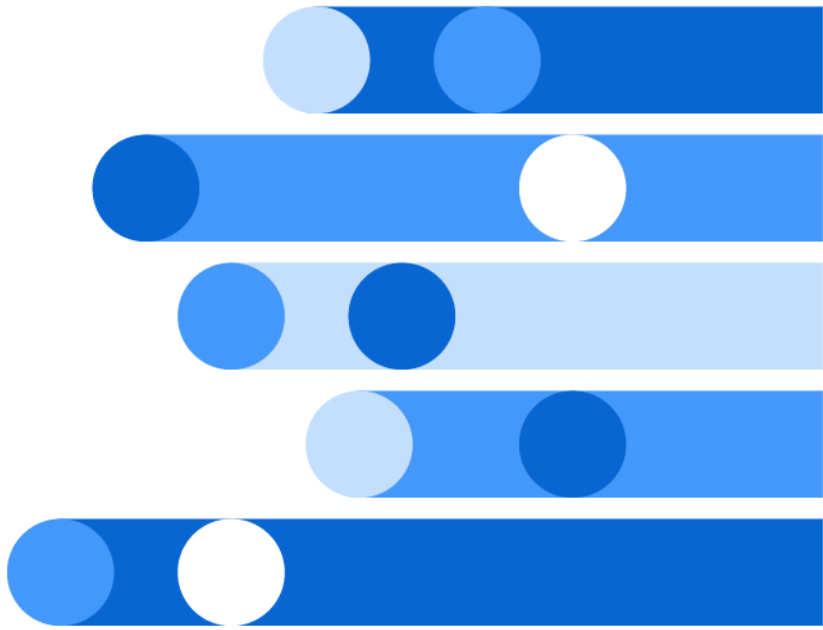
SAS® 9.4 Global Statements: Reference



SAS® Documentation
August 27, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS® 9.4 Global Statements: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Global Statements: Reference

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2024

9.4-P8:lestmtsglobal

Contents

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...
END;

Some system options require that one of two keyword values be specified:

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, ...>)

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument

pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...>;

argument-1 <*options*> <*argument-2* <*options*> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> ...>>;

argument-1=*value-1* <*argument-2*=*value-2* ...>
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1*=*label-1* <*variable-2*=*label-2* ...>;

argument-1 <, *argument-2*, ...>
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (*provider-1:domain-1* <, *provider-2:domain-2*, ...>

INTO :*macro-variable-specification-1* <, :*macro-variable-specification-2*, ...>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic
identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  LINK *label*;

- nonliteral values that are assigned to an argument.

  In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT *variable(s)* <*format* > <DEFAULT = *default-format*>;

Special Characters

The syntax of SAS language elements can contain the following special characters:

=
an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >
angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, ...>)

|
a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...
an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1* <, *item-2*, ...>)

'*value*' or "*value*"
indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
   length color name $8;
   color = 'black';
   name = 'jack';
   game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

What's New in SAS 9.4 Global Statements

Overview

This document supports global statements for SAS 9.4 and SAS Viya.

A highlighted, abbreviated notation of the SAS version and maintenance release specifies when a feature was added to SAS. For example, SAS 9.4M5 indicates that a feature was added during the fifth maintenance release of SAS 9.4.

These are the new SAS Viya 3.5 features, designated using the notation SAS Viya 3.5:

- The FILENAME statement Azure access method enables access to data in Microsoft Azure Data Lake Storage.
- The FILENAME statement S3 access method enables access to data in Amazon S3 files.

These are the new SAS Viya 3.4 features, designated using the notation SAS Viya 3.4:

- The LIBNAME statement now supports the LIBRARYDEFINITION option.
- The JSON LIBNAME statement now supports a NOALLDATA option.
- The LOCKDOWN statement for SAS Viya has moved to SAS Viya Administration: Programming Run-Time Servers. For more information, see SAS Viya LOCKDOWN Statement.

  For information about LOCKDOWN on SAS 9.4, see SAS 9.4 LOCKDOWN Statement.

Support for the FILENAME statement, FILESRVC access method is new for SAS Viya 3.3.

In the March 2024 update to SAS 9.4M8, the FILENAME Azure access method is enhanced to support all host environments supported by SAS, except z/OS.

These are the new and enhanced features for SAS 9.4M8:

- The FILENAME statement Azure access method is supported for SAS 9.4.

- The FILENAME statement S3 access method is supported for SAS 9.4.
- The TD_1MB_ROW environment variable, which specifies whether response row sizes up to 1MB are supported for data on Teradata, is new.

Beginning with SAS 9.4M5, DATA step statements are available in the new *SAS DATA Step Statements: Reference*.

These are the new and enhanced features for SAS 9.4M5:

- The FILENAME statement, ZIP access method supports the GZIP option to specify an external GZIP file.
- The FILENAME statement, EMAIL (SMTP) access method now supports attaching more than one file using multiple !EM_ATTACH! directives.
- The JSON LIBNAME statement now supports an ALLDATA="*name*" option.

These are the new and enhanced features for SAS 9.4M4:

- The FILENAME statement, EMAIL (SMTP) access method supports the SENSITIVITY= option to specify the sensitivity of an email message.
- The default behavior of the CFG= option has changed for the FILENAME statement, Hadoop access method. If CFG= is not provided, the SAS_HADOOP_CONFIG_PATH and SAS_HADOOP_JAR_PATH environment variables are scanned for the location of the required configuration files.
- The FILENAME statement, Hadoop access method supports Knox security.
- The JSON LIBNAME statement enables you to associate a libref with a JSON document.
- The CVP LIBNAME statement enables you to associate a libref with the character variable padding (CVP) engine.

These are the new and enhanced features for SAS 9.4M3:

- The FILENAME statement, FTP access method now supports Secure FTP using Transport Layer Security (TLS).
- The FILENAME statement, Hadoop access method now supports the SAS_HADOOP_CONFIG_PATH environment variable.
- Wildcards (*) are now supported in the FILENAME statement, ZIP access method's MEMBER= syntax for reading or checking the existence of entries in the ZIP file.
- For the FILENAME statement, ZIP access method, a new option, NAMEENCODING=, enables you to specify an encoding for ZIP file entry names and comments that is different from the current session encoding.

These are the new and enhanced features for SAS 9.4:

- read data from user-specified text and to access ZIP files
- specify the name of an authentication domain metadata object in order to connect to the WebDAV server, delete a directory and all of its members, and create a new directory off the parent directory when using the WebDAV access method

- automatically create SAS data files with an enhanced file format that extends the observation count beyond the 32-bit long limitation
- specify whether SAS creates compressed data sets whose observations can be randomly accessed or sequentially accessed
- specify an Accept: header and create a connection between the client and the proxy and between the proxy and the server when accessing a URL through a proxy when using the URL access method
- transfer data in image (binary) mode when using the SFTP access method
- control whether your SAS client has access to a set of directories and files
- embed attachments in an email using HTML. In addition, you can now specify a message/rfc822 content type.
- submit HDFS commands through WebHDFS
- open an AES (Advanced Encryption Standard) encrypted SAS data file

New SAS Statements

These SAS statements are new:

FILENAME Statement: Azure Access Method
: enables you to access data in Microsoft Azure Data Lake Storage.

FILENAME Statement: S3 Access Method
: enables you to access objects in the Simple Storage Service (S3) of Amazon Web Services (AWS).

FILENAME, FILESRVC Access Method
: enables you to store and access files within the SAS Viya file service.

FILENAME, DATAURL Access Method
: enables you to read data from user-specified text.

FILENAME, ZIP Access Method
: enables you to access ZIP files.

LIBNAME, JSON Engine
: associates a libref with a JSON data table and enables you to read JSON data tables.

LIBNAME, CVP Engine
: associates a libref with the character variable padding (CVP) engine to expand character variable lengths so that character data truncation does not occur when a file requires transcoding.

Enhanced SAS Statements

These SAS statements have been enhanced:

FILENAME, EMAIL (SMTP) Access Method

- In SAS 9.4M5, you can attach more than one file to an email using multiple ! EM_ATTACH! directives.
- In SAS 9.4M4, you can set an email sensitivity flag on emails that originate from SAS.
- In SAS 9.4M2, you can embed attachments in an email using HTML. In addition, you can now specify a message/rfc822 content type.
- The default time-out that the EMAIL access method waits for the SMTP server to respond is 30 seconds. Some SMTP servers require more time before they send an acknowledgment to a command from the client. You can use the new EMAILACKWAIT= system option to specify the wait time.
- You can use the EMAIL access method with secure SMTP servers by specifying either the new SSL or TLS protocol options in the EMAILHOST system option. TLS and SSL encrypt data between the client and the outgoing SMTP Server. This encryption does not guarantee an encrypted connection between the client (sender) and the recipient of the message. Message-level encryption and digital signing are currently not supported.

