
Supports:	SAS data set

Syntax

COMPRESS=NO | YES | CHAR | BINARY

Syntax Description

NO
specifies that the rows in a newly created table be uncompressed (fixed-length records).

YES | CHAR
specifies that the rows in a newly created table be compressed (variable-length records) using RLE (Run Length Encoding). RLE compresses rows by reducing repeated consecutive characters (including blanks) to two-byte or three-byte representations.

Tip: Use this compression algorithm for character data.

BINARY
specifies that the rows in a newly created table be compressed (variable-length records) using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file.

Interaction: For the COPY procedure, the default value CLONE uses the compression attribute from the input data set for the output data set instead of the value specified on the COMPRESS= option. This interaction does not apply when using SAS/SHARE or SAS/CONNECT.

Tip: This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (numeric columns). Because the compression function operates on a single record at a time, the record length needs to be several hundred bytes or larger for effective compression.

CONNECTION= LIBNAME Statement Option

Specifies whether operations on a single libref can share a connection to the DBMS.

Valid in:	LIBNAME statement
Default:	SHAREDREAD
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

CONNECTION=SHARED | SHAREDREAD | UNIQUE

Syntax Description

SHARED

specifies that all operations that access DBMS tables in a single libref share a single connection.

Use this option with caution. When a single SHARED connection is used for multiple table opens, a commit or rollback performed on one table being updated also applies to all other tables opened for update. Even if a table is opened for read, its read cursor might be resynchronized as a result of this commit or rollback. If the cursor is resynchronized, there is no guarantee that the new table will match the original table that was being read.

Use SHARED to eliminate the deadlock that can occur when you create and load a DBMS table from an existing table that resides in the same database or tablespace. This only happens in certain output processing situations and is the only recommended use for CONNECTION=SHARED.

SHAREDREAD

specifies that all read operations that access DBMS tables in a single libref share a single connection. A separate connection is established for every table that is opened for update or output operations.

Where available, this is the default because it offers the best performance and it guarantees data integrity.

UNIQUE

specifies that a separate connection is established every time a DBMS table is accessed by your SAS application.

Use UNIQUE if you want each use of a table to have its own connection.

Details

Typically, each DBMS connection has one transaction, or work unit, that is active in the connection. This transaction is affected by commits or rollbacks that are performed within the connection while executing the SAS application. The CONNECTION= option enables you to control the number of connections, and therefore transactions, that are executed and supported for each LIBNAME statement.

For ODBC databases, if the data source supports only one active open cursor per connection, the default value is CONNECTION=UNIQUE. Otherwise, the default value is CONNECTION=SHAREDREAD.

DBCOMMIT= LIBNAME Statement Option

Causes an automatic COMMIT (a permanent writing of data to the DBMS) after a specified number of rows have been processed.

Valid in:	LIBNAME statement
Default:	1000 when inserting rows into a DBMS table; 0 when updating a DBMS table
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

DBCOMMIT=*n*

Syntax Description

n
is an integer greater than or equal to 0.

Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows that are processed includes rows that are not processed successfully. If you set DBCOMMIT=0, a commit is issued only once (after the procedure or DATA step completes).

If the DBCOMMIT= option is explicitly set, the engine fails any update that has a WHERE clause.

SAS data sets cannot be rolled back, so for SAS data sets, this option has no effect. However, if explicitly set, the engine still fails any update that has a WHERE clause even though the value specified has no effect.

See Also

To apply this option to an individual table, use the DBCOMMIT= data set option.

DBGEN_NAME= LIBNAME Statement Option

Specifies how SAS automatically renames columns that contain characters that SAS does not allow, such as $, to valid SAS column names.

Valid in:	LIBNAME statement
Default:	DBMS
Supports:	All

Syntax

DBGEN_NAME= DBMS | SAS

Syntax Description

DBMS
: specifies that the columns are renamed to valid SAS column names. Disallowed characters are converted to underscores. If a column is converted to a name that already exists, a sequence number is appended to the end of the new name.

SAS
: specifies that DBMS columns are renamed to the format _COL*n*, where *n* is the column number (starting with zero). The LIBNAME option PRESERVE_COL_NAMES=YES and the global option VALIDVARNAME=ANY must also be specified.

Details

SAS retains column names when it reads data from tables, unless a column name contains characters that SAS does not allow. When the engine reads column names that contain characters that are not allowed in SAS names, the default behavior is to replace an unsupported character with an underscore.

For example, if you specify DBGEN_NAME=SAS, a DBMS column named **`Dept$Amt`** is renamed to **`_COLn`**. If you specify DBGEN_NAME=DBMS, the **`Dept$Amt`** column is renamed to **`Dept_Amt`**.

Note: SAS does not allow names that include blank spaces or special characters such as @,#,%, $ unless the VALIDVARNAME option is set to ANY.

When the engine encounters a DBMS name that exceeds 32 characters, it truncates the name.

This option is intended primarily for National Language Support, notably for the conversion of kanji to English characters. English characters that are converted from kanji are often those that are not allowed in SAS.

See Also

To apply this option to an individual table, use the DBGEN_NAME= data set option.

DBINDEX= LIBNAME Statement Option

Improves performance when processing a join that involves a large DBMS table and a small SAS data set.

Valid in:	LIBNAME statement
Default:	DBMS-specific
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

DBINDEX=YES | NO

Syntax Description

YES
: specifies that SAS uses columns in the WHERE clause that have defined DBMS indexes.

NO
specifies that SAS does not use indexes that are defined on DBMS columns.

Details

When you are processing a join that involves a large DBMS table and a relatively small SAS data set, you might be able to use DBINDEX= to improve performance.

CAUTION:
Improper use of this option can degrade performance.

See Also

To apply this option to an individual table, use the DBINDEX= data set option.

DBLIBINIT= LIBNAME Statement Option

Specifies a valid data source command to be executed once within the scope of the LIBNAME statement or libref that established the first connection to the data source.

Valid in:	LIBNAME statement
Default:	none
Supports:	All

Syntax

DBLIBINIT=< '> *command*<'>

Syntax Description

command
is any SQL or DataFlux FedSQL language statement, DBMS script, stored procedure, or native DBMS command that is supported by the driver. The command must not return a result set.

Details

The initialization command that you select can be a script, stored procedure, or any DataFlux FedSQL language or DBMS SQL statement that might provide additional control over the interaction between the LIBNAME engine and the data source.

The command executes immediately after the first connection is successfully established. If the command fails, a disconnect occurs and the libref is not assigned. You must specify the command as a single, quoted string, unless it is an environment variable.

DBLIBINIT= fails if either CONNECTION=UNIQUE or DEFER=YES, or if both of these LIBNAME statement options are specified.

Example: Specify an Initialization Command for All Operations in a Libref

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn connection=shared dblibinit='Test';
```

See Also

"DBLIBTERM= LIBNAME Statement Option" on page 72

DBLIBTERM= LIBNAME Statement Option

Specifies a valid data source command to be executed once, before the data source that is associated with the last connection made by the LIBNAME statement or libref disconnects.

Valid in:	LIBNAME statement
Default:	none
Supports:	All

Syntax

DBLIBTERM=< '> *command*<'>

Syntax Description

command
is any SQL or DataFlux FedSQL language statement, DBMS script, stored procedure, or native DBMS command that is supported by the driver. The command must not return a result set.

Details

The termination command that you select can be a script, stored procedure, or any DataFlux FedSQL language or DBMS SQL statement that might provide additional control over the interaction between the engine and the data source. The command executes immediately before SAS terminates the last connection to the data source. If the command fails, a warning message is issued but the library deassignment and disconnect still occurs. You must specify the command as a single, quoted string.

DBLIBTERM= fails if CONNECTION=UNIQUE or DEFER=YES is specified.

Example: Insert a Row After a Library is Cleared

In this example, Row 2 is added after the library has been cleared.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn1;

data mydblib.mytab;
   x=1; y='one';
run;
```

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn1
   dblibterm="insert into mytab values (2, 'two')";

libname mydblib clear;
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn1;

proc sql;
   select * from mydblib.mytab;
quit;
```

DBNULLKEYS= LIBNAME Statement Option

Controls the format of the WHERE clause when you use the DBKEY= data set option.

Valid in:	LIBNAME statement
Default:	DBMS-specific
Supports:	All

Syntax

DBNULLKEYS=YES | NO

Details

If there might be NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, use DBNULLKEYS=YES. This is the default for most datasources. When you specify DBNULLKEYS=YES and specify a column that is not defined as NOT NULL in the DBKEY= data set option, SAS generates a WHERE clause that can find NULL values. For example, if you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause with the following syntax:

```
where ((column = ?) or ((column is null) and (? is null)))
```

This syntax enables SAS to prepare the statement once and use it for any value (NULL or NOT NULL) in the column. Note that this syntax has the potential to be much less efficient than the shorter form of the WHERE clause (presented below). When you specify DBNULLKEYS=NO or specify a column that is defined as NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause.

If you know that there are no NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, you can use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and specify DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause (regardless of whether the column specified in DBKEY= is defined as NOT NULL):

```
where (column = ?)
```

See Also

To apply this option to an individual table, use the DBNULLKEYS= data set option.

DBPROMPT= LIBNAME Statement Option

Specifies whether a window prompts the user to enter connection information before connecting to the data source in interactive mode.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

DBPROMPT=YES | NO

Syntax Description

YES
specifies that a window interactively prompts you for the data source connection options the first time the libref is used.

NO
specifies that you are not prompted for the data source connection options. You must specify either a DSN definition or a fully specified connection string using CONNECT_STRING=. NO is the default.

Details

If you specify DBPROMPT=YES, it is not necessary to provide connection options with the LIBNAME statement. If you specify connection options with the LIBNAME statement and you specify DBPROMPT=YES, the connection option values are displayed in the window (except for the password value, which appears as a series of asterisks). All of these values can be overridden interactively.

The DBPROMPT= option interacts with the DEFER= option to determine when the prompt window appears. If DEFER=NO, the DBPROMPT window appears when the LIBNAME statement is executed. If DEFER=YES, the DBPROMPT window appears the first time a table is opened. The DEFER= option normally defaults to NO but defaults to YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

The DBPROMPT window usually opens only once for each time that the LIBNAME statement is specified. It might open multiple times if DEFER=YES and the connection fails when the engine tries to open a table. In these cases, the DBPROMPT window appears until a successful connection occurs or you click **Cancel**.

The maximum password length for most of the data sources is 32 characters.

Examples

Example 1: Prompt for Connection Information

In the following example, values provided in the LIBNAME statement are pulled into the DBPROMPT window. The values `user1` and `basedsn` appear in the DBPROMPT window and can be edited and confirmed by the user. The password value appears in the DBPROMPT window as a series of asterisks; it can also be edited by the user.

```
libname seclib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn dbprompt=yes defer=no;
```

Example 2: Defer Prompt for Connection Information

In the following example, the DBPROMPT window does not open when the LIBNAME statement is submitted because DEFER=YES. The DBPROMPT window appears when the PRINT procedure is processed. The values **`user1`** and **`basedsn`** appear in the DBPROMPT window and can be edited and confirmed by the user. The password value appears in the DBPROMPT window as a series of asterisks; it can also be edited by the user. Upon confirmation, the connection is made, and the table is opened.

```
libname seclib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn dbprompt=yes defer=yes;

proc print data=lib1.staff; run;
```

DBSASLABEL= LIBNAME Statement Option

Specifies how the table driver returns column labels.

Valid in:	LIBNAME statement
Default:	COMPAT
Supports:	All

Syntax

DBSASLABEL=COMPAT | NONE

Syntax Description

COMPAT
: specifies to return the column label to the application. For data sources that support storing column labels on the table (for example, SAS data sets), the engine returns the label to the application. If there is no label stored, no label is returned. For data sources that do not store column labels on the table, the engine returns the column name as the label.

NONE
: specifies that column labels are not returned even if one exists. The engine returns blanks for the column labels.

Example: Return Blank Labels for Aliases in Headings

The following example demonstrates how DBSASLABEL= is used as a LIBNAME option to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn dbsaslabel=none;
```

```
proc sql;
   select deptno as Department ID, loc as Location
      from mydblib.dept;
```

Without the DBSASLABEL= option set to NONE, the aliases would be ignored, and DEPTNO and LOC would be used as column headings in the result set.

See Also

To apply this option to an individual table, use the DBSASLABEL= data set option.

DEFER= LIBNAME Statement Option

Specifies when the connection to the data source occurs.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

DEFER=NO | YES

Syntax Description

NO
specifies that the connection to the data source occurs when the libref is assigned by the LIBNAME statement.

YES
specifies that the connection to the data source occurs when a data source table is opened.

Details

The default value NO is overridden if DBPROMPT=YES.

DIRECT_EXE= LIBNAME Statement Option

Lets you pass a SAS SQL procedure DELETE statement directly to a data source to handle, which can improve performance.

Valid in:	LIBNAME statement
Default:	none
Supports:	All

Syntax

DIRECT_EXE=DELETE

Syntax Description

DELETE
specifies that a PROC SQL DELETE statement is passed directly to the data source for processing.

Details

If DIRECT_EXE=DELETE is not specified, the PROC SQL DELETE statement is processed by SAS, which reads the table and deletes one row at a time. Specifying DIRECT_EXE=DELETE can improve performance because the PROC SQL DELETE statement is passed directly to the data source, which then deletes all the rows in the table.

Example: Empty a Table from a Database

The following example demonstrates the use of DIRECT_EXE= to empty a table.

```
libname x fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn direct_exe=delete;
data x.db_dft; /*create the SAS data set of 5 rows */
   do col1=1 to 5;
   output;
   end;
run;
options sastrace=",,,d" sastraceloc=saslog nostsuffix;
proc sql;
   delete * from x.db_dft;  /*this delete statement
         is passed directly to the data source*/
quit;
```

By turning trace on, you should see something similar to the following:

Output 12.1 *SAS Log Output*

```
SASTSE_2: Executed: on connection 1
delete from db_dft
```

DIRECT_SQL= LIBNAME Statement Option

Lets you specify whether generated SQL is passed to the data source for processing.

Valid in:	LIBNAME statement
Default:	YES
Supports:	All

Syntax

DIRECT_SQL=YES | NO | NONE | NOGENSQL | NOWHERE |NOFUNCTIONS | NOMULTOUTJOINS

Syntax Description

YES
specifies that generated SQL from PROC SQL is passed directly to the data source for processing.

NO
specifies that generated SQL from PROC SQL is not passed to the data source for processing. This is the same as specifying the value NOGENSQL.

NONE
specifies that generated SQL is not passed to the data source for processing. This includes SQL that is generated from PROC SQL, SAS functions that can be converted into DBMS functions, joins, and WHERE clauses.

NOGENSQL
prevents PROC SQL from generating SQL to be passed to the DBMS for processing.

NOWHERE
prevents WHERE clauses from being passed to the data source for processing. This includes SAS WHERE clauses and PROC SQL generated or PROC SQL specified WHERE clauses.

NOFUNCTIONS
prevents SQL statements from being passed to the data source for processing when they contain functions.

NOMULTOUTJOINS
specifies that PROC SQL will not attempt to pass any multiple outer joins to the data source for processing. Other join statements might be passed down however, including portions of a multiple outer join.

Details

By default, processing is passed to the data source when possible, because the data source might be able to process the functionality more efficiently than SAS does. In some instances, however, you might not want the data source to process the SQL. For example, the presence of null values in DBMS data might cause different results depending on whether the processing takes place in SAS or in the DBMS. If you do not want the data source to handle the SQL, use DIRECT_SQL= to force SAS to handle some or all of the SQL processing.

If you specify DIRECT_SQL=NOGENSQL, PROC SQL does not generate data source SQL. This means that SAS functions, joins, and DISTINCT processing that occur *within* PROC SQL are not passed to the DBMS for processing. (SAS functions *outside* PROC SQL can still be passed to the DBMS.) However, if PROC SQL contains a WHERE clause, the WHERE clause *is* passed to the data source, if possible. Unless you specify DIRECT_SQL=NOWHERE, SAS attempts to pass all WHERE clauses to the data source.

If you specify more than one value for this option, separate the values with spaces and enclose the list of values in parentheses. For example, you could specify DIRECT_SQL=(NOFUNCTIONS, NOWHERE).

Examples

Example 1: Prevent a DBMS from Processing a Join

The following example prevents a join between two tables from being processed by the DBMS, by setting DIRECT_SQL=NOGENSQL. Instead, SAS processes the join.

```
proc sql;
   create view work.v as
      select tab1.deptno, dname from
         mydblib.table1 tab1,
         mydblib.table2 tab2
   where tab1.deptno=tab2.deptno
   using libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
      dsn=oradsn direct_sql=nogensql;
```

Example 2: Prevent a DBMS from Processing a SAS Function

The following example prevents a SAS function from being processed by the DBMS.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn direct_sql=nofunctions;
proc print data=mydblib.tab1;
   where lastname=soundex ('Paul');
run;
```

IGNORE_READ_ONLY_COLUMNS= LIBNAME Statement Option

Specifies whether to ignore or include columns whose data types are read-only when generating an SQL statement for inserts or updates.

Valid in:	LIBNAME statement
Default:	NO
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

IGNORE_READ_ONLY_COLUMNS=YES | NO

Syntax Description

YES
specifies to ignore columns whose data types are read-only when generating INSERT and UPDATE statements.

NO
specifies to include columns whose data types are read-only when generating INSERT and UPDATE statements

Details

Several databases include data types that can be read-only, such as the DB2 TIMESTAMP data type. Also, some databases have properties that allow certain data types to be read-only, such as the Microsoft SQL Server identity property.

When the IGNORE_READ_ONLY_COLUMNS= option is set to NO (the default), and a table contains a column that is read-only, an error is returned indicating that the data could not be modified for that column. For data sources that do not have read-only columns, such as SAS data sets, the option has no effect.

Comparisons

For the following example, a database that contains the table Products is created with two columns: ID and PRODUCT_NAME. The ID column is defined by a read-only data type and PRODUCT_NAME is a character column.

```
create table products (id int identity primary key, product_name varchar(40))
```

If you have a SAS data set that contains the name of your products, you can insert the data from the SAS data set into the Products table :

```
data work.products;
   id=1;
   product_name='screwdriver';
   output;
   id=2;
   product_name='hammer';
   output;
   id=3;
   product_name='saw';
   output;
   id=4;
   product_name='shovel';
   output;
run;
```

With IGNORE_READ_ONLY_COLUMNS=NO (the default), an error is returned by the DBMS because in this example, the ID column cannot be updated. However, if you set the option to YES and execute a PROC APPEND, the append succeeds, and the SQL statement that is generated does not contain the ID column.

```
libname x fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn ignore_read_only_columns=yes;
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc append base=x.products data=work.products;
run;
```

See Also

To apply this option to an individual table, use the IGNORE_ READ_ONLY_COLUMNS= data set option.

INSERT_SQL= LIBNAME Statement Option

Specifies the method that is used to insert rows into a data source.

Valid in:	LIBNAME statement
Default:	NO for SAS data set; all other data sources, the default is YES.
Supports:	All

Syntax

INSERT_SQL=YES | NO

Syntax Description

YES
specifies to use the data source's SQL insert method to insert new rows into a table.

NO
specifies to use an alternate (DBMS-specific) method to insert new rows into a table.

Details

SAS data sets generally have improved insert performance when INSERT_SQL=NO, which is the default for SAS data sets. Other data sources might have inferior insert performance (or might fail) unless INSERT_SQL=YES. You should experiment to determine the optimal setting for your situation.

See Also

To apply this option to an individual table, use the INSERT_SQL= data set option.

INSERTBUFF= LIBNAME Statement Option

Specifies the number of rows in a single insert operation.

Valid in:	LIBNAME statement
Default:	Data source-specific
Supports:	All

Syntax

INSERTBUFF=*positive-integer*

Syntax Description

positive-integer
specifies the number of rows to insert. SAS allows the maximum that is allowed by the data source.

Details

All data sources default to INSERT_SQL=YES except for SAS data sets. When INSERT_SQL=YES, INSERTBUFF= defaults to 1 and single row inserts are used. The optimal value for this option varies with factors such as network type and available memory. You might need to experiment with different values to determine the best value for your site.

The SAS application messages that indicate the success or failure of an insert operation only represent information for a single insert, even when multiple inserts are performed. Therefore, when you assign a value that is greater than INSERTBUFF=1, these messages might be incorrect.

If you specify the DBCOMMIT= option with a value that is less than the value of INSERTBUFF=, then DBCOMMIT= overrides INSERTBUFF=. If neither DBCOMMIT= nor INSERTBUFF= are specified, INSERTBUFF= defaults to a block size of 32K. SAS determines the number of rows by dividing 32K by the size of each row.

Note: When you insert by using the VIEWTABLE window or the FSVIEW or FSEDIT procedure, use INSERTBUFF=1 to prevent the DBMS driver from trying to insert multiple rows. These features do not support inserting more than one row at a time.

Additional driver-specific restrictions might apply.

See Also

To apply this option to an individual table, use the INSERTBUFF= data set option.

MULTI_DATASRC_OPT= LIBNAME Statement Option

Used in place of DBKEY to improve performance when processing a join between two data sources.

Valid in:	LIBNAME statement
Default:	NONE
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

MULTI_DATASRC_OPT=NONE |IN_CLAUSE

Syntax Description

NONE
turns off the functionality of the option.

IN_CLAUSE
specifies that an IN clause containing the values read from a smaller table will be used to retrieve the matching values in a larger table based on a key column designated in an equi-join.

Details

When processing a join between a SAS data set and a DBMS table, the SAS data set should be smaller than the DBMS table for optimal performance. However, in the event that the SAS data set is *larger* than that the DBMS table, the SAS data set will still be used in the IN clause.

When SAS is processing a join between two DBMS tables, SELECT COUNT (*) is issued to determine which table is smaller and if it qualifies for an IN clause.

Currently, the IN clause has a limit of 4,500 unique values.

Setting the DBKEY=data set option overrides MULTI_DATASRC_OPT=.

DIRECT_SQL= can impact this option as well. If DIRECT_SQL=NONE or NOWHERE, the IN clause cannot be built and passed to the DBMS, regardless of the value of MULTI_DATASRC_OPT=. These settings for DIRECT_SQL= prevent a WHERE clause from being passed.

Oracle Details: Oracle can handle an IN clause of only 1,000 values. Therefore, it divides larger IN clauses into multiple, smaller IN clauses. The results are combined into a single result set. For example, if an IN clause contained 4,000 values, Oracle will produce 4 IN clauses that each contain 1,000 values. A single result will be produced, as if all 4,000 values were processed as a whole.

Examples

Example 1: Build and Pass an IN Clause for a Join

The following example builds and passes an IN clause from the SAS table to the DBMS table, retrieving only the necessary data to process the join:

```
proc sql;
   create view work.v as
   select tab2.deptno, tab2.dname from
      work.sastable tab1, dblib.table2 tab2
   where tab1.deptno = tab2.deptno
   using libname dblib fedsvr server="d1234.us.company.com"
      port=2171 user=user1 pwd=pass1
      dsn=oradsn multi_datasrc_opt=in_clause;
quit;
```

Example 2: Prevent Build and Pass of an IN Clause for a Join

The following example prevents the building and passing of the IN clause to the DBMS, requiring all rows from the DBMS table to be brought into SAS for processing the join:

```
libname dblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn multi_datasrc_opt=none;
proc sql;
   select tab2.deptno, tab2.dname from
     work.table1 tab1,
     dblib.table2 tab2
   where tab1.deptno=tab2.deptno;
quit;
```

PRESERVE_COL_NAMES= LIBNAME Statement Option

Preserves spaces, special characters, use of reserved words, and case sensitivity in column names when you create tables.

Valid in:	LIBNAME statement (when you create tables)
Default:	NO
Supports:	All

Syntax

PRESERVE_COL_NAMES= NO | YES

Syntax Description

NO

specifies that column names that are used in table creation are derived from SAS column names by using the SAS column name normalization rules. (For more information see "VALIDVARNAME= System Option" on page 62 .) However, the data source applies its specific normalization rules to the SAS column names when creating the column names.

The use of N-Literals to create column names that use database keywords or special symbols other than the underscore character might be illegal when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

If you name a column with an SQL or FedSQL reserved word, an error is issued.

YES
specifies that column names that are used in table creation are passed to the data source with any special characters, reserved words, and the exact, case-sensitive spelling of the name preserved.

For Teradata, YES is the only supported value for this option.

Details

This option applies only when you create a new table. When you create a table, you assign the column names by using one of the following methods:

- To control the case of the column names, specify columns using the desired case and set PRESERVE_COL_NAMES=YES. If you use special symbols or blanks, you must set VALIDVARNAME= to ANY and use N-Literals.
- To enable the DBMS to normalize the column names according to its naming conventions, specify columns using any case and set PRESERVE_COLUMN_NAMES= NO.

Note: When you read from, insert rows into, or modify data in an existing DBMS table, SAS identifies the database column names by their spelling. Therefore, when the database column exists, the case of the column name does not matter.

Specify the alias PRESERVE_NAMES= if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement. Using this alias saves you some time when coding.

To use column names in your SAS program that are not valid SAS names, you must use one of the following techniques:

- Use the DQUOTE= option in PROC SQL and then reference your columns using double quotation marks. For example:

```
proc sql dquote=ansi;
   select "Total$Cost" from mydblib.mytable;
```

- Specify the global system option VALIDVARNAME=ANY and use name literals in the SAS language. For example:

```
proc print data=mydblib.mytable;
   format 'Total$Cost'n 22.2;
```

If you are creating a table in PROC SQL, you must also include the PRESERVE_COL_NAMES=YES option in your LIBNAME statement. For example:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuid=orauser dsnpwd=orapwd
    preserve_col_names=yes;
proc sql dquote=ansi;
   create table mydblib.mytable ("my$column" int);
```

For more information about using a DataFlux Federation Server reserved word as a column name, see “Reserved Language Keywords” on page 24.

PRESERVE_COL_NAMES= does not apply to the pass-through facility.

See Also

To apply this option to an individual table, use the PRESERVE_COL_NAMES= data set option.

PRESERVE_TAB_NAMES= LIBNAME Statement Option

Preserves spaces, special characters, reserved words, and case sensitivity in DBMS table names.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

PRESERVE_TAB_NAMES=NO | YES

Syntax Description

NO

specifies that when you create tables or refer to an existing table, the table names are derived from SAS member names by using SAS member name normalization. However, most databases apply DBMS-specific normalization rules to the SAS member names. Therefore, the table names are created or referenced in the database following the DBMS-specific normalization rules.

If you use an SQL or DataFlux FedSQL language reserved word as a table name, an error is issued.

When you use SAS to read a list of table names (for example, in the SAS Explorer window), the tables whose names do not conform to the SAS member name normalization rules do not appear in the output. In SAS line mode, the number of tables that do not display from PROC DATASETS due to this restriction is noted as a note. Here is an example:

```
Due to the PRESERVE_TAB_NAMES=NO LIBNAME option setting, 12
table(s) have not been displayed.
```

You will not get this warning when using SAS Explorer.

SAS Explorer displays DBMS table names in capitalized form when PRESERVE_TAB_NAMES=NO. This is now how the tables are represented in the DBMS.

YES

specifies that table names are read from and passed to the data source with any special characters, reserved words, and the exact, case-sensitive spelling of the name preserved. For Teradata, YES is the only supported value for this option. For SAS data sets, you need this option only if you are using an SQL reserved word as a table name. In all other cases, a valid SAS name must be used.

Details

To use table names in your SAS program that are not valid SAS names, use one of the following techniques:

- Use the PROC SQL option DQUOTE= and place double quotation marks around the table name. The libref must specify PRESERVE_TAB_NAMES=YES. For example:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1 dsn=oradsn;
proc sql dquote=ansi;
   select * from mydblib."my table";
```

- Use name literals in the SAS language. The libref must specify PRESERVE_TAB_NAMES=YES. For example:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn preserve_tab_names=yes;
proc print data=mydblib.'my table'n;
run;
```

- To use a DataFlux Federation Server reserved word with DataFlux FEDSQL language or PROC SQL, see "Reserved Language Keywords" on page 24.

Specify the alias PRESERVE_NAMES= to save time if you are specifying both PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= in your LIBNAME statement.

Oracle Details: Unless you specify PRESERVE_TAB_NAMES=YES, the table name that you enter for SCHEMA= or for the DBINDEX= data set option is converted to uppercase.

Example: Results of PRESERVE_TAB_NAMES=YES versus PRESERVE_TAB_NAMES=NO

If you use PROC DATASETS to read the table names in an Oracle database that contains three tables, My_Table, MY_TABLE, and MY TABLE. The results differ depending on the setting of PRESERVE_TAB_NAMES.

If the libref specifies PRESERVE_TAB_NAMES=NO, then the PROC DATASETS output is one table name, MY_TABLE. This is the only table name that is in Oracle normalized form (uppercase letters and a valid symbol, the underscore). My_Table is not displayed because it is not in Oracle normalized form. MY TABLE is not displayed because it is not in SAS member normalized form (the embedded space is a nonstandard SAS character).

If the libref specifies PRESERVE_TAB_NAMES=YES, then the PROC DATASETS output includes all three table names, My_Table, MY_TABLE, and MY TABLE.

See Also

"PRESERVE_COL_NAMES= LIBNAME Statement Option" on page 83

QUALIFIER= LIBNAME Statement Option

Identifies database objects, such as tables, by using a qualifier.

Valid in:	LIBNAME statement
Alias:	CATALOG=
Default:	none
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

QUALIFIER=*qualifier-name*

Details

If this option is omitted, the default qualifier name, if any, is used for the data source. QUALIFIER= can be used for any data source that allows three-part identifier names: *qualifier.schema.object*.

MySQL Details: The MySQL driver does not support three-part identifier names, so a two-part name is used (such as *qualifier.object*).

Example: Specifying Use of a Three-Part Name

In the following LIBNAME statement, the QUALIFIER= option causes any reference in SAS to MYDBLIB.EMPLOYEE to be interpreted by a DB2 database as MYDEPT.SCOTT.EMPLOYEE.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn qualifier=mydept schema=scott;
```

See Also

To apply this option to an individual table, use the QUALIFIER= data set option.

READBUFF= LIBNAME Statement Option

Specifies the number of rows of data to read into the buffer.

Valid in:	LIBNAME statement
Default:	DBMS-specific
Supports:	All

Syntax

READBUFF=*integer*

Syntax Description

integer
is the positive number of rows to hold in memory. SAS allows the maximum number that is allowed by the database.

Details

This option improves performance by specifying a number of rows that can be held in memory for input into SAS. Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for READBUFF= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.

When READBUFF=1, only one row is retrieved at a time. The higher the value for READBUFF=, the more rows are retrieved in one fetch operation. If READBUFF= is not set, certain operations such as SQL SELECT statements cause the number of rows to be set to the client's default value, which for the engine is one row at a time. Setting READBUFF=128 can significantly boost the application's performance.

If you do not specify a value with this option, the engine calculates the buffer size based on the row length of your data (with a minimum of 10) and retrieves the number of rows in each fetch operation.

See Also

To apply this option to an individual table, use the READBUFF= data set option.

READ_ISOLATION_LEVEL= LIBNAME Statement Option

Defines the degree of isolation of the current application process from other concurrently running application processes.

Valid in:	LIBNAME statement
Default:	DBMS-specific
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

READ_ISOLATION_LEVEL=*DBMS-specific value*

Syntax Description

See the reference documentation for your DBMS for additional details.

Details

The degree of isolation defines

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

The ODBC and DB2 drivers ignore this option if READ_LOCK_TYPE= is not set to ROW.

See Also

To apply this option to an individual table, use the READ_ISOLATION_LEVEL= data set option.

READ_LOCK_TYPE= LIBNAME Statement Option

Specifies how data in a DBMS table is locked during a read transaction.

Valid in:	LIBNAME statement

Default:	data source-specific
Supports:	DB2 UNIX/PC, ODBC, Oracle

Syntax

READ_LOCK_TYPE=ROW |NOLOCK

Syntax Description

ROW
locks a row if any of its columns are accessed. If you are accessing a DB2 or ODBC database, READ_LOCK_TYPE=ROW indicates that locking is based on the READ_ISOLATION_LEVEL= option. (This value is valid for connecting to DB2, ODBC, or Oracle databases.)

NOLOCK
does not lock the DBMS table, pages, or rows during a read transaction. (This value is valid for connecting to the Microsoft SQL Server via the ODBC table driver.)

Details

If you omit READ_LOCK_TYPE=, the default is the data source's default action. You can set a lock for one data source table by using the data set option or for a group of DBMS tables by using the LIBNAME option.

Example: Specify a Row-Level Lock for Read and Update

In the following example, the LIBNAME options specify that row-level locking is used when data is read or updated:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1 dsn=oradsn
   read_lock_type=row update_lock_type=row;
```

See Also

To apply this option to an individual table, use the READ_LOCK_TYPE= data set option.

REREAD_EXPOSURE= LIBNAME Statement Option

Specifies whether the interface behaves like a random access engine for the scope of the LIBNAME statement.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

REREAD_EXPOSURE=NO | YES

Syntax Description

NO
: specifies that the engine behaves as a sequential engine with limited random access. This means that your data is protected by the normal data protection that SAS provides.

YES
: specifies that the engine behaves like a random access engine when rereading a row so that you cannot guarantee that the same row is returned. For example, if you read row 5 and someone else deletes it, then the next time you read row 5, you will read a different row. You have the potential for data integrity exposures within the scope of your SAS session.

Details

CAUTION:
Using REREAD_EXPOSURE= could cause data integrity exposures.

Oracle Details: To avoid data integrity problems, it is advisable to set UPDATE_LOCK_TYPE=TABLE if you set REREAD_EXPOSURE=YES.

ODBC Details: To avoid data integrity problems, it is advisable to set UPDATE_ISOLATION_LEVEL=S (serializable) if you set REREAD_EXPOSURE=YES.

SCHEMA= LIBNAME Statement Option

Enables you to read database objects, such as tables, in the specified DBMS schema.

Valid in:	LIBNAME statement
Default:	data source-specific
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

SCHEMA=*schema-name*

Details

If this option is omitted, you connect to the default schema for your data source.

The values for SCHEMA= can be case-sensitive, depending on the data source, so use care when you specify this option. You should set PRESERVE_TAB_NAMES=YES when the value for SCHEMA= contains mixed case characters.