FILENAME, FTP Access Method

In SAS 9.4M3, the following enhancements were made:

- Filenames can contain UTF-8 characters. Only hosts whose FTP servers support the OPTS UTF8 ON or OPTS UTF-8 ON FTP protocol commands can read these filenames.
- The FTP access method now supports Secure FTP by using Transport Layer Security (TLS). Three new statement options, AUTHTLS, PBSZ=, and PROT=, enable you to issue the FTP AUTH TLS command, specify the FTP Data Channel Protection Buffer Size, and specify the FTP Data Channel security command, respectively. A new environment variable, SAS_FTP_AUTHTLS, lets you specify how TLS authentication is enabled.

FILENAME, Hadoop Access Method

- In SAS 9.4M4, the default behavior of the FILENAME statement, Hadoop access method CFG= option has changed. If CFG= is not provided, the SAS_HADOOP_CONFIG_PATH and SAS_HADOOP_JAR_PATH environment variables are scanned for the location of the required configuration files. In addition, Knox security is now supported.
- In SAS 9.4M3, the Hadoop access method now supports the SAS_HADOOP_CONFIG_PATH environment variable. You no longer have to merge properties from multiple Hadoop configuration files into a single

configuration file and specify the CFG= option. In addition, the Hadoop CONCAT= and DIR= options are now mutually exclusive because the SAS_HADOOP_CONFIG_PATH environment variable is available.

- In SAS 9.4M2, you can now submit HDFS commands through WebHDFS. The new SAS environment variable SAS_HADOOP_RESTFUL must be defined and set to the value 1. In addition, the Hadoop configuration file must include the properties for the WebHDFS location.
- A new option, NEW, is used in output mode in conjunction with the DIR option to create the directory that is specified in the FILENAME Hadoop statement.

FILENAME, SFTP Access Method

- Stream-record format has been added to the RECFM= option. Data is transferred in image (binary) mode. The amount of data that is read is controlled by the current LRECL value or by the value of the NBYTE= variable in the INFILE statement.
- A new option, OPTIONSX, enable you to submit private keys and passphrases that are blotted in the SAS log. Private keys and passphrases are necessary when you submit code that contains a FILENAME SFTP statement from SAS Enterprise Guide that runs on a Windows workspace server and authentication is required.

FILENAME, URL Access Method

- A new option, ACCEPT, specifies an Accept: header.
- A new option, CONNECT, creates a connection between the client and the proxy and between the proxy and the server when accessing a URL through a proxy.

FILENAME, WebDAV Access Method

- A new option, AUTHDOMAIN, specifies the name of an authentication domain metadata object in order to connect to the WebDAV server. The authentication domain references credentials (user ID and password) without your having to explicitly specify the credentials.
- A new option, DEL_ALL, enables you to delete a directory and all of its members.
- A new option, MKDIR, specifies a new directory that is created off the parent directory that was specified in the external file option.

FILENAME, ZIP Access Method

In SAS 9.4M5, you can use the GZIP option to specify an external GZIP file.

In SAS 9.4M3, the following enhancements were made:

- A new option, NAMEENCODING=, enables you to specify an encoding for ZIP file entry names and comments that is different from the current session encoding.
- Wildcards (*) are now supported in the MEMBER= syntax for reading or checking the existence of entries in the ZIP file.

LIBNAME

- In SAS Viya 3.4, a new library definition option, LIBRARYDEFINITION, specifies to generate and execute the LIBNAME statement using predefined option name/value pairs stored by the Data Sources microservice.
- The EXTENDOBSCOUNTER= option is now set to YES by default, which creates SAS data files with an enhanced file format. The enhanced file format extends the observation count beyond the 32-bit-long limitation.
- A new option, POINTOBS, specifies whether SAS creates compressed data sets whose observations can be randomly accessed or sequentially accessed.

LOCK

A new option, NOMSG, disables error and warning messages to the SAS log.

SASFILE

The new option ENCRYPTKEY= enables the SASFILE statement to open an AES (Advanced Encryption Standard) encrypted SAS data file.

Locked-Down State Restrictions

The LOCKDOWN statement and LOCKDOWN system option are new in SAS 9.4M1. With LOCKDOWN, if you are running in a client/server environment (for example, you use SAS Enterprise Guide), the SAS server administrator can create an environment where your SAS client has access to a set of directories and files. All other directories and files would be inaccessible. In addition to there being restrictions on directories and files, several language elements are not available when SAS is in a locked-down state.

In SAS 9.4M2, the following FILENAME statement access methods are not available when SAS is in a locked-down state:

- EMAIL (SMTP)
- FTP
- Hadoop
- SOCKET (TCPIP)
- URL (HTTP)

However, your server administrator can re-enable the access method so that it is accessible in the locked-down state. When the Hadoop and URL access methods are in a locked-down state, the HADOOP, HTTP, and SOAP procedures are also placed in a locked-down state. If the Hadoop or URL access method is re-enabled, the HADOOP, HTTP, and SOAP procedures are automatically re-enabled.

For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Language Reference: Concepts*.

1

About SAS Global Statements

Definition of Global Statements

A *SAS global statement* is a string of SAS keywords, SAS names, special characters, and operators that instructs SAS to perform an operation or that gives information to SAS. Global statements can also request information, change the execution of the program from one mode to another, or set values for system options.

Global statements can be specified in a SAS program in the following locations:

- open code
- a DATA step
- a PROC step
- a SAS macro

Notes:

Global statements are not executable statements; they are declarative statements that take effect as soon as SAS compiles the program statements.

Global statements cannot be specified in an IF-THEN/ELSE statement because IF-THEN/ELSE statements require executable statements.

Although it is syntactically valid to specify global statements in conditionally executed blocks of code (for example, in a DO group of an IF-THEN/ELSE statement), it is important to remember that global statements take effect during the compilation phase and that conditionally executed code blocks take effect during the execution phase. This means that global statements that are specified in conditionally executed code blocks take effect regardless of whether the condition is true.

Some Global ODS statements deliver output in a variety of formats such as HTML, PDF, and RTF.

Global statements are grouped by functionality in Global Statements by Category.

Using Global Statements

Global statements generally provide information to SAS, request information or data, move between different modes of execution, or set values for system options. Other global statements (ODS statements) deliver output in a variety of formats, such as in Hypertext Markup Language (HTML). You can use global statements anywhere in a SAS program. However, global statements are not supported on the CAS server. Global statements are not executable; they take effect as soon as SAS compiles program statements.

Global statements can be divided into functional categories. For a list of global statements by category, see "Global Statements by Category" on page 6.

Other SAS software products have additional statements that are used with those products. For more information, see "Other Statement Documentation" on page 2.

Other Statement Documentation

In addition to the statements documented in *SAS Global Statements: Reference*, statements are also documented in these publications:

- *SAS DATA Step Statements: Reference*
- *Base SAS Procedures Guide*
- *SAS Cloud Analytic Services: User's Guide*
- SAS Viya Administration: Programming Run-Time Servers
- *SAS Companion for Windows*
- *SAS Companion for UNIX Environments*
- *SAS Companion for z/OS*
- *SAS Language Interfaces to Metadata*
- *SAS Macro Language: Reference*
- *SAS Output Delivery System: User's Guide*
- *SAS Scalable Performance Data Engine: Reference*
- *SAS XMLV2 and XML LIBNAME Engines: User's Guide*

- *SAS/ACCESS for Relational Databases: Reference*
- *SAS/CONNECT User's Guide*
- *SAS/SHARE User's Guide*
- Application Messaging with SAS
- *SAS DS2 Language Reference*
- *SAS FedSQL Language Reference*
- SAS LIBNAME Engine for SAS Federation Server: User's Guide

2

Dictionary of SAS Global Statements

Global Statements by Category

This table lists and describes SAS global statements, organized by function into these categories:

Table 2.1 *Global Statements by Category*

Statements Category	Functionality
Action	Signals the end of data lines or acts as a placeholder. See Action for a list of statements.
Data Access	Associates reference names with SAS libraries, SAS catalogs, external files and output devices, and accesses remote files. See Data Access for a list of statements.
Information	Gives SAS additional information about the program data vector. See Information for a list of statements.
Log Control	Alters the appearance of the SAS log. See Log Control for a list of statements.
Output Control	Adds titles and footnotes to your SAS output; delivers output in a variety of formats. See Output Control for a list of statements.
Program Control	Governs how SAS processes your SAS program. See Program Control for a list of statements.