Oracle Details: Specify a schema name to be used when referring to database objects. SAS can access another user's database objects by using a specified schema name. If PRESERVE_TAB_NAMES=NO, SAS converts the SCHEMA= value to uppercase because all values in the Oracle data dictionary are uppercase unless quoted.

Teradata Details: If you omit this option, a libref points to your default Teradata database, which often has the same name as your user name. You can use this option to point to a different database. This option enables you to view or modify a different user's DBMS tables or views if you have the required Teradata privileges. (For example, to read another user's tables, you must have the Teradata privilege SELECT for that user's tables.) The Teradata alias for SCHEMA= is DATABASE=. For more information about changing the default database, see the DATABASE statement in your Teradata documentation.

Examples

Example 1: Use SCHEMA to Access a Table in Your Schema

In the following example, SCHEMA= causes any reference in SAS to MYDB.EMPLOYEE to be interpreted by DB2 as SCOTT.EMPLOYEE.

```
libname mydb fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn schema=SCOTT;
```

Example 2: Use SCHEMA to Access Another User's Schema

To access an Oracle object in another schema, use the SCHEMA= option as in the following example. The schema name is typically a person's user name or ID.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn schema=john;
```

Example 3: Use SCHEMA to Access a DBMS Schema

In the following example, the Oracle table SCHEDULE resides in the AIRPORTS schema. To access this table using the PRINT procedure with the libref CARGO, you specify the DBMS schema in the SCHEMA= option. Then you specify CARGO.SCHEDULE in the procedure's DATA statement.

```
libname cargo fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn schema=airports;
proc print data=cargo.schedule;
run;
```

Example 4: Use SCHEMA to Access a Teradata Database

In the following Teradata example, the user ID TESTUSER prints the EMP table, which is located in the OTHERUSER database.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=teradsn schema=otheruser;
proc print data=mydblib.emp;
run;
```

See Also

To apply this option to an individual table, use the SCHEMA= data set option.

SPOOL= LIBNAME Statement Option

Specifies whether SAS creates a utility spool file during read transactions that read data more than once.

Valid in:	LIBNAME statement
Default:	YES
Supports:	All

Syntax

SPOOL=YES | NO | DBMS

Syntax Description

YES
: specifies that SAS creates a utility spool file into which it writes the rows that are read the first time. For subsequent passes through the data, the rows are read from the utility spool file rather than being re-read from the data source table. This guarantees that the row set is the same for every pass through the data.

NO
: specifies that the required rows for all passes of the data are read from the data source table. No spool file is written. There is no guarantee that the row set is the same for each pass through the data.

DBMS
: is valid for Oracle only. The required rows for all passes of the data are read from the DBMS table but additional enforcements are made on the DBMS server side to ensure that the row set is the same for every pass through the data. This setting causes the Oracle driver to satisfy the two-pass requirement by starting a read-only transaction. SPOOL=YES and SPOOL=DBMS have comparable performance results for Oracle; however, SPOOL=DBMS does not use any disk space. When SPOOL is set to DBMS, the CONNECTION option must be set to UNIQUE. If not, an error occurs.

Details

In some cases, SAS processes data in more than one pass through the same set of rows. Spooling is the process of writing rows that have been retrieved during the first pass of a data read to a spool file. In the second pass, rows can be reread without performing I/O to the DBMS a second time. When data must be read more than once, spooling improves performance. Spooling also guarantees that the data remains the same between passes, as most data sources do not support member-level locking.

Teradata Details: SPOOL=NO requires SAS to issue identical SELECT statements to Teradata twice. Additionally, because the Teradata table can be modified between passes, SPOOL=NO can cause data integrity problems. Use SPOOL=NO with discretion.

MySQL Details: Do not use SPOOL=NO with the MySQL driver.

SQL_FUNCTIONS= LIBNAME Statement Option

Specifies that the functions that match those supported by SAS should be passed to the data source.

Valid in:	LIBNAME statement
Default:	NONE
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

SQL_FUNCTIONS=ALL

Syntax Description

ALL
specifies that functions that match those that are supported by SAS should be passed to the data source.

Details

DB2 UNIX/PC, ODBC Details: When SQL_FUNCTIONS= is set to ALL, only the functions that are supported by the database table drivers are passed. Only a fraction of the functions might be available.

DATE	TODAY	QTR	COMPRESS	SUBSTR
DATEPART	DAY	SECOND	INDEX	TRANWRD
DATETIME	HOUR	WEEKDAY	LENGTH	TRIMN
TIME	MINUTE	YEAR	REPEAT	MOD
TIMEPART	MONTH	BYTE	SOUNDEX	

Use of this option can cause unexpected results, especially if used for NULL processing and date/time/timestamp handling. For example, the following SAS code executed without SQL_FUNCTIONS= enabled returns the SAS date 15308:

```
proc sql;
   select distinct DATE () from x.test;
quit;
```

However, the same code with SQL_FUNCTIONS=ALL, returns 2001-1-29, which is an ODBC date format. Care should be exercised when using this option.

STRINGDATES= LIBNAME Statement Option

Specifies whether to read date and time values from the data source as character strings or as numeric date values.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

STRINGDATES=YES | NO

Syntax Description

YES
specifies that SAS reads date and time values as character strings.

NO
specifies that SAS reads date and time values as numeric date values.

Details

NO is also used for Version 6 compatibility. This is the default.

UPDATE_ISOLATION_LEVEL= LIBNAME Statement Option

Specifies the degree of isolation of the current application process from other concurrently running application processes.

Valid in:	LIBNAME statement
Default:	data source-specific
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

UPDATE_ISOLATION_LEVEL= *data source-specific-value*

Syntax Description

The values for this option are DBMS-specific.

Details

The degree of isolation defines the following

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

For ODBC and DB2 under UNIX and PC hosts, this option is ignored if UPDATE_LOCK_TYPE= is not set to ROW.

See Also

To apply this option to an individual table, use the UPDATE_ISOLATION_LEVEL= data set option.

UPDATE_LOCK_TYPE= LIBNAME Statement Option

Specifies how data is locked during an update transaction.

Valid in:	LIBNAME statement
Default:	data source-specific
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

UPDATE_LOCK_TYPE=ROW | TABLE | NOLOCK

Syntax Description

ROW
: locks a row if any of its columns are going to be updated. (This value is valid for all the data sources that are listed for this option.)

TABLE
: locks the entire table. (This value is valid for the SQL Server by using the ODBC table driver.)

NOLOCK
: does not lock the table, page, or any rows when reading them for update. (This value is valid for the Oracle data source.)

Details

You can set a lock for one data source table by using the data set option if available, or for a group of tables by using the LIBNAME option.

See the reference documentation for your DBMS for additional details.

See Also

To apply this option to an individual table, use the UPDATE_LOCK_TYPE= data set option.

UTILCONN_TRANSIENT= LIBNAME Statement Option

Enables utility connections to be maintained or dropped, as needed.

Valid in:	LIBNAME statement
Default:	NO
Supports:	All

Syntax

UTILCONN_TRANSIENT=NO | YES

Syntax Description

NO
: specifies that a utility connection is maintained for the lifetime of the libref.

YES
: specifies that a utility connection is automatically dropped as soon as it is no longer in use.

Details

For data sources that can lock system resources as a result of operations such as DELETE or RENAME, or as a result of queries on system tables or table indices, a utility connection is used. The utility connection prevents the COMMIT statements that are issued to unlock system resources from being submitted on the same connection that is being used for table processing. Keeping the COMMIT statements off of the table processing connection alleviates the problems that they can cause, such as invalidating cursors and committing pending updates on the tables being processed.

Since a utility connection exists for each LIBNAME statement, the number of connections to a data source can become large as multiple librefs are assigned across multiple SAS sessions. Setting UTILCONN_TRANSIENT= to YES keeps these connections from existing when they are not being used, thus reducing the number of current connections to the data source at any point in time.

UTILCONN_TRANSIENT= is ignored by data sources that do not support utility connections.

Chapter 13

Data Set Options for FEDSVR Engine

FEDSVR LIBNAME Engine Data Set Options

About the FEDSVR LIBNAME Engine Data Set Options

Data set options specify actions that apply only to the table on which they are specified and remain in effect for the duration of the DATA step or SAS procedure. For example, data set options let you perform such operations as renaming columns, specifying the last row to process, and assigning a password for a table. For more information about data set options, see *SAS Language Reference: Concepts*.

Restrictions

Only a subset of the data set options provided by SAS are supported for the FEDSVR LIBNAME engine.

The availability and behavior of the data set options are data source specific.

Syntax

Specify a data set option in parentheses after a table name. To specify several data set options, separate them with spaces.

(*option-1=value* <...*option-n=value*>)

These examples show data set options in SAS statements:

```
data salary (encrypt=yes read=green);

proc print data=salary (read=green);
```

How Data Set Options Interact with Other Types of Options

Many types of options share the same name and have the same function. For example, you can request to compress a SAS data set by specifying COMPRESS= as a connection string option and as a LIBNAME statement option.

When more than one type of option with the same function is specified, the software honors the following order of precedence:

1. data set option
2. LIBNAME statement option
3. data source connection string option
4. system option

That is, a data set option overrides a LIBNAME statement option, which overrides a data source connection string option.

Dictionary

ALTER= Data Set Option

Assigns an ALTER password to a SAS data set that prevents users from replacing or deleting the file, and enables access to a read- and write-protected file.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Supports:	SAS data set

Syntax

ALTER=*alter-password*

Syntax Description

alter-password
must be a valid SAS name.

Details

The ALTER= option is supported only for unsecured DSNs. If you try to assign a password to a SAS data set that is protected by Federation Server authorization, the Federation Server will return an error.

BL_LOAD_REPLACE= Data Set Option

Specifies whether DB2 will append or replace rows during bulk loading.

Valid in:	DATA and PROC steps
Default:	NO
Supports:	DB2 UNIX/PC

Syntax

BL_LOAD_REPLACE=NO | YES

Syntax Description

NO
appends new rows of data to the DB2 table.

YES
replaces the existing data in the table.

Details

This option must be specified using the BULKOPTS= container option. The BULKLOAD= data set option must be set to YES.

```
bulkload=yes; bulkopts=(bl_load_replace=yes);
```

BL_LOG= Data Set Option

Identifies a log file that will contain information such as statistics and error information for a bulk load.

Valid in:	DATA and PROC steps
Supports:	DB2 UNIX/PC

Syntax

BL_LOG=*path-and-log-filename*

Syntax Description

path-and-log-filename
is a file to which information about the loading process is written.

Details

When the DB2 bulk-load facility is invoked, it creates a log file. The BL_ prefix distinguishes this log file from the one created by the SAS log. If BL_LOG= is specified with the same path and filename as an existing log, the new log replaces the existing log. If this option is not specified, the log that is created during the load operation is deleted immediately.

This option must be specified with the BULKOPTS= container option. The BULKLOAD= data set option must be set to YES.

```
bulkload=yes; bulkopts=(bl_log='c:\temp\bulkload.log');
```

BL_OPTIONS= Data Set Option

Passes options to the DB2 bulk-load facility, affecting how it loads and processes data.

Valid in:	DATA and PROC steps
Supports:	DB2 UNIX/PC

Syntax

BL_OPTIONS='<*option..., option*>'

Syntax Description

option
specifies an option from the available DB2 options. See details below.

Details

BL_OPTIONS= enables you to pass options to the DB2 bulk-load facility when it is invoked, thereby affecting how data is loaded and processed. You must separate multiple options with commas and enclose the entire string of options in single quotation marks.

This option passes DB2 file type modifiers to DB2 LOAD or IMPORT commands to affect how data is loaded and processed. *Not all DB2 file type modifiers are appropriate for all situations*. You can specify one or more DB2 file type modifiers with .IXF files. For a list of file type modifiers, see the description of the LOAD and IMPORT utilities in the *IBM DB2 Universal Database Data Movement Utilities Guide and Reference*.

This option must be specified using the BULKOPTS= container option. The BULKLOAD= data set option must be set to YES.

```
bulkload=yes; bulkopts=(bl_options 'errors 999, load=2000');
```

BUFNO= Data Set Option

Specifies the number of buffers to be allocated for processing a SAS data set.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Supports:	SAS data set

Syntax

BUFNO=*n* | *n*K | *hex*X | MIN | MAX

Syntax Description

***n* | *n*K**
: specifies the number of buffers in multiples of 1 (bytes); 1,024 (kilobytes). For example, a value of `8` specifies 8 buffers, and a value of `1k` specifies 1024 buffers.

hex
: specifies the number of buffers as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value `2dx` sets the number of buffers to 45 buffers.

MIN
: sets the minimum number of buffers to 0, which causes SAS to use the minimum optimal value for the operating environment. This is the default.

MAX
: sets the number of buffers to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 2^{31}-1, or approximately 2 billion.

Details

The buffer number is not a permanent attribute of the data set; it is valid only for the current SAS session or job.

BUFNO= applies to a SAS data set that is opened for input, output, or update.

A larger number of buffers can speed up execution time by limiting the number of input and output (I/O) operations that are required for a particular SAS data set. However, the improvement in execution time comes at the expense of increased memory consumption.

To reduce I/O operations on a small data set as well as speed execution time, allocate one buffer for each page of data to be processed. This technique is most effective if you read the same rows several times during processing.

Comparisons

If the BUFNO= data set option is not specified, the value of the BUFNO= system option is used. If both are specified in the same SAS session, the value specified for the BUFNO= data set option overrides the value specified for the BUFNO= system option.

BUFSIZE= Data Set Option

Specifies the size of a permanent buffer page for an output SAS data set.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Restriction:	Use with output data sets only.
Supports:	SAS data set

Syntax

BUFSIZE=*n* | *n*K | *n*M | *n*G | *hex*X | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G**

specifies the page size in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, a value of **`8`** specifies a page size of 8 bytes, and a value of **`4k`** specifies a page size of 4096 bytes.

Note: If neither the BUFSIZE= system option nor the data set option are set, the default value is 0. This causes SAS to use the minimum optimal page size for the operating environment. The BUFSIZE= system option will be used in either of the following scenarios:

- if the BUFSIZE= data set option is not set
- if the BUFSIZE= data set option is set to 0

The system option BUFSIZE=0 should be set to reset the page size to the default value for the operating environment.

***hex*X**

specifies the page size as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value **`2dx`** sets the page size to 45 bytes.

MAX

sets the page size to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 2^{31}-1, or approximately 2 billion bytes.

Details

The page size is the amount of data that can be transferred for a single I/O operation to one buffer. The page size is a permanent attribute of the data set and is used when the data set is processed.

A larger page size can speed up execution time by reducing the number of times SAS has to read from or write to the storage medium. However, the improvement in execution time comes at the cost of increased memory consumption.

To change the page size, use a DATA step to copy the data set and either specify a new page or use the SAS default. To reset the page size to the default value in your operating environment, set the BUFSIZE= system option to 0 and set the data set option to 0 or leave it blank.

Note: If you use the COPY procedure to copy a data set to another library that is allocated with a different engine, the specified page size of the data set is not retained.

Operating Environment Information

The default value for BUFSIZE= is determined by your operating environment and is set to optimize sequential access. To improve performance for direct (random) access, you should change the value for BUFSIZE=. For the default setting and possible settings for direct access, see the BUFSIZE= data set option in the SAS documentation for your operating environment.

Comparisons

If the BUFSIZE= data set option is not specified, then the value of the BUFSIZE= system option is used. If both are specified in the same SAS session, the BUFSIZE= data set option overrides the value specified for the BUFSIZE= system option, unless the

BUFSIZE= data set option is set to 0. The BUFSIZE= system option must be set to 0 to reset the page size to the default value for the operating environment.

BULKLOAD= Data Set Option

Loads rows of data as one unit.

Valid in:	DATA and PROC steps
Default:	NO
Supports:	DB2 UNIX/PC, Oracle

Syntax

BULKLOAD= YES | NO

Syntax Description

YES
calls a DBMS-specific bulk-load facility in order to insert or append rows to a DBMS table.

NO
uses the driver to insert or append data to a DBMS table.

Details

Using BULKLOAD=YES is the fastest way to insert rows into a DBMS table.

When BULKLOAD=YES, the first error encountered causes the remaining rows (including the erroneous row) in the buffer to be rejected. No other errors within the same buffer will be detected. In addition, all rows before the error are committed, even if DBCOMMIT= is set larger than the number of the erroneous row.

For details, see the bulk loading topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

See Also

To assign this option to a group of tables or views, use the BULKLOAD= LIBNAME option.

BULKOPTS= Data Set Option

Container option for BL_LOAD_REPLACE=, BL_LOG=, and BL_OPTIONS= data set options.

Valid in:	DATA and PROC steps
Category:	Bulk Loading
Requirement:	Must follow BULKLOAD=YES
Supports:	DB2 UNIX/PC

Syntax

BULKOPTS=(*option1=value optionn=value*)

Syntax Description

BL_LOAD_REPLACE=NO|YES
: specifies whether the CLI LOAD interface replaces the existing data in the data set. For more information, see "BL_LOAD_REPLACE= Data Set Option" on page 100 .

BL_LOG=*pathname*
: specifies a file to which information about the loading process is written. For more information, see "BL_LOG= Data Set Option" on page 100 .

BL_OPTION=*option*
: specifies a valid DB2 option. For more information, see "BL_OPTIONS= Data Set Option" on page 101 .

Details

The BULKOPTS= option is a container option that is required to specify BL_LOAD_REPLACE=, BL_LOG=, and BL_OPTION= data set options. To specify the BULKOPTS= option, you must first specify BULKLOAD=YES. For more information, see "BULKLOAD= Data Set Option" on page 104 .

Example: Specifying BULKOPTS=

The following example uses BULKLOAD=YES and BULKOPTS= options:

```
BULKLOAD=YES;
    BULKOPTS=(BL_LOG='c:\temp\bulkload.log"  BL_LOAD_REPLACE=yes
                                BL_OPTIONS='ERRORS=999, LOAD=2000');
```

COMPRESS= Data Set Option

Specifies the compression of rows in an output table.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Restriction:	Use with output data sets only.
Supports:	SAS data set

Syntax

COMPRESS=NO | YES | CHAR | BINARY

Syntax Description

NO
: specifies that the rows in a newly created data set are uncompressed (fixed-length records).

YES | CHAR
specifies that the rows in a newly created data set are compressed (variable-length records) by SAS using RLE (Run Length Encoding). RLE compresses rows by reducing repeated consecutive characters (including blanks) to two-byte or three-byte representations.

Alias: ON

Tip: Use this compression algorithm for character data.

BINARY
specifies that the rows in a newly created data set are compressed (variable-length records) by SAS using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file.

Tip: This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (numeric columns). Because the compression function operates on a single record at a time, the record length needs to be several hundred bytes or larger for effective compression.

Details

Compressing a file is a process that reduces the number of bytes required to represent each row. Advantages of compressing a file include reduced storage requirements for the file and fewer I/O operations necessary to read or write to the data during processing. However, more CPU resources are required to read a compressed file (because of the overhead of uncompressing each row). There are also situations where the resulting file size might increase rather than decrease.

Use the COMPRESS= data set option to compress an individual file. Specify the option for output data sets only, that is, data sets named in the DATA statement of a DATA step or in the OUT= option of a SAS procedure.

After a file is compressed, the setting is a permanent attribute of the file, which means that to change the setting, you must re-create the file. That is, to uncompress a file, specify COMPRESS=NO for a DATA step that copies the compressed file.

Comparisons

The COMPRESS= data set option overrides the COMPRESS= option in the LIBNAME statement, the COMPRESS= connection string option, and the COMPRESS= system option.

When you create a compressed SAS data set, you can also specify REUSE=YES (as a data set option or connection option) in order to track and reuse space. With REUSE=YES, new rows are inserted in space freed when other rows are updated or deleted. When the default REUSE=NO is in effect, new rows are appended to the existing file.

DBCOMMIT= Data Set Option

Causes an automatic COMMIT (a permanent writing of data to the DBMS) after a specified number of rows have been processed.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

DBCOMMIT=*n*

Syntax Description

n
is an integer greater than or equal to 0.

Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows processed includes rows that are not processed successfully. When DBCOMMIT=0, a commit is issued only once (after the procedure or DATA step completes).

If the DBCOMMIT= option is explicitly set, any update that has a WHERE clause fails.

SAS data sets cannot be rolled back, so for these data sources, this option has no effect. However, if explicitly set, the LIBNAME engine will still fail any update that has a WHERE clause even though the value specified on this option has no effect.

Teradata Details: The Teradata driver alias for this option is CHECKPOINT. See the Fastload capability description in the data source reference documentation for Teradata for the default behavior of this option.

Example: Specify the Number of Rows to Process

In the following example, a commit is issued after every 10 rows are processed:

```
data oracle.dept (dbcommit=10);
   set myoralib.staff;
run;
```

DBCONDITION= Data Set Option

Specifies criteria for subsetting and ordering DBMS data.

Valid in:	DATA and PROC steps
Supports:	All

Syntax

DBCONDITION=*"DBMS-SQL-query-clause"*

Syntax Description

DBMS-SQL-query-clause
is a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

Details

This option enables you to specify selection criteria in the form of DBMS-specific SQL query clauses, which are passed directly to the DBMS for processing. When selection criteria are passed directly to the DBMS for processing, performance is often enhanced. The DBMS checks the criteria for syntax errors when it receives the SQL query.

The DBKEY= and DBINDEX= options are ignored when you use DBCONDITION=. DBCONDITION= is ignored if it specifies ORDER BY and you also use a BY statement.

Example: Return Only Condition-Specific Rows

In the following example, the function that is passed to the DBMS with the DBCONDITION= option causes the DBMS to return to SAS only the rows that satisfy the condition.

```
proc sql;
   create view smithnames as
      select lastname from myoralib.employees
             (dbcondition="where soundex(lastname) = soundex('SMYTHE')" )
              using libname myoralib oracle user=testuser pw=testpass path=dbmssrv;
select lastname from smithnames;
```

DBCREATE_TABLE_OPTS= Data Set Option

Specifies syntax to be added to the CREATE TABLE statement.

Valid in:	DATA and PROC steps
Supports:	MySQL

Syntax

DBCREATE_TABLE_OPTS=*'SQL-clauses'*

Syntax Description

MySQL-specific-SQL-clauses
are one or more clauses that can be appended to the end of an SQL CREATE TABLE statement.

Details

This option enables you to add clauses to the end of the SQL CREATE TABLE statement. The engine passes the SQL CREATE TABLE statement and its clauses to MySQL, which executes the statement and creates the table. This option applies only when you are creating a table by specifying a libref associated with data.

DBFORCE= Data Set Option

Specifies whether to force the truncation of data during insert processing.

Valid in:	DATA and PROC steps
Default:	NO
Supports:	All

Syntax

DBFORCE=YES | NO

Syntax Description

YES
specifies that the rows which contain data values that exceed the length of the DBMS column are inserted, and the data values are truncated to fit the DBMS column length.

NO
specifies that the rows which contain data values that exceed the DBMS column length are not inserted.

Details

This option determines how the driver handles rows that contain data values that exceed the length of the DBMS column.

The SAS data set option FORCE= overrides this option when it is used with PROC APPEND or the PROC SQL UPDATE statement. The PROC SQL UPDATE statement does not provide a warning before truncating the data.

Example: Truncate Data during Insert Processing

In the following example, two librefs are associated with Oracle databases. The default databases and schemas are used and therefore are not specified. In the DATA step, MYDBLIB.DEPT is created from the Oracle data referenced by MYORALIB.STAFF. The LASTNAME column is a character column of length 20 in MYORALIB.STAFF. During the creation of MYDBLIB.DEPT, the LASTNAME column is stored as a column of type character and length 10 by using DBFORCE=YES.

```
libname myoralib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn1 dsnuser=orauser dsnpwd=orapwd;
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn2 dsnuser=orauser dsnpwd=orapwd;
data mydblib.dept (dbtype=(lastname='char(10)') dbforce=yes);
   set myoralib.staff;
run;
```

DBGEN_NAME= Data Set Option

Specifies how SAS renames columns automatically when they contain characters that SAS does not allow.

Valid in:	DATA and PROC steps
Default:	DBMS
Supports:	All

Syntax

DBGEN_NAME=DBMS | SAS

Syntax Description

DBMS
: specifies that disallowed characters are converted to underscores.

SAS
: specifies that DBMS columns that contain disallowed characters are converted into valid SAS column names, using the format _COL*n*, where *n* is the column number (starting with zero). If a name is converted to a name that already exists, a sequence number is appended to the end of the new name.

Details

SAS retains column names when reading data from DBMS tables, unless a column name contains characters that SAS does not allow, such as $ or @. SAS allows alphanumeric characters and the underscore (_).

This option is intended primarily for National Language Support, notably the conversion of kanji to English characters because the English characters converted from kanji are often those that are not allowed in SAS.

Note: The various interfaces handle name collisions differently in SAS Version 6. Some interfaces appended to the end of the name; other interfaces replaced the last characters in the name. Some interfaces used a single sequence number, other interfaces used unique counters. If you specify VALIDVARNAME=V6, name collisions are handled the same as they were in SAS Version 6.

Example:

If you specify DBGEN_NAME=SAS, a DBMS column named DEPT$AMT is renamed to _COL*n* where *n* is the column number.

If you specify DBGEN_NAME=DBMS,a DBMS column named DEPT$AMT is renamed to DEPT_AMT.

DBINDEX= Data Set Option

Detects and verifies that indexes exist on a DBMS table. If they do exist and are of the correct type, a join query that is passed to the DBMS might improve performance.

Valid in:	DATA and PROC steps
Default:	DBMS specific
Supports:	All

Syntax

DBINDEX=YES | NO | <'>*index-name*<'>

Syntax Description

YES
: triggers the interface to search for all indexes on a table and return them to SAS for evaluation. If a usable index is found, then the join WHERE clause is passed to the DBMS for processing. A usable index is expected to have at least the same attributes as the join column.

NO
no automated index search is performed.

index-name
verifies the index name that is specified for the index columns on the DBMS table. This requires the same type of call as when DBINDEX=YES is used.

Details

When processing a join that involves a large DBMS table and a relatively small SAS data set, you might be able to use DBINDEX= to improve performance.

CAUTION:
Improper use of this option can degrade performance.

Queries must be issued to the necessary DBMS control or system tables to extract index information about a specific table or validate the index that you specified.

Examples

Example 1: Use DBINDEX= in a LIBNAME Statement

The following SAS data set is used in these examples:

```
data s1;
   a=1; y='aaaaa'; output;
   a=2; y='bbbbb'; output;
   a=5; y='ccccc'; output;
run;
```

The following example demonstrates the use of DBINDEX= in the LIBNAME statement:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd
   dbindex=yes;
proc sql;
   select * from s1 aa, x.dbtab bb where aa.a=bb.a;
   select * from s1 aa, mydblib.dbtab bb where aa.a=bb.a;
```

The DBINDEX= values for table dbtab are retrieved from the DBMS and compared with the join values. In this case, a match was found. Therefore, the join is passed down to the DBMS using the index. If the index `a` was not found, the join would take place in SAS.

Example 2: Use DBINDEX= in a SAS DATA Step

The following example demonstrates the use of DBINDEX= in the SAS DATA step:

```
data a;
   set s1;
   set x.dbtab(dbindex=yes) key=a;
   set mydblib.dbtab(dbindex=yes) key=a;
run;
```

The key is validated against the list from the DBMS. If `a` is an index, then a pass down occurs. Otherwise, the join takes place in SAS.

Example 3: Use DBINDEX= in PROC SQL

The following example demonstrates the use of DBINDEX= in PROC SQL:

```
proc sql;
   select * from s1 aa, x.dbtab(dbindex=yes) bb where aa.a=bb.a;
   select * from s1 aa, mylib.dbtab(dbindex=yes) bb where aa.a=bb.a;
   /*or*/
   select * from s1 aa, x.dbtab(dbindex=a) bb where aa.a=bb.a;
   select * from s1 aa, mylib.dbtab(dbindex=a) bb where aa.a=bb.a;
```

DBKEY= Data Set Option

Specifies a key column to optimize DBMS retrieval. Can improve performance when you are processing a join that involves a large DBMS table and a small SAS data set or DBMS table.

Valid in: DATA and PROC steps

Supports: All

Syntax

DBKEY=(<'> *column-1*<'> <… <'>*column-n*<'>>)

Syntax Description

column

used by SAS to build an internal WHERE clause to search for matches in the DBMS table based on the key column. For example:

```
select * from sas.a, dbms.b(dbkey=x) where a.x=b.x;
```

In this example, DBKEY= specifies column **x**, which matches the key column designated in the WHERE clause. However, if the DBKEY= column does not match the key column in the WHERE clause, then DBKEY= is not used.

Details

When processing a join that involves a large DBMS table and a relatively small SAS data set, you might be able to use DBKEY= to improve performance.

When you specify DBKEY=, it is *strongly* recommended that an index exists for the key column in the underlying DBMS table. Performance can be severely degraded without an index.

CAUTION:

Improper use of this option can decrease performance.

Examples

Example 1: Using DBKEY= with MODIFY=

The following example uses DBKEY= with the MODIFY statement in a DATA step:

```
libname invty fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn dsnuser=db2user dsnpwd=db2pwd;
data invty.stock;
   set addinv;
   modify invty.stock(dbkey=partno) key=dbkey;
   INSTOCK=instock+nwstock;
   RECDATE=today();
   if _iorc_=0 then replace;
run;
```

Example 2: Using More than One DBKEY= Value

To use more than one value for DBKEY=, you must include the second value as a join in the WHERE clause. In the following example, the PROC SQL brings the entire DBMS table into SAS and then proceeds with processing:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc sql;
create table work.barbkey as
select keyvalues.empid, employees.hiredate, employees.jobcode
     from mydblib.employees(dbkey=(empid jobcode))
     inner join work.keyvalues on employees.empid = keyvalues.empid;
         quit;
```

DBLABEL= Data Set Option

Specifies whether to use SAS column labels or SAS column names as the DBMS column names during output processing.

Valid in:	DATA and PROC steps
Default:	NO
Supports:	All

Syntax

DBLABEL=YES | NO

Syntax Description

YES
specifies that SAS column *labels* are used as DBMS column names during output processing.

NO
specifies that SAS column *names* are used as DBMS column names.

Details

This option is valid only for creating DBMS tables.

Example: Specify a Variable Label

In the following example, a SAS data set, NEW, is created with one column C1. This column is assigned a label of DEPTNUM. In the second DATA step, the MYDBLIB.MYDEPT table is created by using DEPTNUM as the DBMS column name. Setting DBLABEL=YES enables the label to be used as the column name.

```
data new;
   label c1='deptnum';
   c1=001;
run;
data mydblib.mydept(dblabel=yes);
   set new;
run;
proc print data=mydblib.mydept;
run;
```

DBMASTER= Data Set Option

Designates which table is the larger table when you are processing a join that involves tables from two different types of databases.

Valid in:	DATA and PROC steps
Supports:	All

Syntax

DBMASTER=*YES*

Syntax Description

YES
designates which of two tables references in a join operation is the larger table.

Example: Join Two Tables

In the following example, a table from an Oracle database and a table from a DB2 database are joined. DBMASTER= is set to YES to indicate that the Oracle table is the larger table. The DB2 table is the smaller table.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
libname mydblib2 fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn dsnuser=db2user dsnpwd=db2pwd;
proc sql;
   select * from mydblib.bigtab(dbmaster=yes), mydblib2.smalltab
bigtab.x=smalltab.x;
```

DBNULL= Data Set Option

When a table is created, this option indicates whether a null is a valid value for the specified columns.

Valid in:	DATA and PROC steps
Default:	data source-specific
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

DBNULL=<_ALL_=YES | NO > | (<*column-name-1*=YES | NO > <…<*column-name-n*=YES | NO >>)

Syntax Description

ALL
specifies that the YES or NO applies to all columns in the table. (This is valid for Oracle and Teradata only.)

YES
specifies that a null is not valid for the specified columns in the table.

NO
specifies that a null is not valid for the specified columns in the table.

Details

This option is valid only when you are creating tables. If you specify more than one column name, the names must be separated with spaces.

The DBNULL= option processes values from left to right, so if you specify a column name twice, or if you use the _ALL_ value, the last value overrides the first value that is specified for the column.

Examples

Example 1: Prevent Specific Columns from Accepting Null Values

In the following example, by using the DBNULL= option, the EMPID and JOBCODE columns in the new MYDBLIB.MYDEPT2 table are prevented from accepting NULL values. If the EMPLOYEES table contains NULL values in the EMPID or JOBCODE columns, the DATA step fails.

```
data mydblib.mydept2(dbnull=(empid=no jobcode=no));
   set mydblib.employees;
run;
```

Example 2: Prevent All Columns from Accepting Null Values

In the following example, all columns in the new MYDBLIB.MYDEPT3 table except for the JOBCODE column are prevented from accepting NULL values. If the EMPLOYEES table contains NULL values in any column other than the JOBCODE column, the DATA step fails.

```
data mydblib.mydept3(dbnull=(_ALL_=no jobcode=YES));
   set mydblib.employees;
run;
```

DBNULLKEYS= Data Set Option

Controls the format of the WHERE clause with regard to NULL values when you use the DBKEY= data set option.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting

Supports: DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

DBNULLKEYS=YES | NO

Details

If there might be NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, then use DBNULLKEYS=YES. When you specify DBNULLKEYS=YES and specify a column that is not defined as NOT NULL in the DBKEY= data set option, SAS generates a WHERE clause that can find NULL values. For example, if you specify DBKEY=COLUMN and COLUMN is not defined as NOT NULL, SAS generates a WHERE clause with the following syntax:

```
WHERE ((COLUMN = ?) OR ((COLUMN IS NULL) AND (? IS NULL)))
```

This syntax enables SAS to prepare the statement once and use it for any value (NULL or NOT NULL) in the column. Note that this syntax has the potential to be much less efficient than the shorter form of the WHERE clause (presented below). When you specify DBNULLKEYS=NO or specify a column that is defined as NOT NULL in the DBKEY= option, SAS generates a simple WHERE clause.

If you know that there are no NULL values in the transaction table or the master table for the columns that you specify in the DBKEY= option, you can use DBNULLKEYS=NO. If you specify DBNULLKEYS=NO and specify DBKEY=COLUMN, SAS generates a shorter form of the WHERE clause (regardless of whether the column specified in DBKEY= is defined as NOT NULL):

```
WHERE (COLUMN = ?)
```

DBSASLABEL= Data Set Option

Specifies how the table driver returns column labels.

Valid in: DATA and PROC steps

Default: COMPAT

Supports: All

Syntax

DBSASLABEL=COMPAT | NONE

Syntax Description

COMPAT
specifies to return the column label to the application. For data sources that support storing column labels on the table (for example, SAS data sets), the engine returns the label to the application. If there is no label stored, no label is returned. For data sources that do not store column labels on the table, the engine returns the column name as the label.

NONE
specifies that column labels are not returned even if one exists. The engine returns blanks for the column labels.

Example: Return Blank Labels for Aliases in Headings

The following example demonstrates how DBSASLABEL= is used as a data set option to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn dbsaslabel=none;
proc sql;
   select deptno as Department ID, loc as Location
   from mydblib.dept (dbsaslabel=none);
```

Without the DBSASLABEL= option set to NONE, the aliases would be ignored and DEPTNO and LOC would be used as column headings in the result set.