This table provides brief descriptions of SAS global statements. For more detailed information, see the individual statements.

Category	Language Elements	Description
Action	Null Statement (p. 196)	Signals the end of data lines or acts as a placeholder.
Data Access	CATNAME Statement (p. 9)	Logically combines two or more catalogs into one by associating them with a catref (a shortcut name); clears one or all catrefs; lists the concatenated catalogs in one concatenation or in all concatenations.
	FILENAME Statement (p. 19)	Associates a SAS fileref with an external file or an output device, disassociates a fileref and external file, or lists attributes of external files.
	FILENAME Statement: Azure Access Method (p. 30)	Enables you to access data in Microsoft Azure Data Lake Storage.
	FILENAME Statement: CATALOG Access Method (p. 32)	Enables you to reference a SAS catalog as an external file.
	FILENAME Statement: CLIPBOARD Access Method (p. 36)	Enables you to read text data from and write text data to the clipboard on the host computer.
	FILENAME Statement: DATAURL Access Method (p. 39)	Enables you to read data from user-specified text.
	FILENAME Statement: EMAIL (SMTP) Access Method (p. 42)	Enables you to send electronic mail programmatically from SAS using the SMTP (Simple Mail Transfer Protocol) email interface.
	FILENAME Statement: FILESRVC Access Method (p. 59)	Enables you to store and retrieve user content using the SAS Viya Files service.
	FILENAME Statement: FTP Access Method (p. 67)	Enables you to access remote files by using the FTP protocol.
	FILENAME Statement: Hadoop Access Method (p. 86)	Enables you to access files on a Hadoop Distributed File System (HDFS) whose location is specified in a configuration file.
	FILENAME Statement: S3 Access Method (p. 92)	Enables you to access Amazon S3 files.
	FILENAME Statement: SFTP Access Method (p. 95)	Enables you to access remote files by using the SFTP protocol.
	FILENAME Statement: SOCKET Access Method (p. 103)	Enables you to read from or write to a TCP/IP socket.
	FILENAME Statement: URL Access Method (p. 108)	Enables you to access remote files by using the URL access method.

Category	Language Elements	Description
	FILENAME Statement: WebDAV Access Method (p. 114)	Enables you to access remote files by using the WebDAV protocol.
	FILENAME Statement: ZIP Access Method (p. 123)	Enables you to access ZIP files.
	LIBNAME Statement (p. 139)	Associates or disassociates a SAS library with a libref (a shortcut name), clears one or all librefs, lists the characteristics of a SAS library, concatenates SAS libraries, or concatenates SAS catalogs.
	LIBNAME Statement: CVP Engine (p. 157)	Associates a libref for the character variable padding (CVP) engine to expand character variable lengths so that character data truncation does not occur when a file requires transcoding.
	LIBNAME Statement: JMP Engine (p. 162)	Associates a libref with a JMP data table and enables you to read and write JMP data tables.
	LIBNAME Statement: JSON Engine (p. 164)	Provides read-only sequential access to JSON data.
	LIBNAME Statement: WebDAV Server Access (p. 184)	Associates a libref with a SAS library and enables access to a WebDAV (Web-based Distributed Authoring And Versioning) server.
Information	MISSING Statement (p. 194)	Assigns characters in your input data to represent special missing values for numeric data.
Log Control	Comment Statement (p. 14)	Specifies the purpose of the statement or program.
	PAGE Statement (p. 199)	Skips to a new page in the SAS log.
	RESETLINE Statement (p. 200)	Restarts the program line numbers in the SAS log to 1.
	SKIP Statement (p. 211)	Creates a blank line in the SAS log.
Output Control	FOOTNOTE Statement (p. 128)	Writes up to 10 lines of text at the bottom of the procedure or DATA step output.
	TITLE Statement (p. 212)	Specifies title lines for SAS output.
Program Control	CHECKPOINT EXECUTE_ALWAYS Statement (p. 13)	Indicates to execute the DATA step or PROC step that immediately follows without considering the checkpoint-restart data.
	DM Statement (p. 16)	Enables you to turn SAS Command Line commands into SAS global programming statements in the SAS Display Manager environment.
	ENDSAS Statement (p. 18)	Stops SAS program execution as soon as the statement is encountered in a SAS program.

Category	Language Elements	Description
	%INCLUDE Statement (p. 132)	Brings a SAS programming statement, data lines, or both, into a current SAS program.
	%LIST Statement (p. 189)	Displays lines that are entered in the current session.
	LOCK Statement (p. 191)	Acquires, lists, or releases an exclusive lock on an existing SAS file.
	OPTIONS Statement (p. 198)	Specifies or changes the value of one or more SAS system options.
	RUN Statement (p. 201)	Executes the previously entered SAS statements.
	%RUN Statement (p. 203)	Ends source statements following a %INCLUDE * statement.
	SASFILE Statement (p. 204)	Opens a SAS data set and allocates enough buffers to hold the entire file in memory.
	SYSECHO Statement (p. 212)	Sends a global statement complete event and passes a text string back to the IOM client.

Dictionary

CATNAME Statement

Logically combines two or more catalogs into one by associating them with a catref (a shortcut name); clears one or all catrefs; lists the concatenated catalogs in one concatenation or in all concatenations.

Valid in: Anywhere

Category: Data Access

Syntax

CATNAME <*libref.*> *catref*

< (*libref-1.catalog-1* <(ACCESS=READONLY)>

<...*libref-n.catalog–n* <(ACCESS=READONLY)>>)>;

CATNAME <*libref.*> *catref* CLEAR | _ALL_ CLEAR;

CATNAME <*libref.*> *catref* LIST | _ALL_ LIST;

Arguments

libref
is any previously assigned SAS libref. If you do not specify a libref, SAS concatenates the catalog in the Work library, using the catref that you specify.

Range 1 to 8 bytes

Restriction The libref must have been previously assigned.

catref
is a unique catalog reference name for a catalog or a catalog concatenation that is specified in the statement. Separate the catref from the libref with a period, as in *libref.catref*. Any SAS name can be used for this catref.

catalog
is the name of a catalog that is available for use in the catalog concatenation.

Options

CLEAR
disassociates a currently assigned *catref* or *libref.catref*.

Tip Specify a specific *catref* or *libref.catref* to disassociate it from a single concatenation. Specify _ALL_ CLEAR to disassociate all currently assigned *catref* or *libref.catref* concatenations.

ALL CLEAR
disassociates all currently assigned *catref* or *libref.catref* concatenations.

LIST
writes the catalog names that are included in the specified concatenation to the SAS log.

Tip Specify *catref* or *libref.catref* to list the attributes of a single concatenation. Specify _ALL_ to list the attributes of all catalog concatenations in your current session.

ALL LIST
writes all catalog names that are included in any current catalog concatenation to the SAS log.

ACCESS=READONLY
assigns a read-only attribute to the catalog. SAS allows users to read from the catalog entries but not to update information or to write new information.

Details

Why Use CATNAME?

CATNAME is useful because it enables you to access entries in multiple catalogs by specifying a single catalog reference name (*libref.catref* or *catref*). After you create a catalog concatenation, you can specify the catref in any context that accepts a simple (non-concatenated) catref.

Rules for Catalog Concatenation

To use catalog concatenation effectively, you must understand the rules that determine how catalog entries are located among the concatenated catalogs:

- When a catalog entry is opened for input or update, the concatenated catalogs are searched and the first occurrence of the specified entry is used.
- When a catalog entry is opened for output, it is created in the first catalog that is listed in the concatenation.

  Note: A new catalog entry is created in the first catalog even if there is an entry with the same name in another part of the concatenation.

  Note: If the first catalog in a concatenation that is opened for update does not exist, the item is written to the next catalog that exists in the concatenation.

- When you want to delete or rename a catalog entry, only the first occurrence of the entry is affected.
- Anytime a list of catalog entries is displayed, only one occurrence of a catalog entry name is shown.

  Note: Even if the name occurs multiple times in the concatenation, only the first occurrence is shown.

Comparisons

- The CATNAME statement is like a LIBNAME statement for catalogs. The LIBNAME statement enables you to assign a shortcut name to a SAS library so that you can use the shortcut name to find the files and use the data that they contain. CATNAME enables you to assign a short name *<libref.>catref* (libref is optional) to one or more catalogs so that SAS can find the catalogs and use all or some of the entries in each catalog.
- The CATNAME statement *explicitly* concatenates SAS catalogs. You can use the LIBNAME statement to *implicitly* concatenate SAS catalogs.