See Also

To assign this option to a group of tables or views, use the DBSASLABEL= LIBNAME option.

DBSASTYPE= Data Set Option

Specifies data types to override the default SAS data types during input processing.

Valid in:	DATA and PROC steps
Default:	DBMS-specific
Supports:	All

Syntax

DBSASTYPE=*(column-name-1*=<'> *SAS-data-type*<'> <*...column-name-n*=<'>*SAS-data-type*<'>>)

Syntax Description

column-name
specifies a DBMS column name.

SAS-data-type
specifies a SAS data type. SAS data types include the following: CHAR(*n*), NUMERIC, DATETIME, DATE, TIME. See the documentation for your database for details.

Details

By default, the LIBNAME engine converts each DBMS data type to a SAS data type during input processing. When you need a different data type, you can use this option to override the default and assign a SAS data type to each specified DBMS column. Some conversions might not be supported. If a conversion is not supported, SAS prints an error to the log.

Examples

Example 1: Override the Default Data Type

In the following example, DBSASTYPE= specifies a data type to use for the column MYCOLUMN when SAS is printing ODBC data. If the data in this DBMS column is stored in a format that SAS does not support, such as SQL_DOUBLE(20), this enables SAS to print the values.

```
proc print data=mylib.mytable
   (dbsastype=(mycolumn='CHAR(20)'));
run;
```

Example 2: Convert Column Length

In the following example, the data stored in the DBMS FIBERSIZE column has a data type that provides more precision than what SAS could accurately support, such as DECIMAL(20). If you used only PROC PRINT on the DBMS table, the data might be rounded or displayed as a missing value. Instead, you could use DBSASTYPE= to convert the column to a character field of the length 21. Because the DBMS performs the conversion before the data is brought into SAS, there is no loss of precision.

```
proc print data=mylib.specprod
   (dbsastype=(fibersize='CHAR(21)'));
run;
```

Example 3: Append Tables to Match Data Types

The following example, uses DBSASTYPE= to append one table to another when the data types are not comparable. If the SAS data set has a column EMPID defined as CHAR(20) and the DBMS table has an EMPID column defined as DECIMAL (20), you can use DBSASTYPE= to make them match:

```
proc append base=dblib.hrdata (dbsastype=(empid='CHAR(20)'))
           data=saslib.personnel;
run;
```

DBSASTYPE= specifies to SAS that the EMPID is defined as a character field of length 20. When a row is inserted from the SAS data set into a DBMS table, the DBMS performs a conversion of the character field to the DBMS data type DECIMAL(20).

DBTYPE= Data Set Option

Specifies a data type to use instead of the default DBMS data type when SAS creates a DBMS table.

Valid in:	DATA and PROC steps
Default:	DBMS-specific
Supports:	All

Syntax

DBTYPE=(*column-name-1*=<'> *DBMS-type*<'>
<...*column-name-n*=<'>*DBMS-type*<'>>)

Syntax Description

column-name
specifies a DBMS column name.

DBMS-type
specifies a DBMS data type. See the documentation for your data source for the default data types for your DBMS.

Details

By default, the interface for your DBMS converts each SAS data type to a predetermined DBMS data type when writing data to your DBMS. When you need a different data type, use DBTYPE= to override the default data type.

Teradata Details: In Teradata, you can use DBTYPE= to specify data attributes for a column. See your Teradata CREATE TABLE documentation for information about the data type attributes that you can specify. If you specify DBNULL=NO for a column, do not also use DBTYPE= to specify NOT NULL for that column. If you do, 'NOT NULL' is inserted twice in the column definition. This causes Teradata to generate an error message.

Examples

Example 1: Specify Data Types for Columns

In the following example, DBTYPE= specifies the data types that are used when you create columns in the DBMS table.

```
data mydblib.newdept(dbtype=(deptno='number(10,2)' city='char(25)'));
   set mydblib.dept;
run;
```

Example 2: Specify Data Types for Columns in a New Table

The following example creates a new Teradata table, NEWDEPT, specifying the Teradata data types for the DEPTNO and CITY columns.

```
data mydblib.newdept(dbtype=(deptno='byteint' city='char(25)'));
   set dept;
run;
```

Example 3: Specify a Data Type for a Column in a New Table

The following example creates a new Teradata table, NEWEMPLOYEES, and specifies a data type and attributes for the EMPNO column. The example encloses the Teradata type and attribute information in double quotation marks. Single quotation marks conflict with those that are required by the Teradata FORMAT attribute. If you use single quotation marks, SAS returns syntax error messages.

```
data mydblib.newemployees(dbtype= (empno="SMALLINT FORMAT '9(5)'
     CHECK (empno >= 100 AND empno <= 2000)"));
   set mydblib.employees;
run;
```

DROP= Data Set Option

For an input data set, excludes the specified columns from processing; for an output data set, excludes the specified columns from being written to the data set.

Valid in:	DATA and PROC steps
Category:	Variable Control
Supports:	All

Syntax

DROP=*column(s)*

Syntax Description

column(s)
lists one or more column names. You can list the columns in any form that SAS allows.

Details

If the option is associated with an input data set, the columns are not available for processing. If the DROP= data set option is associated with an output data set, SAS does not write the columns to the output data set, but they are available for processing.

Comparisons

- The DROP= data set option differs from the DROP statement in these ways:
  - In DATA steps, the DROP= data set option can apply to both input and output data sets. The DROP statement applies only to output data sets.
  - In DATA steps, when you create multiple output data sets, use the DROP= data set option to write different columns to different data sets. The DROP statement applies to all output data sets.
  - In PROC steps, you can use only the DROP= data set option, not the DROP statement.
- The KEEP= data set option specifies a list of columns to be included in processing or to be written to the output data set.

Examples

Example 1: Excluding Columns from Input

In this example, the columns SALARY and GENDER are not included in processing and they are not written to either output data set:

```
data payroll;
   input idnum $3. +3 gender $1. +4 jobcode $3. +9 salary 5.
         +2 birth date7. +2 hired date7.;
   informat birth date7. hired date7.;
   format birth date7. hired date7.;
   datalines;
919   M    TA2          34376  12SEP60  04JUN87
653   F    ME2          35108  15OCT64  09AUG90
400   M    ME1          29769  05NOV67  16OCT90
350   F    FA3          32886  31AUG65  29JUL90
401   M    TA3          38822  13DEC50  17NOV85
499   M    ME3          43025  26APR54  07JUN80
101   M    SCP          18723  06JUN62  01OCT90
333   M    PT2          88606  30MAR61  10FEB81
402   M    TA2          32615  17JAN63  02DEC90
479   F    TA3          38785  22DEC68  05OCT89
403   M    ME1          28072  28JAN69  21DEC91
739   M    PT1          66517  25DEC64  27JAN91
658   M    SCP          17943  08APR67  29FEB92
428   F    PT1          68767  04APR60  16NOV91
782   M    ME2          35345  04DEC70  22FEB92
244   M    ME2          36925  31AUG63  17JAN88
383   M    BCK          25823  25JAN68  20OCT92
574   M    FA2          28572  27APR60  20DEC92
789   M    SCP          18326  25JAN57  11APR78
404   M    PT2          91376  24FEB53  01JAN80
437   F    FA3          33104  20SEP60  31AUG84
639   F    TA3          40260  26JUN57  28JAN84
269   M    NA1          41690  03MAY72  28NOV92
065   M    ME2          35090  26JAN44  07JAN87
876   M    TA3          39675  20MAY58  27APR85
037   F    TA1          28558  10APR64  13SEP92
129   F    ME2          34929  08DEC61  17AUG91
988   M    FA3          32217  30NOV59  18SEP84
405   M    SCP          18056  05MAR66  26JAN92
430   F    TA2          32925  28FEB62  27APR87
983   F    FA3          33419  28FEB62  27APR87
134   F    TA2          33462  05MAR69  21DEC88
118   M    PT3         111379  16JAN44  18DEC80
438   F    TA3          39223  15MAR65  18NOV87
125   F    FA2          28888  08NOV68  11DEC87
475   F    FA2          27787  15DEC61  13JUL90
117   M    TA3          39771  05JUN63  13AUG92
935   F    NA2          51081  28MAR54  16OCT81
124   F    FA1          23177  10JUL58  01OCT90
422   F    FA1          22454  04JUN64  06APR91
616   F    TA2          34137  01MAR70  04JUN93
406   M    ME2          35185  08MAR61  17FEB87
120   M    ME1          28619  11SEP72  07OCT93
094   M    FA1          22268  02APR70  17APR91
389   M    BCK          25028  15JUL59  18AUG90
905   M    PT1          65111  16APR72  29MAY92
407   M    PT1          68096  23MAR69  18MAR90
114   F    TA2          32928  18SEP69  27JUN87
410   M    PT2          84685  03MAY67  07NOV86
439   F    PT1          70736  06MAR64  10SEP90
409   M    ME3          41551  19APR50  22OCT81
408   M    TA2          34138  29MAR60  14OCT87
121   M    ME1          29112  26SEP71  07DEC91
991   F    TA1          27645  07MAY72  12DEC92
102   M    TA2          34542  01OCT59  15APR91
```

```
356   M    ME2         36869  26SEP57  22FEB83
545   M    PT1         66130  12AUG59  29MAY90
292   F    ME2         36691  28OCT64  02JUL89
440   F    ME2         35757  27SEP62  09APR91
368   M    FA2         27808  11JUN61  03NOV84
369   M    TA2         33705  28DEC61  13MAR87
411   M    FA2         27265  27MAY61  01DEC89
113   F    FA1         22367  15JAN68  17OCT91
704   M    BCK         25465  30AUG66  28JUN87
900   M    ME2         35105  25MAY62  27OCT87
126   F    TA3         40899  28MAY63  21NOV80
677   M    BCK         26007  05NOV63  27MAR89
441   F    FA2         27158  19NOV69  23MAR91
421   M    TA2         33155  08JAN59  28FEB90
119   M    TA1         26924  20JUN62  06SEP88
834   M    BCK         26896  08FEB72  02JUL92
777   M    PT3        109630  23SEP51  21JUN81
663   M    BCK         26452  11JAN67  11AUG91
106   M    PT2         89632  06NOV57  16AUG84
103   F    FA1         23738  16FEB68  23JUL92
477   M    FA2         28566  21MAR64  07MAR88
476   F    TA2         34803  30MAY66  17MAR87
379   M    ME3         42264  08AUG61  10JUN84
104   M    SCP         17946  25APR63  10JUN91
009   M    TA1         28880  02MAR59  26MAR92
412   M    ME1         27799  18JUN56  05DEC91
115   F    FA3         32699  22AUG60  29FEB80
128   F    TA2         32777  23MAY65  20OCT90
442   F    PT2         84536  05SEP66  12APR88
417   M    NA2         52270  27JUN64  07MAR89
478   M    PT2         84203  09AUG59  24OCT90
673   M    BCK         25477  27FEB70  15JUL91
839   F    NA1         43433  29NOV70  03JUL93
;

data myfiles.plan1 myfiles.plan2;
   set myfiles.payroll (drop=salary gender);
   if hired <'01jan98'd then output myfiles.plan1;
   else output myfiles.plan2;
run;

proc print data=myfiles.plan1(obs=2);
   title 'plan1';
proc print data=myfiles.plan2(obs=2);
   title 'plan2';
run;
```

You cannot use SALARY or GENDER in any logic in the DATA step because DROP= prevents the SET statement from reading them from PAYROLL.

Example 2: Processing Columns without Writing Them to a Data Set

In this example, SALARY and GENDER are not written to PLAN2, but they are written to PLAN1:

```
data myfiles.plan1 myfiles.plan2 (drop=salary gender);
   set myfiles.payroll;
   if hired <'01jan98'd then output myfiles.plan1;
```

```
      else output myfiles.plan2;
   run;
```

ENCRYPT= Data Set Option

Specifies whether to encrypt an output table.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Restriction:	Use with output tables only.
Supports:	SAS data set

Syntax

ENCRYPT=YES | NO

Syntax Description

YES

encrypts the file. The encryption method uses passwords. At a minimum, you must specify the READ= or the PW= data set option at the same time that you specify ENCRYPT=YES. Because the encryption method uses passwords, you cannot change *any* password on an encrypted data set without re-creating the data set.

CAUTION: Record all passwords. If you forget the password, you cannot reset it without assistance from SAS. The process is time-consuming and resource-intensive.

NO

does not encrypt the file.

Details

- Encryption requires approximately the same amount of CPU resources as compression.
- When a SAS data set is encrypted, all associated indexes are also encrypted.
- You cannot use PROC CPORT on encrypted files.

Example: Using the ENCRYPT=YES Option

This example creates an encrypted SAS data set:

```
data myfiles.salary (encrypt=yes);
   input name $ yrsal bonuspct;
   datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
Agnetha   70998  4.1
;
```

FIRSTOBS= Data Set Option

Specifies the first row in a data source that SAS processes.

Valid in:	DATA and PROC steps
Category:	Observation Control
Restriction:	Valid for input (read) processing only.
Supports:	All

Syntax

FIRSTOBS= *n*| *n*K | *n*M | *n*G | *hex*X | MIN | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G**
specifies the number of the first row to process in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, a value of `8` specifies the 8th row, and a value of `3k` specifies 3,072.

***hex*X**
specifies the number of the first row to process as a hexadecimal value. You must specify the value beginning with a number (0-9), followed by an X. For example, the value `2dx` sets the 45th row as the first row to process.

MIN
sets the number of the first row to process to 1. This is the default.

MAX
sets the number of the first row to process to the maximum number of rows in the data set, up to the largest eight-byte, signed integer, which is 2^{63}-1, or approximately 9.2 quintillion rows.

Details

The FIRSTOBS= data set option affects a single, existing file. Use the FIRSTOBS= system option to affect all steps for the duration of your current SAS session.

FIRSTOBS= is valid for input (read) processing only. Specifying FIRSTOBS= is not valid for output or update processing.

You can apply FIRSTOBS= processing to WHERE processing.

Comparisons

- The FIRSTOBS= data set option overrides the FIRSTOBS= system option for the individual data set.
- The FIRSTOBS= data set option specifies a starting point for processing. The OBS= data set option specifies an ending point.
- When external files are read, the FIRSTOBS= option in the INFILE statement specifies which record to read first.

Examples

Example 1: Use FIRSTOBS= in PROC PRINT

This PROC step prints the data set STUDY beginning with row 20:

```
data myfiles.study;
   input char $ @@;
   datalines;
aa bb cc dd ee ff gg
hh ii jj kk ll mm nn
oo pp qq rr ss tt uu
vv ww xx yy zz
;

proc print data=myfiles.study (firstobs=20);
run;
```

Example 2: Use FIRSTOBS= in the SET Statement

This SET statement uses both FIRSTOBS= and OBS= to read only rows 5 through 10 from the data set STUDY. Data set NEW contains six rows.

```
data new;
  set study(firstobs=5 obs=10);
run;

proc print data=new;
 run;
```

IDXNAME= Data Set Option

Directs SAS to use a specific index to match the conditions of a WHERE expression.

Valid in:	DATA and PROC steps
Category:	User Control of SAS Index Usage
Restrictions:	Use with input data sets only. Mutually exclusive with IDXWHERE= data set option
Supports:	SAS data set

Syntax

IDXNAME=*index-name*

Syntax Description

index-name

specifies the name (up to 32 characters) of a simple or composite index for the SAS data set. SAS does not attempt to determine whether the specified index is the best one or if a sequential search might be more resource efficient.

Interaction: The specification is not a permanent attribute of the data set and is valid only for the current use of the data set.

Details

By default, to satisfy the conditions of a WHERE expression for an indexed SAS data set, SAS identifies zero or more candidate indexes that could be used to optimize the WHERE expression. From the list of candidate indexes, SAS selects the one that it determines will provide the best performance, or rejects all of the indexes if a sequential pass of the data is expected to be more efficient.

Because the index that SAS selects might not always provide the best optimization, you can direct SAS to use one of the candidate indexes by specifying the IDXNAME= data set option. If you specify an index that SAS does not identify as a candidate index, then IDXNAME= will not process the request. That is, IDXNAME= will not allow you to specify an index that would produce incorrect results.

Comparisons

IDXWHERE= enables you to override the SAS decision about whether to use an index.