Examples

Example 1: Assigning and Using a Catalog Concatenation

You might need to access entries in several SAS catalogs. The most efficient way to access the information is to logically concatenate the catalogs. Catalog concatenation enables access to the information without actually creating a new, separate, and possibly very large catalog.

Assign librefs to the SAS libraries that contain the catalogs that you want to concatenate:

```
libname mylib1 'data-library-1';
libname mylib2 'data-library-2';
```

Assign a catref, which can be any valid SAS name, to the list of catalogs that you want to logically concatenate:

```
catname allcats (mylib1.catalog1 mylib2.catalog2);
```

The SAS log displays this message:

Example Code 2.1 *Log Output from CATNAME Statement*

```
NOTE: Catalog concatenation WORK.ALLCATS has been created.
```

Because no libref is specified, the libref is Work by default. When you want to access a catalog entry in either of these catalogs, use the libref Work and the catalog reference name ALLCATS instead of the original librefs and catalog names. For example, to access a catalog entry named APPKEYS.KEYS in the catalog MYLIB1.CATALOG1, specify

```
work.allcats.appkeys.keys
```

Example 2: Creating a Nested Catalog Concatenation

After you create a concatenated catalog, you can use CATNAME to combine your concatenation with other single catalogs or other concatenated catalogs. Nested catalog concatenation is useful, because you can use a single catref to access many different catalog combinations.

```
libname local 'my_dir';
libname main  'public_dir';
catname private_catalog (local.my_application_code
                         local.my_frames
                         local.my_formats);
catname combined_catalogs (private_catalog
                         main.public_catalog);
```

In the above example, you could work on private copies of your application entries by using PRIVATE_CATALOG. If you want to see how your entries function when they are combined with the public version of the application, you can use COMBINED_CATALOGS.

See Also

Statements:

- "FILENAME Statement" on page 19
- "FILENAME Statement: CATALOG Access Method" on page 32
- "LIBNAME Statement" on page 139 for a discussion of implicitly concatenating SAS catalogs

CHECKPOINT EXECUTE_ALWAYS Statement

Indicates to execute the DATA step or PROC step that immediately follows without considering the checkpoint-restart data.

Valid in: Anywhere

Category: Program Control

Syntax

CHECKPOINT EXECUTE_ALWAYS;

Without Arguments

The CHECKPOINT EXECUTE_ALWAYS statement indicates to SAS that the DATA step or PROC step that immediately follows is to be executed without considering the checkpoint data.

Details

If checkpoint-restart mode is enabled and a batch program terminates without completing, the program can be rerun beginning with the DATA step or PROC step that was executing when it terminated. DATA or PROC steps that completed before the batch program terminated are not reexecuted. If a DATA step or a PROC step must be reexecuted, you can add the CHECKPOINT EXECUTE_ALWAYS statement before the step. Using the CHECKPOINT EXECUTE_ALWAYS statement ensures that SAS always executes the step without regard to the checkpoint-restart data.

See Also

- "Checkpoint Mode and Restart Mode" in *SAS Language Reference: Concepts*

System Options:

- "STEPCHKPT System Option" in *SAS System Options: Reference*
- "STEPCHKPTLIB= System Option" in *SAS System Options: Reference*
- "STEPRESTART System Option" in *SAS System Options: Reference*

Comment Statement

Specifies the purpose of the statement or program.

Valid in: Anywhere

Category: Log Control

Syntax

**message*;

or

*/*message*/*

Arguments

****message*;**
specifies the text that explains or documents the statement or program.

Range: These comments can be any length and are terminated with a semicolon.

Restrictions: These comments must be written as separate statements.

These comments cannot contain internal semicolons.

A macro statement or macro variable reference that is contained inside this form of comment is processed by the SAS macro facility. This form of comment cannot be used to hide text from the SAS macro facility.

Tip: When using comments within a macro definition or to hide text from the SAS macro facility, use this style comment:

```
/* message */
```

/*message*/
specifies the text that explains or documents the statement or program.

Range: These comments can be any length.

Restriction: This type of comment cannot be nested.

Windows specifics

If you use the Enhanced Editor, you can comment out a block of code by highlighting the block and then pressing CTRL-/ (forward slash). To uncomment a block of code, highlight the block and press CTRL-SHIFT-/ (forward slash).

Tips

These comments can contain semicolons and unmatched quotation marks.

You can write these comments within statements or anywhere a single blank can appear in your SAS code.

Details

You can use the comment statement anywhere in a SAS program to document the purpose of the program, explain unusual segments of the program, or describe steps in a complex program or calculation. SAS ignores text in comment statements during processing.

CAUTION

Avoid placing the /* comment symbols in columns 1 and 2. In some operating environments, SAS might interpret a /* in columns 1 and 2 as a request to end the SAS program or session.

Note: You can add these lines to your code to fix unmatched comment tags, unmatched quotation marks, and missing semicolons.

```
/* '; * "; */;
quit;
run;
```

Example: Using the Comment Statement

These examples illustrate the two types of comments:

- This example uses the **message*; format:

```
*This code finds the number in the BY group;
```

- This example uses the **message*; format:

```
*--------------------------------------*
|  This uses one comment statement     |
|           to draw a box.             |
*--------------------------------------*;
```

- This example uses the */*message*/* format:

```
input @1 name $20.  /* last name    */
      @200 test 8.  /* score test   */
      @50 age 3.;   /* customer age */
```

- This example uses the /*message*/ format:

```
/* For example 1 use: x=abc;
   for example 2 use: y=ghi; */
```

DM Statement

Enables you to turn SAS **Command Line** commands into SAS global programming statements in the SAS Display Manager environment.

Valid in: Anywhere

Category: Program Control

See: SAS Command Line
"Commands under UNIX" in *SAS Companion for UNIX Environments*
"SAS Commands under Windows" in *SAS Companion for Windows*

Syntax

DM <*window*> '*command(s)*' <*window*> <CONTINUE> ;

Arguments

window
specifies the active window.

Default If you omit the window name, SAS uses the Program Editor window as the default.

Example
```
dm log 'clear';
```

'command(s)'
can be any windowing environment command or text editor command and must be enclosed in single quotation marks. If you want to issue several commands, separate them with semicolons.

See "Commands under UNIX" in *SAS Companion for UNIX Environments*, "SAS Commands under Windows" in *SAS Companion for Windows*, and SAS Windowing Environment.

CONTINUE
causes SAS to execute any SAS statements that follow the DM statement in the Program Editor window and, if a windowing command in the DM statement called a window, makes that window active.

Note For example, if you specify Log as the active window and have other SAS statements that follow the DM statement (for example, in an autoexec

file), those statements are not submitted to SAS until control returns to the SAS interface.

Tip Any windows that are activated by the SAS statements (such as the Output window) appear before the window that is to be made active.

Details

The SAS Display Manager (DM) is also known as the SAS Windowing Environment.

For a list of SAS commands under the Windows operating environment, see "SAS Commands under Windows" in *SAS Companion for Windows*. For a list of SAS commands under the UNIX operating environment, see "Commands under UNIX" in *SAS Companion for UNIX Environments*.

Execution occurs when the DM statement is submitted to SAS. You can use this statement to modify the windowing environment:

- Change SAS interface features during a SAS session.
- Change SAS interface features at the beginning of each SAS session by placing the DM statement in an autoexec file.
- Perform utility functions in windowing applications, such as saving a file with the FILE command or clearing a window with the CLEAR command.

Window placement affects the outcome of the statement:

- If you name a window before the commands, those commands apply to that window.
- If you name a window after the commands, SAS executes the commands and then makes that window the active window. The active window is opened and contains the cursor.

Examples

Example 1: Using the DM Statement

- `dm 'color text cyan; color command red';`
- ```
  dm log 'clear; pgm; color numbers green'
     output;
  ```
- `dm 'caps on';`
- `dm log 'clear' output;`
  ```

Example 2: Using the CONTINUE Option with SAS Statements That Do Not Activate a Window

This example causes SAS to display the first window of the SAS/AF application, executes the DATA step, moves the cursor to the first field of the SAS/AF application window, and makes that window active.

```
dm 'af c=your-program' continue;
data temp;
   . . . more SAS statements . . .
run;
```

Example 3: Using the CONTINUE Option with SAS Statements That Activate a Window

This example displays the first window of the SAS/AF application and executes the PROC PRINT step, which activates the Output window. Closing the Output window moves the cursor to the last active window.

```
dm 'af c=your-program' continue;
proc print data=temp;
run;
```

Example 4: Using the DM Statement to Display the Results Viewer Window

This examples causes SAS to display the Results Viewer window. You can also define a function key to perform this action.

```
dm 'next "results viewer"' continue;
```

ENDSAS Statement

Stops SAS program execution as soon as the statement is encountered in a SAS program.

Valid in: Anywhere

Category: Program Control

Syntax

ENDSAS;

Without Arguments

The ENDSAS statement can be specified in either a DATA step or a PROC step. When the DATA or PROC step runs, the ENDSAS statement stops program execution as soon as the statement is encountered in a SAS program.

Details

Note: ENDSAS statements are always executed at the point that they are encountered in a DATA step. Use the ABORT RETURN statement to stop processing when an error condition occurs (for example, in the clause of an IF-THEN statement or a SELECT statement).

Comparisons

You can also terminate a SAS job or session by using the BYE or the ENDSAS command from any SAS window command line. For more information, see the online Help for SAS windows.

See Also

"SYSSTARTID Automatic Macro Variable" in *SAS Macro Language: Reference*

FILENAME Statement

Associates a SAS fileref with an external file or an output device, disassociates a fileref and external file, or lists attributes of external files.

Valid in:	Anywhere
Category:	Data Access
Restrictions:	On Windows, the fully qualified path to the external file (including the name of the external file) cannot exceed 260 bytes in length. For more information, see "Referencing Files Using UNC Paths" in *SAS Companion for Windows*. When SAS is in a locked-down state, the FILENAME statement is not available for files that are not in the lockdown path list. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Language Reference: Concepts*.
Windows specifics:	The fully qualified path to the external file (including the name of the external file) cannot exceed 260 bytes in length. For more information, see "Referencing Files Using UNC Paths" in *SAS Companion for Windows*.
See:	FILENAME Statement under Windows, UNIX, and z/OS "FILENAME Statement: Windows" in *SAS Companion for Windows* "FILENAME Statement: UNIX" in *SAS Companion for UNIX Environments* "FILENAME Statement: z/OS" in *SAS Companion for z/OS*

Syntax

Form 1: **FILENAME** *fileref* <*device-type*> '*external-file*' <ENCODING='*encoding-value*'> <*options*> <*operating-environment-options*>;

Form 2: **FILENAME** *fileref* <*device-type*> <*options*> <*operating-environment-options* >;

Form 3: **FILENAME** *fileref* CLEAR | _ALL_ CLEAR;

Form 4: **FILENAME** *fileref* LIST | _ALL_ LIST ;

Arguments

fileref

is any SAS name that you use when you assign a new fileref. When you disassociate a currently assigned fileref or when you list file attributes with the FILENAME statement, specify a fileref that was previously assigned with a FILENAME statement or an operating environment-level command.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it by using another FILENAME statement. Change the fileref for a file as often as you want.

device-type

specifies the type of device or the access method that is used if the fileref points to an input or output device or location that is not a physical file:

device-type
ACTIVEMQ

specifies an access method that enables you to access an ActiveMQ messaging broker.

Restriction This device type is not supported in SAS Viya.

Interaction If the DATA step does not recognize the access method option, the DATA step passes the option to the access method for handling.

See "FILENAME Statement: ACTIVEMQ Access Method" in *Application Messaging with SAS*

AZURE

specifies an access method that enables you to access data in Microsoft Azure Data Lake Storage.

See "FILENAME Statement: Azure Access Method" on page 30

CATALOG

specifies an access method that enables you to reference a SAS catalog as an external file.

See "FILENAME Statement: CATALOG Access Method" on page 32

DATAURL
specifies an access method that enables you to read data from user-specified text.

See "FILENAME Statement: DATAURL Access Method" on page 39

DISK
specifies that the device is a disk drive.

Tip When you assign a fileref to a file on disk, you are not required to specify DISK.

DUMMY
specifies that the output to the file is discarded.

Tip Specifying DUMMY can be useful for testing.

EMAIL
specifies an access method that enables you to send electronic mail programmatically from SAS by using the SMTP (Simple Mail Transfer Protocol) email interface.

See "FILENAME Statement: EMAIL (SMTP) Access Method" on page 42

FTP
specifies an access method that enables you to access remote files by using the FTP protocol.

See "FILENAME Statement: FTP Access Method" on page 67

GTERM
indicates that the output device type is a graphics device that receives graphics data.

HADOOP
specifies an access method that enables you to access files on a Hadoop Distributed File System (HDFS) whose location is specified in a configuration file.

See "FILENAME Statement: Hadoop Access Method" on page 86

JMS
specifies a Java Message Service (JMS) destination.

Restriction This device type is not supported in SAS Viya.

PIPE
specifies an unnamed pipe.

Note Some operating environments do not support pipes.

PLOTTER
specifies an unbuffered graphics output device.

PRINTER
specifies a printer or printer spool file.

S3
specifies an access method that enables you to access Amazon S3 files.

See "FILENAME Statement: S3 Access Method" on page 92

SFTP
specifies an access method that enables you to access remote files by using the SFTP protocol.

See "FILENAME Statement: SFTP Access Method" on page 95

SOCKET
specifies an access method that enables you to read from or write to a TCP/IP socket.

See "FILENAME Statement: SOCKET Access Method" on page 103

TAPE
specifies a tape drive.

TEMP
creates a temporary file that exists only as long as the filename is assigned. The temporary file can be accessed only through the logical name and is available only while the logical name exists.

Restriction Do not specify a physical pathname. If you do, SAS returns an error.

Tip Files manipulated by the TEMP device can have the same attributes and behave identically to DISK files.

TERMINAL
specifies the user's terminal.

UPRINTER
specifies a Universal Printing printer definition name.

Tip If you do not specify the printer name in the FILENAME statement, the PRINTERPATH options control which Universal Printer is used and the destination of the output.

URL
specifies an access method that enables you to access remote files by using the URL access method.

See "FILENAME Statement: URL Access Method" on page 108

WEBDAV
specifies an access method that enables you to access remote files by using the WebDAV protocol.

See "FILENAME Statement: WebDAV Access Method" on page 114

ZIP
specifies an access method that enables you to access ZIP files.

See "FILENAME Statement: ZIP Access Method" on page 123

Requirement *device-type* must immediately follow *fileref* in the statement.

Operating environment Additional specifications might be required when you specify some devices. See the SAS documentation for your operating environment before specifying a value other than DISK. Values in addition to the ones listed here might be available in some operating environments.

See "FILENAME Statement: SFTP Access Method" on page 95

'external-file'
is the physical name of an external file. Enclose external filename in quotation marks. The physical name is the name that is recognized by the operating environment.

Restrictions On Windows, the fully qualified path to the external file (including the name of the external file) cannot exceed 260 bytes in length. For more information, see "Referencing Files Using UNC Paths" in *SAS Companion for Windows*.

Operating environment For more information about specifying the physical names of external files, see the SAS documentation for your operating environment. "Referencing External Files" in *SAS Companion for Windows*, "Specifying Pathnames in UNIX Environments" in *SAS Companion for UNIX Environments*, and "Specifying Physical Files" in *SAS Companion for z/OS*

Tips Specify *external-file* when you assign a fileref to an external file.

You can associate a fileref with a single file or with an aggregate file storage location by specifying the fully qualified pathname.

ENCODING= *'encoding-value'*
specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

Default SAS assumes that an external file is in the same encoding as the session encoding.

Restrictions The UPRINTER device type does not support the ENCODING= argument.

Not all device types support the encoding option. For more information, see the documentation for your operating system.

You cannot use the FILENAME statement to specify an encoding for a transport file that is created with PROC CPORT. In order for a transport file to be imported successfully, the encodings of the source and target SAS sessions must be compatible.

See For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide* .

Examples "Example 5: Specifying an Encoding When Reading an External File" on page 28

"Example 6: Specifying an Encoding When Writing to an External File" on page 29

CLEAR
disassociates one or more currently assigned filerefs.

Tip Specify *fileref* to disassociate a single fileref. Specify _ALL_ to disassociate all currently assigned filerefs.

ALL
specifies that the CLEAR or LIST argument applies to all currently assigned filerefs.

LIST
writes the attributes of one or more files to the SAS log.

Interaction Specify *fileref* to list the attributes of a single file. Specify _ALL_ to list the attributes of all files that have filerefs in your current session.

Options

RECFM=*record-format*
specifies the record format of the external file.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

Operating environment Values for *record-format* are dependent on the operating environment. For more information, see the SAS documentation for your operating environment.

Operating Environment Options

Operating environment options specify details, such as file attributes and processing attributes, that are specific to your operating environment.

Operating Environment Information: For a list of valid specifications, see the SAS documentation for your operating environment.

Details

Operating Environment Information

Operating Environment Information: Using the FILENAME statement requires operating environment-specific information. See the SAS documentation for your operating environment before using this statement. Note also that commands are available in some operating environments that associate a fileref with a file and that break that association.

Definitions

external file
: is a file that is created and maintained in the operating environment from which you need to read data, SAS programming statements, or autocall macros, or to which you want to write output. An external file can be a single file or an aggregate storage location that contains many individual external files. See "Example 3: Associating a Fileref with an Aggregate Storage Location" on page 27.

  Operating Environment Information: Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set. For more information about specifying external files, see the SAS documentation for your operating environment.

fileref
: (a file reference name) is a shorthand reference to an external file. After you associate a fileref with an external file, you can use it as a shorthand reference for that file in SAS programming statements (such as INFILE, FILE, and %INCLUDE) and in other commands and statements in SAS software that access external files.

Reading Delimited Data from an External File

Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option in either EBCDIC or ASCII environments.

For example, when you read an EBCDIC text file on an ASCII platform, it is recommended that you specify the ENCODING= option in the FILENAME statement. However, if you use the DSD and the DLM or DLMSTR= options in the INFILE statement, the ENCODING= option is a requirement because these options must contain certain characters in the session encoding (such as quotation marks, commas, and blanks).

The use of encoding-specific informats should be reserved for use with true binary files. That is, the files contain both character and non-character fields.

Associating a Fileref with an External File (*Form 1*)

Use this form of the FILENAME statement to associate a fileref with an external file on disk:

FILENAME *fileref 'external-file'* <*operating-environment-options*>;

To associate a fileref with a file other than a disk file, you might need to specify a device type, depending on your operating environment, as shown in this form:

FILENAME *fileref* <*device-type*> <*operating-environment-options*>;

The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. Change the fileref for a file as often as you want.

To specify a character-set encoding, use this form:

FILENAME *fileref* <*device-type*> <*operating-environment-options*>;

Associating a Fileref with a Terminal, Printer, Universal Printer, or Plotter (*Form 2*)

To associate a fileref with an output device, use this form:

FILENAME *fileref device-type* <*operating-environment-options*>;

Disassociating a Fileref from an External File (*Form 3*)

To disassociate a fileref from a file, use a FILENAME statement that specifies the fileref and the CLEAR option.

FILENAME *fileref* CLEAR | _ALL_ CLEAR;

Writing File Attributes to the SAS Log (*Form 4*)

Use a FILENAME statement to write the attributes of one or more external files to the SAS log. Specify *fileref* to list the attributes of one file; use _ALL_ to list the attributes of all the files that have been assigned filerefs in your current SAS session.

FILENAME *fileref* LIST | _ALL_ LIST;

Comparisons

The FILENAME statement assigns a fileref to an external file. The LIBNAME statement assigns a libref to a SAS library. Use the LIBNAME, SAS/ACCESS statement to access DBMS tables.

Examples

Example 1: Specifying a Fileref or a Physical Filename

You can specify an external file either by associating a fileref with the file and then specifying the fileref or by specifying the physical filename in quotation marks:

```
filename sales 'your-input-file';
data jansales;
      /* specifying a fileref */
   infile sales;
   input salesrep $20. +6 jansales febsales
         marsales;
run;
data jansales;
      /* physical filename in quotation marks */
   infile 'your-input-file';
   input salesrep $20. +6 jansales febsales
         marsales;
run;
```

Example 2: Using a FILENAME and a LIBNAME Statement

This example reads data from a file that has been associated with the fileref GREEN and creates a permanent SAS data set stored in a SAS library that has been associated with the libref SAVE.

```
filename green 'your-input-file';
libname save 'SAS-library';
data save.vegetable;
   infile green;
   input lettuce cabbage broccoli;
run;
```

Example 3: Associating a Fileref with an Aggregate Storage Location

If you associate a fileref with an aggregate storage location, use the fileref, followed in parentheses by an individual filename, to read from or write to any of the individual external files that are stored there.

Operating Environment Information: Some operating environments enable you to read from but not write to members of aggregate storage locations. For more information, see the SAS documentation for your operating environment.

In this example, each DATA step reads from an external file (REGION1 and REGION2, respectively) that is stored in the same aggregate storage location and that is referenced by the fileref SALES.

```
filename sales 'aggregate-storage-location';
data total1;
   infile sales(region1);
   input machine $ jansales febsales marsales;
   totsale=jansales+febsales+marsales;
run;
```

```
data total2;
   infile sales(region2);
   input machine $ jansales febsales marsales;
   totsale=jansales+febsales+marsales;
run;
```

Example 4: Routing PUT Statement Output

In this example, the FILENAME statement associates the fileref OUT with a printer that is specified with an operating environment-dependent option. The FILE statement directs PUT statement output to that printer.

```
filename out printer operating-environment-option;
data sales;
   file out print;
   input salesrep $20. +6 jansales
         febsales marsales;
   put _infile_;
   datalines;
Jones, E. A.              124357 155321 167895
Lee, C. R.                111245 127564 143255
Desmond, R. T.             97631 101345 117865
;
```

You can use the FILENAME and FILE statements to route PUT statement output to several devices during the same session. To route PUT statement output to your display monitor, use the TERMINAL option in the FILENAME statement, as shown here:

```
filename show terminal;
data sales;
   file show;
   input salesrep $20. +6 jansales
         febsales marsales;
   put _infile_;
   datalines;
Jones, E. A.              124357 155321 167895
Lee, C. R.                111245 127564 143255
Desmond, R. T.             97631 101345 117865
;
```

Example 5: Specifying an Encoding When Reading an External File

This example creates a SAS data set from an external file. The external file is in UTF-8 character-set encoding, and the current SAS session is in the Wlatin1 encoding. By default, SAS assumes that an external file is in the same encoding as the session encoding, which causes the character data to be written to the new SAS data set incorrectly.

To tell SAS what encoding to use when reading the external file, specify the ENCODING= option. When you tell SAS that the external file is in UTF-8, SAS then transcodes the external file from UTF-8 to the current session encoding when writing to the new SAS data set. Therefore, the data is written to the new data set correctly in Wlatin1.

```
libname myfiles  'SAS-library';

filename extfile 'external-file' encoding="utf-8";
data myfiles.unicode;
   infile extfile;
   input Make $ Model $ Year;
run;
```

Note: You cannot use the FILENAME statement to specify an encoding for a transport file that is created with PROC CPORT. In order for a transport file to be imported successfully, the encodings of the source and target SAS sessions must be compatible.

Example 6: Specifying an Encoding When Writing to an External File

This example creates an external file from a SAS data set. The current session encoding is Wlatin1, but the external file's encoding needs to be UTF-8. By default, SAS writes the external file using the current session encoding.

To tell SAS what encoding to use when writing data to the external file, specify the ENCODING= option. When you tell SAS that the external file is to be in UTF-8 encoding, SAS then transcodes the data from Wlatin1 to the specified UTF-8 encoding when writing to the external file.

```
libname myfiles  'SAS-library';
filename outfile 'external-file' encoding="utf-8";

data _null_;
   set myfiles.cars;
   file outfile;
   put Make Model Year;
run;
```

See Also

Statements:

- "FILE Statement" in *SAS DATA Step Statements: Reference*
- "%INCLUDE Statement" on page 132
- "INFILE Statement" in *SAS DATA Step Statements: Reference*
- "FILENAME Statement: CATALOG Access Method" on page 32
- "FILENAME Statement: ACTIVEMQ Access Method" in *Application Messaging with SAS*
- "FILENAME Statement: DATAURL Access Method" on page 39
- "FILENAME Statement: EMAIL (SMTP) Access Method" on page 42
- "FILENAME Statement: FTP Access Method" on page 67

- "FILENAME Statement: Hadoop Access Method" on page 86
- "FILENAME Statement: JMS Access Method" in *Application Messaging with SAS*
- "FILENAME Statement: SFTP Access Method" on page 95
- "FILENAME Statement: SOCKET Access Method" on page 103
- "FILENAME Statement: URL Access Method" on page 108
- "FILENAME Statement: WebDAV Access Method" on page 114
- "FILENAME Statement: ZIP Access Method" on page 123
- "LIBNAME Statement" on page 139
- "LOCKDOWN Statement" in SAS Intelligence Platform: Application Server Administration Guide

System Options:

- "LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide

SAS Windowing Interface Commands:

- See the FILE and INCLUDE commands in the Base SAS Help and Documentation

FILENAME Statement: Azure Access Method

Enables you to access data in Microsoft Azure Data Lake Storage.

Category: Data Access

Restrictions: Support for the Azure access method in SAS 9 begins in SAS 9.4M8.
The Azure access method is not supported on z/OS platforms.

See: "AZUREAUTHCACHELOC System Option" in *SAS System Options: Reference*
Process for Device Code Authorization
"AZURETENANTID= System Option" in *SAS System Options: Reference*

Syntax

FILENAME *fileref* ADLS *"object path"* <*adls-options*>;

Arguments

fileref
is a valid fileref.

ADLS
specifies the Azure Data Lake Storage (ADLS) access method.

object-path
specifies the Azure object that you want to access.

Required Options

The Azure access method accepts these options:

ACCOUNTNAME="*account name*"
specifies the name of the Azure storage account.

APPLICATIONID="*application id*"
specifies the Azure application ID that is created in the Microsoft Entra ID (formerly Azure Active Directory).

FILESYSTEM="*file system*"
specifies the name of the Azure file system.

Details

Creating an Azure Account

Azure Data Lake Storage (ADLS) is a Microsoft cloud computing service. The FILENAME statement Azure access method enables you to access your data on ADLS.

To use ADLS, set up an account with Microsoft Azure. For information about setting up an account, see the Microsoft Azure portal. You also have to set up your system to use OAuth authentication tokens. Contact your system administrator for information about using OAuth tokens.

Example: Obtaining an Authorization Device Code to Access ADLS Data

This example shows the process for Microsoft device-code authentication and authorization the first time you run a SAS program to access data on a Microsoft ADLS server. If you already have a JSON credentials file, then this authentication and authorization step is not required. For more information, see "Details" in *SAS System Options: Reference*. Note the following key points:

- The first time you run a program to access data on a Microsoft ADLS server, SAS generates the following ERROR messages in the SAS log:

```
ERROR: Cannot obtain connection to ADLS.
ERROR: To sign in, use a web browser to open the page https://microsoft.com/devicelogin
       and enter the code xxxxxx to authenticate.
```

- The messages provide instructions for authenticating with the Microsoft ADLS server.

1 Use a browser to open the Microsoft device authorization page.

2 Enter the code that is provided in the SAS log. After you enter the code and get the required authorization, you can successfully run the program without errors.

- SAS maintains an authorization file (`.sasadls_userid.json`) in your **$HOME** directory that contains the credentials needed for future data access. After you have completed the authorization process, you do not have to repeat it unless the JSON file is deleted or your credentials change.
- The following program creates a file named `example.txt` on the ADLS server and writes a line of text to the file:

Note: You can specify the AZUREAUTHCACHELOC system option to change the location of the JSON file.

```
options azuretenantid = "user-tenant-id";   /*1*/

filename out adls "path/example.txt"
   applicationid="application-id"
   accountname="account-name"
   filesystem="filesystem-name";

data _null_;
   file out;
   put 'line 1';
run;
```

1 Specify your AZURETENANTID system option tenant ID. Run the program and complete the authorization and authentication process by opening the web page that is referenced in the SAS log and enter the device code. After you have completed the authorization process, re-submit the SAS program. The program creates a file named `example.txt` on the ADLS server and writes a line of text to the file.

Example Code 2.2 *Log Output*

```
ERROR: Cannot obtain connection to ADLS. Check options and tokens.
ERROR: To sign in, use a web browser to open the page
     https://microsoft.com/devicelogin and enter the code <CODE> to authenticate.
```

FILENAME Statement: CATALOG Access Method

Enables you to reference a SAS catalog as an external file.

Valid in: Anywhere

Category: Data Access

Syntax

FILENAME *fileref* CATALOG '*catalog*' <*catalog-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

CATALOG
specifies the access method that enables you to reference a SAS catalog as an external file. You can then use any SAS commands, statements, or procedures that can access external files to access a SAS catalog.

Alias LIBRARY

Tips This access method makes it possible for you to invoke an autocall macro directly from a SAS catalog.

With this access method, you can read any type of catalog entry, but you can write only to entries of type LOG, OUTPUT, SOURCE, and CATAMS.

If you want to access an entire catalog (instead of a single entry), you must specify its two-level name in the *catalog* parameter.

'catalog'
is a valid two-, three-, or four-part SAS catalog name, where the parts represent *library.catalog.entry.entrytype*.

Default The default entry type is CATAMS.

Restriction The CATAMS entry type is used only by the CATALOG access method. The CPORT and CIMPORT procedures do not support this entry type.

Catalog Options

catalog-options can be any of these values:

LRECL=*lrecl*
where *lrecl* is the maximum record length for the data in bytes.

Default For input, the actual LRECL value of the file is the default. For output, the default is 132.

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

RECFM=*recfm*
where *recfm* is one of four record formats:

F
is a fixed-record format. Data is transferred in image (binary) mode.

P
is a print format.

S
is a stream-record format. Data is transferred in image (binary) mode.

Interactions The amount of data that is read is controlled by the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL.

In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

See The NBYTE= option in the INFILE statement.

V
is a variable-record format (the default). In this format, records have varying lengths, and the records are separated by newlines.

Default V

DESC=*description*
where *description* is a text description of the catalog.

MOD
specifies to append to the file.

Default If you omit MOD, the file is replaced.

Details

The CATALOG access method in the FILENAME statement enables you to reference a SAS catalog as an external file. You can then use any SAS commands, statements, or procedures that can access external files to access a SAS catalog. For example, the catalog access method makes it possible for you to invoke an autocall macro directly from a SAS catalog. See "Example 5: Executing an Autocall Macro from a SAS Catalog" on page 36.

With the CATALOG access method, you can read any type of catalog entry, but you can write to only entries of type LOG, OUTPUT, SOURCE, and CATAMS. If you want to access an entire catalog (instead of a single entry), you must specify its two-level name in the *catalog* argument.

Examples

Example 1: Using %INCLUDE with a Catalog Entry

This example submits the source program that is contained in SASUSER.PROFILE.SASINP.SOURCE:

```
filename fileref1
         catalog 'sasuser.profile.sasinp.source';
%include fileref1;
```

Example 2: Using %INCLUDE with Several Entries in a Single Catalog

This example submits the source code from three entries in the catalog MYLIB.INCLUDE. When no entry type is specified, the default is CATAMS.

```
   filename dir catalog 'mylib.include';
   %include dir(mem1);
   %include dir(mem2);
   %include dir(mem3);
```

Example 3: Reading and Writing a CATAMS Entry

This example uses a DATA step to write data to a CATAMS entry, and another DATA step to read it back in:

```
filename mydata
         catalog 'sasuser.data.update.catams';
   /* write data to catalog entry update.catams */
data _null_;
   file mydata;
   do i=1 to 10;
      put i;
   end;
run;
   /* read data from catalog entry update.catams */
data _null_;
   infile mydata;
   input;
   put _INFILE_;
run;
```

Example 4: Writing to a SOURCE Entry

This example writes code to a catalog SOURCE entry and then submits it for processing:

```
filename incit
         catalog 'sasuser.profile.sasinp.source';
data _null_;
   file incit;
   put 'proc options; run;';
run;
%include incit;
```

Example 5: Executing an Autocall Macro from a SAS Catalog

If you store an autocall macro in a SOURCE entry in a SAS catalog, you can point to that entry and invoke the macro in a SAS job. Use these steps:

1. Store the source code for the macro in a SOURCE entry in a SAS catalog. The name of the entry is the macro name.
2. Use a LIBNAME statement to assign a libref to that SAS library.
3. Use a FILENAME statement with the CATALOG specification to assign a fileref to the catalog: *libref.catalog*.
4. Use the SASAUTOS= option and specify the fileref so that the system knows where to locate the macro. Also set MAUTOSOURCE to activate the autocall facility.

This example points to a SAS catalog named MYSAS.MYCAT. It then invokes a macro named REPORTS, which is stored as a SAS catalog entry named MYSAS.MYCAT.REPORTS.SOURCE:

```
libname mysas 'SAS-library';
filename mymacros catalog 'mysas.mycat';
options sasautos=mymacros mautosource;
%reports
```

See Also

Statements:

- "FILENAME Statement" on page 19

FILENAME Statement: CLIPBOARD Access Method

Enables you to read text data from and write text data to the clipboard on the host computer.

Valid in: Anywhere

Category: Data Access

Syntax

FILENAME *fileref* CLIPBRD <BUFFER=*paste-buffer-name*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

CLIPBRD
specifies the access method that enables you to read data from or write data to the clipboard on the host computer.

BUFFER=*paste-buffer-name*
creates and names the paste buffer. You can create any number of paste buffers by naming them with the BUFFER= argument in the STORE command.

Details

The FILENAME statement, CLIPBOARD Access Method enables you to share data within SAS and between SAS and applications other than SAS.

Comparisons

The STORE command copies marked text in the current window and stores the copy in a paste buffer.

You can also copy data to the clipboard by using the **Explorer** pop-up menu item **Copy Contents to Clipboard** .

Examples

Example 1: Using ODS to Write a Data Set as HTML to the Clipboard

This example uses the Sashelp.Air data set as the input file. The ODS is used to write the data set in HTML format to the clipboard.