Example: Specify an Index for a WHERE Expression

This example uses the IDXNAME= data set option in order to direct SAS to use a specific index to optimize the WHERE expression. SAS then disregards the possibility that a sequential search of the data set might be more resource efficient and does not attempt to determine whether the specified index is the best one. (Note that the EMPNUM index was not created with the NOMISS option.)

```
data mydata.empnew;
   set mydata.employee (idxname=empnum);
   where empnum < 2000;
run;
```

IDXWHERE= Data Set Option

Specifies whether SAS uses an index search or a sequential search to match the conditions of a WHERE expression.

Valid in:	DATA and PROC steps
Category:	User Control of SAS Index Usage
Restrictions:	Use with input data sets only. Mutually exclusive with IDXNAME= data set option
Supports:	SAS data set

Syntax

IDXWHERE=YES|NO

Syntax Description

YES
tells SAS to choose the best index to optimize a WHERE expression, and to disregard the possibility that a sequential search of the data set might be more resource-efficient.

NO
tells SAS to ignore all indexes and satisfy the conditions of a WHERE expression with a sequential search of the data set.

Details

By default, to satisfy the conditions of a WHERE expression for an indexed SAS data set, SAS decides whether to use an index or to read the data set sequentially. The software estimates the relative efficiency and chooses the method that is more efficient.

You might need to override the software's decision by specifying the IDXWHERE= data set option, because the decision is based on general rules that occasionally cannot produce the best results. That is, by specifying the IDXWHERE= data set option, you are able to determine the processing method.

Note: The specification is not a permanent attribute of the data set and is valid only for the current use of the data set.

Comparisons

IDXNAME= enables you to direct SAS to use a specific index.

Examples

Example 1: Specifying Index Usage

This example uses the IDXWHERE= data set option to tell SAS to decide which index is the best to optimize the WHERE expression. SAS then disregards the possibility that a sequential search of the data set might be more resource-efficient:

```
data mydata.empnew;
   set mydata.employee (idxwhere=yes);
   where empnum < 2000;
```

Example 2: Specifying No Index Usage

This example uses the IDXWHERE= data set option to tell SAS to ignore indexes and to satisfy the conditions of the WHERE expression with a sequential search of the data set:

```
   data mydata.empnew;
      set mydata.employee (idxwhere=no);
      where empnum < 2000;
```

IGNORE_READ_ONLY_COLUMNS= Data Set Option

Specifies whether to ignore or include columns whose data types are read-only when generating an SQL statement for inserts or updates.

Valid in:	DATA and PROC steps
Default:	NO
Supports:	All

Syntax

IGNORE_READ_ONLY_COLUMNS=YES | NO

Syntax Description

YES
specifies to ignore columns whose data types are read-only when generating INSERT and UPDATE statements

NO
specifies to include columns whose data types are read-only when generating INSERT and UPDATE statements

Details

Several databases include data types that can be read-only, such as the DB2 TIMESTAMP data type. Also, some databases have properties that allow certain data types to be read-only, such as the Microsoft SQL Server identity property.

When the IGNORE_READ_ONLY_COLUMNS= option is set to NO (the default), and a table contains a column that is read-only, an error is returned indicating that the data could not be modified for that column.

Example: Insert Data into a Table

For the following example, a database that contains the table Products is created with two columns: ID and PRODUCT_NAME. The ID column is defined by a read-only data type and PRODUCT_NAME is a character column.

```
create table x.products (id int identity primary key, product_name varchar(40))
```

If you have a SAS data set that contains the name of your products, you can insert the data from the SAS data set into the Products table:

```
data x.products;
   id=1;
   product_name='screwdriver';
   output;
   id=2;
   product_name='hammer';
   output;
   id=3;
   product_name='saw';
   output;
   id=4;
   product_name='shovel';
   output;
run;
```

With IGNORE_READ_ONLY_COLUMNS=NO (the default), an error is returned by the database because in this example, the ID column cannot be updated. However, if you set the option to YES and execute a PROC APPEND, the append succeeds, and the SQL statement that is generated does not contain the ID column.

```
libname x fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=db2dsn dsnuser=db2user dsnpwd=db2pwd
   ignore_read_only_columns=yes;
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
proc append base=x.productsdata=work.products;
run;
```

INSERTBUFF= Data Set Option

Specifies the number of rows in a single insert operation.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	All

Syntax

INSERTBUFF=*positive-integer*

Syntax Description

positive-integer
specifies the number of rows to insert.

Details

All data sources default to INSERT_SQL=YES except for SAS data sets. When INSERT_SQL=YES, INSERTBUFF= defaults to 1 and single row inserts are used. The optimal value for this option varies with factors such as network type and available memory. You might need to experiment with the different values to determine the best value for your site.

The SAS application messages that indicate the success or failure of an insert operation only represent information for a single insert, even when multiple inserts are performed. Therefore, when you assign a value that is greater than INSERTBUFF=1, these messages might be incorrect.

If you specify the DBCOMMIT= option with a value that is less than the value of the INSERTBUFF=, then DBCOMMIT= overrides INSERTBUFF=. If neither DBCOMMIT nor INSERTBUFF are specified, INSERTBUFF defaults to a block size of 32K. SAS determines the number of rows by dividing 32K by the size of each row.

Note: When you insert by using the VIEWTABLE window or the FSVIEW or FSEDIT procedure, use INSERTBUFF=1 to prevent the driver from trying to insert multiple rows. These features do not support inserting more than one row at a time.

Note: Additional driver-specific restrictions might apply.

DB2 under UNIX and PC Hosts Details: You must specify INSERT_SQL=YES in order to use this option. If one row in the insert buffer fails, all rows in the insert buffer fail.

Microsoft SQL Server, Greenplum Details: You must specify INSERT_SQL=YES in order to use this option.

INSERT_SQL= Data Set Option

Determines the method that is used to insert rows into a data source.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting

Supports: All

Syntax

INSERT_SQL=YES | NO

Syntax Description

YES
specifies to use the data source's SQL insert method to insert new rows into a table.

NO
specifies to use an alternate (DBMS-specific) method to add new rows to a table.

Details

SAS data sets generally have improved insert performance when INSERT_SQL=NO, which is the default for that SAS data sets. Other data sources might have inferior insert performance (or might fail) unless INSERT_SQL=YES. You should experiment to determine the optimal setting for your situation.

KEEP= Data Set Option

For an input data set, specifies the columns to process; for an output data set, specifies the columns to write to the data set.

Valid in: DATA and PROC steps

Category: Variable Control

Supports: All

Syntax

KEEP=*column(s)*

Syntax Description

column(s)
lists one or more column names. You can list the columns in any form that SAS allows.

Details

When the KEEP= data set option is associated with an input data set, only those columns that are listed after the KEEP= data set option are available for processing. When the KEEP= data set option is associated with an output data set, only the columns listed after the option are written to the output data set. However, all columns are available for processing.

Comparisons

- The KEEP= data set option differs from the KEEP statement in the following ways:
  - In DATA steps, the KEEP= data set option can apply to both input and output data sets. The KEEP statement applies only to output data sets.

- In DATA steps, when you create multiple output data sets, use the KEEP= data set option to write different columns to different data sets. The KEEP statement applies to all output data sets.
- In PROC steps, you can use only the KEEP= data set option, not the KEEP statement.

- The DROP= data set option specifies columns to omit during processing or to omit from the output data set.

Example: Limit Columns in the SET Statement

In this example, only IDNUM and SALARY are read from PAYROLL, and they are the only columns in PAYROLL that are available for processing:

```
data myfiles.bonus;
   set myfiles.payroll (keep=idnum salary);
   bonus=salary*1.1;
run;

   proc print data=myfiles.bonus(obs=2);
     title 'bonus data set';
   run;
```

LABEL= Data Set Option

Specifies a label for a SAS data set.

Valid in: DATA and PROC steps

Category: Data Set Control

Supports: SAS data set

Syntax

LABEL='*label*'

Syntax Description

'label'
: specifies a text string of up to 256 characters. If the label text contains single quotation marks, use double quotation marks around the label. Or, use two single quotation marks in the label text and enclose the string in single quotation marks. To remove a label from a data set, assign a label that is equal to a blank that is enclosed in quotation marks.

Details

You can use the LABEL= option on both input and output data sets. When you use LABEL= on input data sets, it assigns a label for the file for the duration of that DATA or PROC step. When it is specified for an output data set, the label becomes a permanent part of that file and can be printed using the CONTENTS or DATASETS procedure, and modified using PROC DATASETS.

Comparisons

- The LABEL= data set option enables you to specify labels only for data sets. You can specify labels for the columns in a data set using the LABEL statement.
- The LABEL= option in the ATTRIB statement also enables you to assign labels to columns.

Example: Assign Labels to Data Sets

These examples assign labels to data sets:

```
data myfiles.w2 (label='1976 W2 Info, Hourly');

data myfiles.new (label='Peter''s List');

data myfiles.new (label="Hillside's Daily Account");

data myfiles.sales (label='Sales For May(NE)');
```

NULLCHAR= Data Set Option

Indicates how missing character values are handled during insert, update, DBINDEX=, and DBKEY= processing.

Valid in:	DATA and PROC steps
Default:	SAS
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

NULLCHAR= SAS | YES | NO

Syntax Description

SAS
indicates that missing character values are treated as NULL values if the DBMS allows NULLs. Otherwise, they are treated as the NULLCHARVAL= value.

YES
indicates that missing character values are treated as NULL values if the DBMS allows NULLs. Otherwise, an error is returned.

NO
indicates that missing character values are treated as the NULLCHARVAL= value (regardless of whether the DBMS allows NULLs for the column).

Details

This option affects insert and update processing and also applies when you use the DBINDEX= and DBKEY= options.

This option works in conjunction with the NULLCHARVAL= data set option, which determines what is inserted when NULL values are not allowed.

All missing numeric values (represented in SAS as '.') are treated by the DBMS as NULLs.

NULLCHARVAL= Data Set Option

Defines the character string that replaces missing character values during insert, update, DBINDEX=, and DBKEY= processing.

Valid in:	DATA and PROC steps
Default:	blank
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

NULLCHARVAL=*'character-string'*

Details

This option affects insert and update processing and also applies when you use the DBINDEX= and DBKEY= options.

This option works with the NULLCHAR= option, which determines whether a missing character value is treated as a NULL value.

If NULLCHARVAL= is longer than the maximum column width, one of the following occurs:

- The string is truncated if DBFORCE=YES.
- The operation fails if DBFORCE=NO.

OBS= Data Set Option

Specifies the last row in a data source that SAS processes.

Valid in:	DATA and PROC steps
Category:	Observation Control
Default:	MAX
Restriction:	Use with input files only
Supports:	All

Syntax

OBS= *n* | *n*K | *n*M | *n*G | *n*T | *hex*X | MIN | MAX

Syntax Description

***n* | *n*K | *n*M | *n*G | *n*T**
specifies a number to indicate when to stop processing rows, with *n* being an integer. Using one of the letter notations results in multiplying the integer by a specific value. That is, specifying K (kilobytes) multiplies the integer by 1,024, M (megabytes) multiplies by 1,048,576, G (gigabytes) multiplies by 1,073,741,824, or T (terabytes)

multiplies by 1,099,511,627,776. For example, a value of **`20`** specifies 20 rows, whereas a value of **`3m`** specifies 3,145,728 rows.

***hex*X**
: specifies a number to indicate when to stop processing as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the hexadecimal value F8 must be specified as **`0F8x`** in order to specify the decimal equivalent of 248. The value **`2dx`** specifies the decimal equivalent of 45.

MIN
: sets the number to indicate when to stop processing to 0. Use OBS=0 in order to create an empty data set that has the structure, but not the rows, of another data set.

Interaction: If OBS=0 and the NOREPLACE option is in effect, then SAS can still take certain actions because it actually executes each DATA and PROC step in the program, using no rows. For example, SAS executes procedures, such as CONTENTS and DATASETS, that process libraries or SAS data sets.

MAX
: sets the number to indicate when to stop processing to the maximum number of rows in the data set, up to the largest 8-byte, signed integer, which is 2^{63}-1, or approximately 9.2 quintillion. This is the default.

Details

OBS= tells SAS when to stop processing rows. For example, if OBS=10 is specified, the result is ten rows or records.

OBS= is valid only when an existing data set is read.

In WHERE processing, SAS first subsets the data, and then applies OBS= to the subset. The FEDSVR engine does not have the concept of observation numbering from the original data set. It sends back the number of rows requested, numbered chronologically, regardless of where they occur in the data set.

Comparisons

- The OBS= data set option overrides the OBS= system option for the individual data set.
- The OBS= data set option specifies an ending point for processing. The FIRSTOBS= data set option specifies a starting point.
- The OBS= data set option enables you to select rows from data sets. You can select rows to be read from external data files by using the OBS= option in the INFILE statement.

Examples

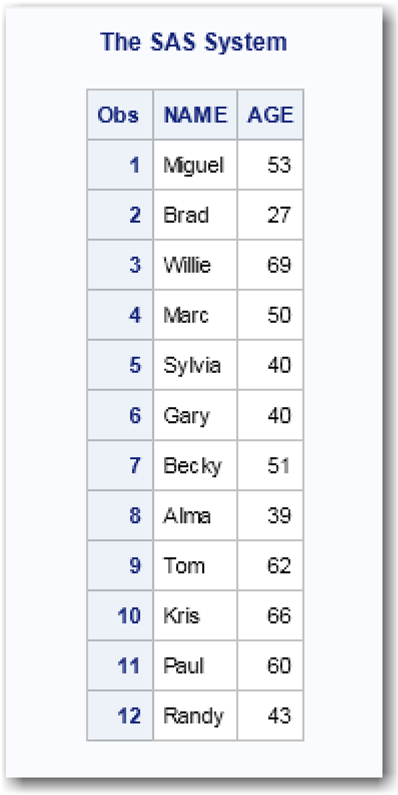
Example 1: Using OBS= to Specify When to Stop Processing Rows

This example illustrates the result of using OBS= to tell SAS when to stop processing rows. This example creates a SAS data set that contains 15 rows, and then executes the PRINT procedure with OBS=12. The result is 12 rows.

```
data myfiles.Ages;
   input Name $ Age;
   datalines;
Miguel 53
Brad 27
Willie 69
```

```
Marc 50
Sylvia 40
Gary 40
Becky 51
Alma 39
Tom 62
Kris 66
Paul 60
Randy 43
Barbara 52
Virginia 72
Arun 25
;
proc print data=myfiles.Ages (obs=12);
run;
```

Output 13.1 *PROC PRINT Output by Using OBS=*


The SAS System

Obs	NAME	AGE
1	Miguel	53
2	Brad	27
3	Willie	69
4	Marc	50
5	Sylvia	40
6	Gary	40
7	Becky	51
8	Alma	39
9	Tom	62
10	Kris	66
11	Paul	60
12	Randy	43

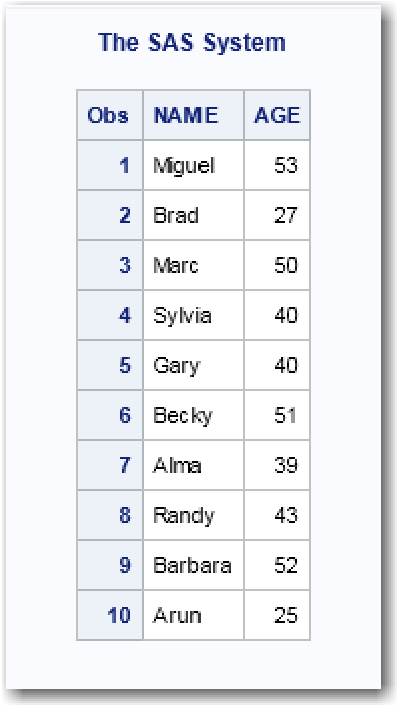
Example 2: Using OBS= with WHERE Processing

This example illustrates the result of using OBS= along with WHERE processing. The example uses the data set that was created in Example 1, which contains 15 rows, and assumes that the SAS session has been reset to the defaults FIRSTOBS=1 and OBS=MAX. This example returns the first 10 rows that meet the WHERE criteria.

```
libname myfiles fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=basedsn;

options obs=10;
proc print data=myfiles.Ages;
```

```
   where Age LT 60;
run;
```

Output 13.2 *PROC PRINT Using a WHERE Statement and OBS=*



The SAS System

Obs	NAME	AGE
1	Miguel	53
2	Brad	27
3	Marc	50
4	Sylvia	40
5	Gary	40
6	Becky	51
7	Alma	39
8	Randy	43
9	Barbara	52
10	Arun	25

PRESERVE_COL_NAMES= Data Set Option

Preserves spaces, special characters, and case sensitivity in column names when you create tables.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	All

Syntax

PRESERVE_COL_NAMES=NO | YES

Syntax Description

NO

specifies that column names that are used in table creation are derived from SAS column names by using the SAS column name normalization rules. (For more information, see the "VALIDVARNAME= System Option" on page 62 .) However, the data source applies its specific normalization rules to the SAS column names when it creates the column names.

The use of N-Literals to create column names that use database keywords or special symbols other than the underscore character might be illegal when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

YES

specifies that column names that are used in table creation are passed to the data source with special characters and the exact, case-sensitive spelling of the name preserved.

Teradata Details: YES is the only supported value for this option.

Details

This option applies only when you create a new table. When you create a table, you assign the column names by using one of the following methods:

- To control the case of the column names, specify columns with the desired case and set PRESERVE_COL_NAMES=YES. If you use special symbols or blanks, you must set VALIDVARNAME=ANY and use N-Literals.
- To enable the DBMS to normalize the column names according to its naming conventions, specify columns with any case and set PRESERVE_COL_NAMES=NO.

Note: When you read from, insert rows into, or modify data in an existing DBMS table, SAS identifies the database column names by their spelling. Therefore, when the database column exists, the case of the column does not matter.

For additional information, see the naming conventions topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

Specify the alias PRESERVE_NAMES= if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement. Using this alias saves you some time when coding.

To use column names in your SAS program that are not valid SAS names, you must use one of the following techniques:

- Use the DQUOTE= option in PROC SQL and then reference your columns using double quotation marks. For example:

```
proc sql dquote=ansi;
   select "Total$Cost" from mydblib.mytable;
```

- Specify the global system option VALIDVARNAME=ANY and use name literals in the SAS language. For example:

```
proc print data=mydblib.mytable;
   format 'Total$Cost'n 22.2;
```

Note that if you are *creating* a table in PROC SQL, you must also include the PRESERVE_COL_NAMES=YES option. For example:

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
proc sql dquote=ansi;
   create table mydblib.mytable (preserve_col_names=yes) ("my$column" int);
```

PRESERVE_COL_NAMES= does not apply to the pass-through facility.

PW= Data Set Option

Assigns a READ, WRITE, and ALTER password to a SAS data set and enables access to a password-protected file.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Supports:	SAS data set

Syntax

PW=*password*

Syntax Description

password
must be a valid SAS name.

Details

The PW= option is supported only for unsecured DSNs. If you try to assign a password to a SAS data set that is protected by Federation Server authorization, the Federation Server will return an error.

QUALIFIER= Data Set Option

Identifies database objects, such as tables, by using a qualifier.

Valid in:	DATA and PROC steps
Alias:	CATALOG=
Default:	LIBNAME statement setting
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

QUALIFIER=*qualifier-name*

Details

If this option is omitted, the default qualifier name, if any, is used for the data source. QUALIFIER= can be used for any data source that allows three-part identifier names: *qualifier.schema.object*.

READ= Data Set Option

Assigns a READ password to a SAS data set that prevents users from reading the file, unless they enter the password.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Supports:	SAS data set

Syntax

READ=*read-password*

Syntax Description

read-password
must be a valid SAS name.

Details

The READ= option is supported only for unsecured DSNs. If you try to assign a password to a SAS data set that is protected by Federation Server authorization, the Federation Server will return an error.

READ_ISOLATION_LEVEL= Data Set Option

Specifies which level of read isolation locking to use when you are reading data.

Valid in:	DATA and PROC steps
Default:	DBMS-specific
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

READ_ISOLATION_LEVEL=*DBMS-specific-value*

Syntax Description

dbms-specific-value
See the documentation for your data source for the values for your DBMS.

Details

The ODBC and DB2 drivers ignore this option if READ_LOCK_TYPE= is not set to ROW.

The degree of isolation defines the following:

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

For additional information, see the locking topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

READ_LOCK_TYPE= Data Set Option

Specifies how data in a DBMS table is locked during a read transaction.

Valid in:	DATA and PROC steps
Default:	DBMS-specific

Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

READ_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK | VIEW

Syntax Description

TABLE
: locks the entire DBMS table. If you specify READ_LOCK_TYPE=TABLE, you must also specify the LIBNAME statement option CONNECTION=UNIQUE, or you will receive an error message. Setting CONNECTION=UNIQUE ensures that your table lock is not lost, for example, due to another table closing and committing rows in the same connection. (This value is valid in DB2 under UNIX and PC hosts, ODBC, Oracle, and Teradata interfaces.)

NOLOCK
: does not lock the DBMS table, pages, or any rows during a read transaction. (This value is valid in the Oracle interface and in the ODBC interfaces when you use the Microsoft SQL Server driver.)

VIEW
: locks the entire DBMS view. (This value is valid in the Teradata interface.)

Details

If you omit READ_LOCK_TYPE=, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement.

For additional information, see the locking topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

READBUFF= Data Set Option

Specifies the number of rows of data to read into the buffer.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	All

Syntax

READBUFF=*integer*

Syntax Description

integer
: is the positive number of rows to hold in memory. SAS allows the maximum number that is allowed by the database.

Details

This option improves performance by specifying a number of rows that can be held in memory for input into SAS. Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for READBUFF= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.

When READBUFF=1, only one row is retrieved at a time. The higher the value for READBUFF=, the more rows are retrieved in one fetch operation. If READBUFF= is not set, certain operations such as SQL SELECT statements cause the number of rows to be set to the client's default value, which for the LIBNAME engine is one row at a time. Setting READBUFF=128 can significantly boost the application's performance.

If you do not specify a value with this option, the engine calculates the buffer size based on the row length of your data (with a minimum of 10) and retrieves the number of rows in each fetch operation.

RENAME= Data Set Option

Changes the name of a column.

Valid in:	DATA and PROC steps
Category:	Variable Control
Supports:	All

Syntax

RENAME=(*old-name-1*=*new-name-1* < ...*old-name-n*=*new-name-n*>)

Syntax Description

old-name
the column that you want to rename.

new-name
the new name of the column. It must be a valid SAS name.

Details

If you use the RENAME= data set option when you create a data set, the new column name is included in the output data set. If you use RENAME= on an input data set, the new name is used in DATA step programming statements.

If you use RENAME= on an input data set that is used in a SAS procedure, SAS changes the name of the column in that procedure. If you use RENAME= along with WHERE processing such as a WHERE statement or WHERE= data set option, the new name is applied before the data is processed. You must use the new name in the WHERE expression.

If you use RENAME= in the same DATA step with either the DROP= or the KEEP= data set option, the DROP= or KEEP= data set option is applied before RENAME=. You must use the old name in the DROP= and KEEP= data set options. You cannot drop and rename the same column in the same statement.

Note: The RENAME= data set option has an effect only on data sets opened in output mode.

Comparisons

- The RENAME= data set option differs from the RENAME statement in the following ways:
  - The RENAME= data set option can be used in PROC steps and the RENAME statement cannot.
  - The RENAME statement applies to all output data sets. If you want to rename different columns in different data sets, you must use the RENAME= data set option.
  - To rename columns before processing begins, you must use a RENAME= data set option on the input data set or data sets.
- Use the RENAME statement or the RENAME= data set option when program logic requires that you rename columns such as two input data sets that have columns with the same name.

Examples

Example 1: Renaming a Column at Time of Output

This example uses RENAME= in the DATA statement to show that the column is renamed at the time it is written to the output data set. The column keeps its original name, X, during the DATA step processing:

```
data myfiles.one;
  input x y z;
  datalines;
24 595 439
243 343 034
;

data myfiles.two(rename=(x=keys));
   set one;
   z=x+y;
run;
```

Example 2: Renaming a Column at Time of Input

This example renames column X to a column named KEYS in the SET statement, which is a rename before DATA step processing. KEYS, not X, is the name to use for the column for DATA step processing.

```
data myfiles.three;
   set one(rename=(x=keys));
   z=keys+y;
run;
```

Example 3: Renaming a Column for a SAS Procedure with WHERE Processing

This example renames column Score1 to a column named Score2 for the PRINT procedure. Because the new name is applied before the data is processed, the new name must be specified in the WHERE statement.

```
proc print data=test (rename=(score1=score2));
   where score2 gt 75;
run;
```

REUSE= Data Set Option

Specifies whether new rows can be written to freed space in a compressed SAS data set.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Restriction:	Use with output data sets only.
Supports:	SAS data set

Syntax

REUSE=NO | YES

Syntax Description

NO
does not track and reuse space in a compressed SAS data set. New rows are appended to the existing data set. Specifying NO results in less efficient data storage if you delete or update many rows in the data set.

YES
tracks and reuses space in a compressed SAS data set. New rows are inserted in the space that is freed when other rows are updated or deleted.

If you plan to use procedures that add rows to the end of a compressed SAS data set (for example, the APPEND and FSEDIT procedures), use REUSE=NO. REUSE=YES causes new rows to be added wherever there is space in the file, not necessarily at the end of the file.

Details

By default, new rows are appended to an existing compressed SAS data set. If you want to track and reuse free space by deleting or updating other rows, use the REUSE= data set option when you create a compressed SAS data set.

REUSE= has meaning only when you are creating a new data set with the COMPRESS=YES data set option or system option. Using REUSE= when you are accessing an existing SAS data set has no effect.

Comparisons

The REUSE= data set option overrides the REUSE= system option.

SASDATEFMT= Data Set Option

Changes the SAS date format of a DBMS column.

Valid in:	DATA and PROC steps
Default:	DBMS-specific
Supports:	All

Syntax

SASDATEFMT=(*DBMS-date-col-1='SAS-date-format'*
<... *DBMS-date-col-n='SAS-date-format'*>)

Syntax Description

DBMS-date-col
specifies the name of a date column in a DBMS table.

SAS-date-format
specifies a SAS date format that has an equivalent (like-named) informat. For example, DATETIME21.2 is both a SAS format and a SAS informat, so it is a valid value for the *SAS-date-format* argument.

Details

If the date format of a SAS column does not match the date format of the corresponding DBMS column, you must convert the SAS date values to the appropriate DBMS date values. The SASDATEFMT= option enables you to convert date values from the default SAS date format to another SAS date format that you specify.

Use the SASDATEFMT= option to prevent date type mismatches in the following circumstances:

- during input operations to convert DBMS date values to the correct SAS DATE, TIME, or DATETIME values
- during output operations to convert SAS DATE, TIME, or DATETIME values to the correct DBMS date values.

The column names specified in this option must be DATE, DATETIME, or TIME columns; columns of any other type are ignored.

The format specified must be a valid date format; output with any other format is unpredictable.

If the SAS date format and the DBMS date format match, this option is not needed.

The default SAS date format is DBMS-specific and is determined by the data type of the DBMS column. See the documentation for your data source.

Note: For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. The SASDATEFMT= option does not currently handle these date types; however, you can use a PROC SQL view to convert the DBMS data to a SAS date format as you retrieve the data, or use a format statement in other contexts.

Oracle details: It is recommended that the DBSASTYPE= data set option be used instead of SASDATEFMT=.

Examples

Example 1: Change the Date Format in Oracle

In the following example, the APPEND procedure adds SAS data from the SASLIB.DELAY data set to the Oracle table that is accessed by MYDBLIB.INTERNAT. Using SASDATEFMT=, the default SAS format for the Oracle column DATES is changed to the DATE9. format. Data output from SASLIB.DELAY into the DATES column in MYDBLIB.INTERNAT now converts from the DATE9. format to the Oracle format assigned to that type.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
libname saslib 'your-SAS-library';
proc append base=mydblib.internat (sasdatefmt=(dates='date9.'))
   force data=saslib.delay;
run;
```

Example 2: Change a SAS Date Format to a Teradata Format

In the following example, SASDATEFMT= converts DATE1, a SAS DATETIME value, to a Teradata date column named DATE1.

```
libname x fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=teradsn dsnuser=terauser dsnpwd=terapwd;
proc sql noerrorstop;
   create table x.dateinfo ( date1 date );
   insert into x.dateinfo
   (sasdatefmt=( date1='datetime21.') )
    values ( '31dec2000:01:02:30'dt );
```

Example 3: Change a Teradata Date Format to a SAS Format

In the following example, SASDATEFMT= converts DATE1 (a Teradata date column) to a SAS DATETIME type named DATE1.

```
libname x fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=teradsn dsnuser=terauser dsnpwd=terapwd;
data sas_local;
   format date1 datetime21.;
   set x.dateinfo (sasdatefmt=(date1='datetime21.'));
run;
```

SCHEMA= Data Set Option

Enables you to read database objects, such as tables, in the specified DBMS schema.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	DB2 UNIX/PC, Greenplum, MySQL, ODBC, Oracle, Teradata

Syntax

SCHEMA=*schema-name*

Syntax Description

schema-name
is the name assigned to a logical classification of objects in a relational database.

Details

For this option to work, you must have appropriate privileges to the schema that is specified.

The values for SCHEMA= are usually case-sensitive, so be careful when you specify this option.

Oracle Details: If PRESERVE_TAB_NAMES=NO, SAS converts the SCHEMA= value to uppercase because all values in the Oracle data dictionary are converted to uppercase unless quoted.

Teradata Details: If you omit this option, a libref points to your default Teradata database, which often has the same name as your user name. You can use this option to point to a different database. This option enables you to view or modify a different user's DBMS tables if you have the required Teradata privileges. (For example, to read another user's tables, you must have the Teradata privilege SELECT for that user's tables.) The Teradata interface alias for SCHEMA= is DATABASE=. For more information about changing the default database, see the DATABASE statement in your Teradata documentation.

Examples

Example 1: Specify a Schema for a SAS Data Set

In the following example, SCHEMA= causes MYDB.TEMP_EMPS to be interpreted by DB2 as SCOTT.TEMP_EMPS.

```
proc print data=mydb.temp_emps (schema=SCOTT);
run;
```

Example 2: Specify a Schema for an Oracle Table

In the following example, SAS sends any reference to Employees as Scott.Employees.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=oradsn dsnuser=orauser dsnpwd=orapwd;
proc print data=mydblib.employees (schema=scott);
run;
```

Example 3: Specify a Schema for a Database

In the following example, user TESTUSER prints the contents of the Employees table, which is located in the Donna database.

```
libname mydblib fedsvr server="d1234.us.company.com"
   port=2171 user=user1 pwd=pass1
   dsn=teradsn dsnuser=terauser dsnpwd=terapwd;
proc print data=mydblib.employees (schema=donna);
run;
```

TYPE= Data Set Option

Specifies the data set type for a specially structured data set.

Valid in:	DATA and PROC steps
Category:	Data Set Control

Supports:	SAS data set

Syntax

TYPE=*data-set-type*

Syntax Description

data-set-type
specifies the special type of the data set.

Details

Use the TYPE= data set option in a DATA step to create a special data set in the proper format. In a procedure statement, use the option to identify the special type of the data set.

You can use the CONTENTS procedure to determine the type of a data set.

Most data sets do not have a specified type. However, there are several specially structured SAS data sets that are used by some SAS/STAT procedures. These data sets contain special columns and rows, and they are usually created by SAS statistical procedures. Because most of the special data sets are used with SAS/STAT software, they are described in the *SAS/STAT User's Guide*.

Other values are available in other SAS software products and are described in the appropriate documentation.

Note: If you use a DATA step with a SET statement to modify a special SAS data set, you must specify the TYPE= option in the DATA statement. The *data-set-type* is not automatically copied to the data set that is created.

UPDATE_ISOLATION_LEVEL= Data Set Option

Specifies the degree of isolation of the current application process from other concurrently running application processes.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

UPDATE_ISOLATION_LEVEL=*DBMS-specific-value*

Syntax Description

dbms-specific-value
See the documentation for your data source for the values for your DBMS.

Details

The degree of isolation identifies the following:

- the degree to which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree to which update activity of other concurrently executing application processes can affect the current application.

For additional information, see the locking topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

UPDATE_LOCK_TYPE= Data Set Option

Specifies how data is locked during an update transaction.

Valid in:	DATA and PROC steps
Default:	LIBNAME statement setting
Supports:	DB2 UNIX/PC, MySQL, ODBC, Oracle, Teradata

Syntax

UPDATE_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK | VIEW

Syntax Description

TABLE
locks the entire table. (This value is valid in DB2 under UNIX and PC hosts, ODBC, Oracle, and Teradata data sources.)

NOLOCK
does not lock the table, page, or any rows when reading them for update. (This value is valid in the ODBC, and Oracle data sources.)

VIEW
locks the entire DBMS view. (This is valid in the Teradata interface.)

Details

If you omit UPDATE_LOCK_TYPE=, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement. You can set a lock for one DBMS table by using the data set option or for a group of DBMS tables by using the LIBNAME statement option.

For additional information, see the locking topic in the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

WHERE= Data Set Option

Specifies specific conditions to use to select rows from a data set.

Valid in:	DATA and PROC steps
Category:	Observation Control
Restriction:	Cannot be used with the POINT= option in the SET and MODIFY statements.
Supports:	All

Syntax

WHERE=(*where-expression-1*<*logical-operator where-expression-n*>)

Syntax Description

where-expression

is an arithmetic or logical expression that consists of a sequence of operators, operands, and SAS functions. An operand is a column, a SAS function, or a constant. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation. The expression must be enclosed in parentheses.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

Details

- Use the WHERE= data set option with an input data set to select rows that meet the condition specified in the WHERE expression before SAS brings them into the DATA or PROC step for processing. Selecting rows that meet the conditions of the WHERE expression is the first operation SAS performs in each iteration of the DATA step.

  You can also select rows that are written to an output data set. In general, selecting rows at the point of input is more efficient than selecting them at the point of output. However, there are some cases when selecting rows at the point of input is not practical or not possible.
- You can apply OBS= and FIRSTOBS= processing to WHERE processing.
- You cannot use the WHERE= data set option with the POINT= option in the SET and MODIFY statements.
- If you use both the WHERE= data set option and the WHERE statement in the same DATA step, SAS ignores the WHERE statement for data sets with the WHERE= data set option. However, you can use the WHERE= data set option with the WHERE command in SAS/FSP software.

Note: Using indexed SAS data sets can improve performance significantly when you are using WHERE expressions to access a subset of the rows in a SAS data set.

Comparisons

- The WHERE statement applies to all input data sets, whereas the WHERE= data set option selects rows only from the data set for which it is specified.
- Do not confuse the purpose of the WHERE= data set option. The DROP= and KEEP= data set options select columns for processing. The WHERE= data set option selects rows.

Examples

Example 1: Selecting Rows from an Input Data Set

This example uses the WHERE= data set option to subset the SALES data set as it is read into another data set:

```
data sales;
   input product $ sales store $;
   datalines;
```

```
gizmo 234  parkview
gizmo 303  central
gizmo 124  mountain
gizmo 524  lakeside
whizmo 234 mountain
whizmo 273 lakeside
whizmo 234 parkview
whizmo 233 central
spintop 23 parkview
spintop 83 central
spintop 22 mountain
spintop 44 lakeside
;

data myfiles.whizmo;
   set myfiles.sales (where=(product='whizmo'));
run;

proc print data=myfiles.whizmo;
   title 'whizmo data set';
run;
```

Example 2: Selecting Rows from an Output Data Set

This example uses the WHERE= data set option to subset the SALES output data set:

```
data myfiles.whizmo (where=(product='whizmo'));
   set myfiles.sales;
run;

proc print data=myfiles.whizmo;
   title 'whizmo data set';
run;
```

WRITE= Data Set Option

Assigns a WRITE password to a SAS data set that prevents users from writing to a file, unless they enter the password.

Valid in:	DATA and PROC steps
Category:	Data Set Control
Supports:	SAS data set

Syntax

WRITE=*write-password*

Syntax Description

write-password
must be a valid SAS name.

Details

The WRITE= option is supported only for unsecured DSNs. If you try to assign a password to a SAS data set that is protected by Federation Server authorization, the Federation Server will return an error.

Part 4

Appendix

Appendix 1

LIBNAME Statement Syntax for Submitting a Fully-Specified Connection String

Dictionary

LIBNAME Statement Syntax for Submitting a Fully Specified Connection String

Accesses data by submitting a fully specified connection string rather than a DataFlux DSN.

Requirement: You can connect only to data sources and locations as defined through the server administration. A non-administrative user must have Connect privilege on the specified data services in order to connect to them.

Interaction: The FEDSVR LIBNAME engine supports accessing only one data source at a time. For more information, see "Federated DSNs" on page 11.

Tip: The preferred method to access data is to reference a DataFlux DSN.

See: "LIBNAME Statement Syntax" on page 41 for the LIBNAME statement syntax that shows how to reference a DataFlux DSN, and for information about server connection arguments and data source processing options.

Syntax

LIBNAME *libref* **FEDSVR**
server-connection-arguments
data-source-connection-arguments
<*data-source-processing-options*> ;

Data Source Connection Arguments

The data source connection arguments provide the connection information to access the data source or data sources. To access the data, you can submit a fully specified connection string, which includes all of the information that is required to connect to the data, such as the table driver name, physical location of the data, and any necessary authentication information that is required to retrieve data. The data source connection arguments to submit a fully specified connection string are as follows:

CONNECT_STRING="*connection-string*"

submits a fully specified connection string, which defines how to connect to the data. A connection string identifies the query language to be submitted as well as the information required to connect to the data source or data sources.

The fully specified connection string options are as follows:

DRIVER=FEDSQL;

requests the DataFlux FedSQL language driver to query the data source or data sources. By using the FedSQL language statements, an application can access SAS data sets as well as third-party database tables. The language driver takes advantage of the capabilities of a data source by passing it operations when possible. If the data source cannot do the operation, the language driver handles the operation.

Restriction: The FEDSVR engine does not support submitting the SQL that is implemented by a third-party database.

Requirement: You must specify DRIVER=FEDSQL for all data sources except a SAS data set. The DataFlux FedSQL language driver is automatically available for a SAS data set.

Interaction: For a SAS data set, whether you specify the language driver determines the required syntax. If you specify DRIVER=FEDSQL, the CONOPTS= argument is required. If you do not specify DRIVER=FEDSQL, CONOPTS= is not required. The following connection strings are both valid for a SAS data set:

```
libname lib1 fedsvr server="1234.us.company.com"
   port=2171 uid="myid" pwd="mypwd"
   connect_string="driver=fedsql;
   conopts=(catalog=base; driver=base;
   schema=(name=base; primarypath='c:\SASdata';))";
```

```
libname lib1 fedsvr server="1234.us.company.com"
   port=2171 uid="myid" pwd="mypwd"
   connect_string="catalog=base; driver=base;
   schema=(name=base; primarypath='c:\SASdata';)";
```

CONOPTS=(*data-source-connection-arguments*) ;

specifies the connection arguments in order to connect to a particular data source or data sources. For example:

```
libname lib1 fedsvr server="1234.us.company.com"
   port=2171 uid="myid" pwd="mypwd"
   connect_string="driver=fedsql;
   conopts=(driver=base; catalog=base;
   schema=(name=one; primarypath='c:\mydir';))";
```

Requirements:

Enclose the data source connection arguments in parentheses.

Precede the data source connection arguments with the keyword CONOPTS= if the language driver is specified. You must specify the language driver for all data sources except the SAS data set.

Note: In its initial release, the FEDSVR engine supports only fetch and insert functionality for Greenplum data. Update and delete operations are not supported for Greenplum.

See: Data source connection arguments are data source specific. For the data source connection arguments for a SAS data set, see "SAS Data Set Connection Arguments" on page 159. For third-party database connection arguments, see the appropriate data source reference in the *DataFlux Federation Server Administrator's Guide*.

Alias: NOPROMPT= and CONNECTSTR=

Requirements:

The connection string must be enclosed in double quotation marks.

You must have Connect privilege to the specified data source on the DataFlux Federation Server to submit a fully specified connection string.

Interaction: Some data sources require that you specify a catalog or a schema name in the connection string. For data sources where mixed case is allowed, to reference a mixed case catalog or schema name, you must enclose the name in quotation marks. For example,

```
libname lib1 fedsvr  server="1234.us.company.com"
   port=2171 uid="myid" pwd="mypwd"
   connect_string="driver=fedsql;
   conopts=(driver=base; catalog='CAT_base';
   schema=(name=one; primarypath='c:\mydir';))";
```

Note: If the connection string properties include a password, it appears as a string of X characters (**`XXXXXXXX`**) when the connection string is written to the SAS log.

Tip: When the connection string is written to the SAS log, it appears as a string of X characters (**`XXXXXXXXXX`**).

Appendix 2
Data Source Connection Arguments for SAS Data

Dictionary

SAS Data Set Connection Arguments

To access a SAS data set, you can submit a fully specified connection string, which defines how to connect to the data.

Syntax

CATALOG= *catalog-identifier*;

DRIVER=BASE;

SCHEMA=(*attributes*);

<ACCESS=READONLY | TEMP; >

<COMPRESS=NO | YES | CHAR | BINARY; >

<ENCODING=*encoding-value;*>

<LOCKTABLE=SHARE | EXCLUSIVE; >

NAME=*schema-identifier*;

PRIMARYPATH=*physical-location*;

Data Source Connection Arguments

CATALOG=*catalog-identifier*;
specifies the name of an SQL catalog that is defined for the BASE data service on the DataFlux Federation Server. A catalog groups logically related schemas. A catalog name can be up to 32 characters long.

Requirement: You must specify a catalog.

Note: You can specify more than one catalog. For example:

```
"driver=fedsql;
   conopts=((driver=base;catalog=basea;
      schema=(name=one;primarypath='c:\data\basedata'));
   (driver=base;catalog=baseb;
      schema=(name=two;primarypath='c:\myfiles')))";
```

DRIVER=BASE;

identifies the data source that you want to access, which is a SAS data set.

Requirement: You must specify DRIVER=BASE to access a SAS data set.

SCHEMA=(*attributes*);

specifies schema attributes that are specific to a SAS data set. A schema is a data container object that groups tables, which is unique within the catalog that qualifies table names. For a SAS data set, a schema is similar to a SAS library, which is a collection of tables and which has assigned attributes.

ACCESS=READONLY | TEMP;

READONLY

assigns a read-only attribute to the schema. You cannot open a SAS data set to update or write new information.

TEMP

specifies that the SAS data sets be treated as scratch files. That is, the system will not consume CPU cycles to ensure that the files do not become corrupted.

Interaction: This is an optional attribute.

Tip: Use ACCESS=TEMP to save resources only when the data is recoverable. If TEMP is specified, data in memory might not be written to disk on a regular basis. This saves I/O, but could cause a loss of data if there is a crash.

COMPRESS=NO | YES | CHAR | BINARY;

controls the compression of rows in created SAS data sets.

NO

specifies that the rows in a newly created SAS data set are uncompressed (fixed-length records). This is the default.

YES | CHAR

specifies that the rows in a newly created SAS data set are compressed (variable-length records) by using RLE (Run Length Encoding). RLE compresses rows by reducing repeated consecutive characters (including blanks) to two- or three-byte representations.

Tip: Use this compression algorithm for character data.

BINARY

specifies that the rows in a newly created SAS data set are compressed (variable-length records) by using RDC (Ross Data Compression). RDC combines run-length encoding and sliding-window compression to compress the file.

Tip: This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (numeric columns). Because the compression function operates on a single record at a time, the record length must be several hundred bytes or larger for effective compression.

Default: NO

See: For more information about compressing, see *SAS Language Reference: Concepts*

ENCODING=*encoding-value*;
: overrides and transcodes the encoding for input or output processing of SAS data sets.

Default: The default value is the current SAS session setting.

LOCKTABLE=SHARE | EXCLUSIVE
: places exclusive or shared locks on SAS data sets. You can lock tables only if you are the owner or have been granted the necessary privilege.

SHARE
: locks tables in shared mode, allowing other users or processes to read data from the tables, but preventing other users from updating.

EXCLUSIVE
: locks tables exclusively, preventing other users from accessing any table that you open.

Default: The default value is SHARE.

NAME=*schema-identifier*;
: specifies the name of an SQL schema that is defined on the DataFlux Federation Server. The schema identifier is an alias for the physical location of the SAS library, which is much like the Base SAS libref. A schema name must be a valid SAS name and can be up to 32 characters long.

Requirement: You must specify a schema identifier.

PRIMARYPATH=*physical-location*;
: specifies the physical location for the SAS library, which is a collection of one or more SAS files. For example, in directory-based operating environments, a SAS library is a group of SAS files that are stored in the same directory.

Requirement: You must specify a primary path.

Requirement: Include the attributes within parentheses, and separate each attribute with a semicolon. For example:

```
schema=(name=one;primarypath='c:\data\basedata')
```

Note: You can specify more than one schema. For example:

```
driver=base;catalog=base;
   schema=(name=one;primarypath='c:\data\basedata')
   schema=(name=two;primarypath='c:\temp')
   schema=(name=three;primarypath='c:\myfiles')
```

Appendix 3
Examples of a Fully-Specified Connection String

Access a SAS Data Set with a Fully-Specified Connection String

This example accesses a SAS data set by connecting to the DataFlux Federation Server and by submitting a fully-specified connection string. In the code, the following occurs:

1. The LIBNAME statement assigns the libref LIB1 and specifies the FEDSVR engine and DataFlux Federation Server server connection arguments.
2. The CONNECT_STRING= data source connection option provides a fully-specified connection string for a SAS data set, which includes specifying the BASE table driver and the location of the SAS library.

```
libname lib1 fedsvr server="d1235.us.company.com"
    port=2171 user="sasadm@saspw" pwd=1pwd 1
    connect_string="driver=base;2
    catalog=base;
    schema=(name=tsbase; primarypath='c:\myfiles\base')";
```

The following output shows the results of the LIBNAME statement in the SAS log:

Log A3.1 *Example of a library assignment message from the log*

```
7    libname lib1 fedsvr
8       server="d1234.us.company.com" protocol=com
9        user="sasadm@saspw" pwd=XXXXXX
10       connect_string=XXXXXXXXXXXXXX
11   XXXXXXXXXXXXXXXXX
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXX
12 ! XX;
NOTE: Libref LIB1 was successfully assigned as follows:
      Engine:        FEDSVR
      Physical Name:
```

Access an Oracle Database with a Fully-Specified Connection String

This example accesses an Oracle database by connecting to the DataFlux Federation Server and by submitting a fully-specified connection string.

1. The LIBNAME statement assigns the libref Myfiles and specifies the FEDSVR engine and Dataflux Federation Server server connection arguments.
2. The DRIVER=FEDSQL option specifies the language driver, which is required for an Oracle database when using the LIBNAME engine.
3. The CONOPTS= option specifies the connection string arguments for an Oracle database server.
4. The PRINT procedure prints an Oracle table.

```
libname myfiles fedsvr server=a123.us.company.com
   port=2171 user="sasadm@saspw" pwd=lpwd 1
   connect_string="driver=fedsql; 2
   conopts=(driver=oracle; uid=oracleid; pwd=oraclepw;
      path=or9010; catalog=cat_oracle;)"; 3

proc print data=myfiles.table1; 4
run;
```

Glossary

American National Standards Institute
the organization that coordinates the development of voluntary consensus standards for products, services, processes, systems, and personnel in the United States. ANSI works with the International Organization for Standardization to establish global standards. Short form: ANSI.

ANSI
See American National Standards Institute

authentication
See client authentication

Base SAS
the core product that is part of SAS Foundation and is installed with every deployment of SAS software. Base SAS provides an information delivery system for accessing, managing, analyzing, and presenting data.

catalog
See SQL catalog

client authentication
the process of verifying the identity of a person or process for security purposes.

column
a vertical component of a table. Each column has a unique name, contains data of a specific type, and has particular attributes. A column is analogous to a variable in SAS terminology.

connection string
information that defines how to connect an application to the data. In the DataFlux Federation Server, a connection string identifies the query language syntax that the application submits as well as the information that is required to connect to a data source or data sources.

data set
See SAS data set

data source
a table, view, or file from which you will extract information. Sources can be in any format that SAS can access, on any supported hardware platform. The metadata for a source is typically an input to a job.

data source name
a persistent identifier that is associated with a data source definition. The data source definition specifies how to locate and access a data source, including any authentication (such as a user name and password) that a user must provide. Short form: DSN.

data type
an attribute of every column in a table or database. The data type tells the operating system how much physical storage to set aside for the column, and specifies what type of data the column will contain. It is similar to the type attribute of SAS variables.

data view
See SAS data view

database
an organized collection of related data. A database usually contains named files, named objects, or other named entities such as tables, views, and indexes.

database management system
a software application that enables you to create and manipulate data that is stored in the form of databases. Short form: DBMS.

DataFlux Authentication Server
a component of the DataFlux Data Management Platform that provides a central location for the management of connections between DataFlux clients, DataFlux servers, and native database servers.

DBMS
See database management system

driver
a special-purpose software program that enables two disparate software programs such as an application and an API to interact.

DSN
See data source name

federated DSN
a data source name that references multiple data sources. The data sources can be on the same DBMS, or on a different one.

Federation Server authorization
an authorization process in which a database internal to the DataFlux Federation Server stores security definitions for users and objects. The process returns decisions on the actions that can be taken on resources, based on the requesting user's identity and group memberships.

grouping data source name
See federated DSN

grouping DSN
See federated DSN

library reference
See libref

libref
a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

metadata server
a server that provides metadata management services to one or more client applications. A SAS Metadata Server is an example.

missing value
a type of value for a variable that contains no data for a particular row or column. By default, SAS writes a missing numeric value as a single period and a missing character value as a blank space.

null value
a special value that indicates the absence of information. Null values are analogous to SAS missing values.

observation
a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

SAS data file
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views.

SAS data view
a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. Short form: data view.

SAS file
a specially structured file that is created, organized, and maintained by SAS. A SAS file can be a SAS data set, a catalog, a stored program, an access descriptor, a utility file, a multidimensional database file, a financial database file, a data mining database file, or an item store file.

SAS library
one or more files that are defined, recognized, and accessible by SAS, and that are referenced and stored as a unit. Each file is a member of the library.

SAS Management Console
a Java application that provides a single user interface for performing SAS administrative tasks.

SAS Metadata Server
a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

SAS variable
a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations (rows).

schema
See SQL schema

server
software that provides either resources or services to requesting clients, possibly over a network.

source
See data source

SQL
See Structured Query Language

SQL catalog
an implementation of the ANSI SQL:1999 standard for a named collection of logically related schemas. The catalog is the first-level (top) grouping mechanism in a data organization hierarchy that qualifies schemas.

SQL schema
an implementation of the ANSI SQL:1999 standard for a data container object that groups logically related objects such as tables and views and other objects that are supported by a data source such as stored procedures. The schema provides a unique namespace that is used along with a catalog to qualify names.

Structured Query Language
a standardized, high-level query language that is used in relational database management systems to create and manipulate objects in a database management system. SAS implements SQL through the SQL procedure. Short form: SQL.

table driver
software that interacts with a data source in order to read from and write to proprietary file formats.

type
See data type

variable
See SAS variable

Index