```
filename _temp_ clipbrd;
   ods noresults;
   ods html file=_temp_ rs=none style=minimal;
   proc print data=Sashelp.'Air'N noobs;
run;
ods results;
filename _temp_;
```

Example 2: Using the DATA Step to Write a Data Set as Comma-separated Values to the Clipboard

This example uses the Sashelp.Air data set as the input file. The data is written in the DATA step as comma-separated values to the clipboard.

```
filename _temp1_ temp;
filename _temp2_ clipbrd;
proc contents data=Sashelp."Air"N out=info noprint;
proc sort data=info;
   by npos;
run;
data _null_;
   set info end=eof;
   ;
   file _temp1_ dsd;
   put name @@;
   if _n_=1 then do;
      call execute("data _null_;
      set Sashelp.""Air""N;
      file _temp1_ dsd mod;
      put");
   end;
   call execute(trim(name));
   if eof then call execute('; run;');
run;
data _null_;
   infile _temp1_;
   file _temp2_;
   input;
   put _infile_;
run;
filename _temp1_ clear;
filename _temp2_ clear;
```

Example 3: Using the DATA Step to Write Text to the Clipboard

This example writes three lines to the clipboard.

```
filename clippy clipbrd;
data _null_;
   file clippy;
   put 'Line 1';
   put 'Line 2';
   put 'Line 3';
run;
```

Example 4: Using the DATA Step to Retrieve Text from the Clipboard

This example writes three lines to the clipboard and then retrieves them.

```
filename clippy clipbrd;
data _null_;
   file clippy;
   put 'Line 1';
```

```
   put 'Line 2';
   put 'Line 3';
run;
data _null_;
   infile clippy;
   input;
   put _infile_;
run;
```

See Also

Commands:

- The STORE command in the Base SAS Help and Documentation

Statements:

- "FILENAME Statement" on page 19

FILENAME Statement: DATAURL Access Method

Enables you to read data from user-specified text.

Valid in: Anywhere

Category: Data Access

Syntax

FILENAME *fileref* **DATAURL** '*data-url-specification*' <*data-url-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

DATAURL
specifies the access method that enables you to read data from *data-url-specification*.

'data-url-specification'
specifies the data.

Requirement The data must be in one of these formats:

- data:, *file-data*, where data can consist of characters and URL-encoded

characters. Examples are `%20` and `%00`. **`data`** must be lowercase.

- data: ;base64, *base64-data*, where data consists of base64-encoded data. **`data`** must be lowercase.

Example "Example 2: Base64 Encoded Data" on page 41

data-url-options

can be any of these values:

LRECL=*record-length*

where *lrecl* is the logical record length of the data.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

RECFM=*recfm*

where *recfm* is one of three record formats:

F

is a fixed-record format. Thus, all records are of size LRECL with no line delimiters. Data is transferred in image (binary) mode.

S

is a stream-record format. Data is transferred in image (binary) mode.

V

is a variable-record format (the default). In this format, records have varying lengths, and the records are transferred in text mode.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

Details

The DATAURL access method is similar to the URL access method. The DATAURL access method reads small amounts of data directly from a data URL specification instead of reading data from a network location.

Multiple lines of data can be read from *data-url-specification*. A null byte is a line delimiter.

Examples

Example 1: Accessing Simple Data

This example accesses three lines of data by using `%00` as URL-encoded null bytes to terminate each line.

```
filename in dataurl "data:,line one%00line two%00line three%00";
  data _NULL_;
    infile in;
    input;
    list;
  run;
```

```
NOTE: The infile IN is:
      Filename=data:,line one%00line two%00line three%00,
      Lrecl=256,Recfm=Variable

RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+---
1         line one 8
2         line two 8
3         line three 10
NOTE: 3 records were read from the infile IN.
      The minimum record length was 8.
      The maximum record length was 10.
```

Example 2: Base64 Encoded Data

This example accesses data by using base64-encoded data.

```
filename in dataurl "data:;base64,dGhpcyBpcyBhIGJhc2UgNjQgZW5jb2RpbmcgZXhhbXBsZS4=" ;
  data _NULL_;
    infile in;
    input;
    list;
  run;
```

```
NOTE: The infile IN is:
      Filename=data:;base64,dGhpcyBpcyBhIGJhc2UgNjQgZW5jb2RpbmcgZXhhbXBsZS4=,
      Lrecl=256,Recfm=Variable

RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+---
1         this is a base 64 encoding example. 35
NOTE: 1 record was read from the infile IN.
      The minimum record length was 35.
      The maximum record length was 35.
```

See Also

Statements:

- “FILENAME Statement” on page 19

FILENAME Statement: EMAIL (SMTP) Access Method

Enables you to send electronic mail programmatically from SAS using the SMTP (Simple Mail Transfer Protocol) email interface.

Valid in: Anywhere

Category: Data Access

Restriction: When SAS is in a locked-down state, the FILENAME statement, EMAIL access method is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Language Reference: Concepts*.

Syntax

FILENAME *fileref* EMAIL < '*address*' > <*email-options*>;

Arguments

fileref
is a valid file reference. The fileref is a name that is temporarily assigned to an external file or to a device type. Note that the fileref cannot exceed eight bytes.

Range 1 to 8 bytes

EMAIL
specifies the EMAIL device type, which provides the access method that enables you to send electronic mail programmatically from SAS. In order to use SAS to send a message to an SMTP server, you must enable SMTP email. For more information, see "The SMTP E-Mail Interface" in *SAS Language Reference: Concepts*.

'address'
is the email address to which you want to send the message. You must enclose the address in single or double quotation marks. To specify more than one address, you must enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name along with an address, enclose the address in angle brackets (< >). Specifying an address as a FILENAME statement argument is optional if you specify the TO= email option or the PUT statement !EM_TO! directive, which overrides an *address* specification.

Email Options

You can use any of these email options in the FILENAME statement to specify attributes for the electronic message. You can also specify these options in the FILE statement. Email options that you specify in the FILE statement override any corresponding email options that you specified in the FILENAME statement.

ATTACH='*filename.ext*' | ATTACH= ('*filename.ext*' *attachment-options*)
specifies the physical name of the file or files to be attached to the message and any options to modify attachment specifications. The physical name is the name that is recognized by the operating environment. Enclose the physical name in quotation marks. To attach more than one file, enclose the group of files in parentheses, enclose each file in quotation marks, and separate each with a space. Here are examples:

```
attach="/u/userid/opinion.txt"
attach=('C:\Status\June2001.txt' 'C:\Status\July2001.txt')
attach="user.misc.pds(member)"
```

The *attachment-options* include these values:

CONTENT_TYPE='*content/type*'
specifies the content type for the attached file. You must enclose the value in quotation marks. If you do not specify a content type, SAS tries to determine the correct content type based on the filename. For example, if you do not specify a content type, a filename of **`home.html`** is sent with a content type of text/html.

Alias CT= and TYPE=

Default If SAS cannot determine a content type based on the filename and extension, the default value is text/plain.

ENCODING='*encoding-value*'
specifies the text encoding of the attachment that is read into SAS. You must enclose the value in quotation marks.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

EXTENSION='*extension*'
specifies a different file extension to be used for the specified attachment. You must enclose the value in quotation marks. This extension is used by the recipient's email program for selecting the appropriate utility to use for displaying the attachment. This example results in the attachment **`home.html`** being received as **`index.htm`**.

```
attach=("home.html" name="index" ext="htm")
```

Alias EXT=

Note If you specify *extension=""*, the specified attachment has no file extension.

INLINED="*reference-name*"
specifies a reference name that can be used to embed attachments in an email by using HTML. To embed an attachment, set `content_type="text/`

`html";` and reference the attachment from the email text with `SRC="cid:`*`reference-name`*`"`. The *reference-name* is specified with the INLINE= option.

An example is `attach=("image.jpg" inlined="myimage")`. From the email text, you reference this image by using `<IMG SRC="cid:myimage">`.

Note If the image is not referenced, then the recipients see it as a regular attachment.

See "Example 6: Creating an Email with an Embedded Image" on page 58

LRECL=*lrecl*
where *lrecl* is the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

NAME='*filename*'
specifies a different name to be used for the specified attachment. You must enclose the value in quotation marks. This example results in the attachment **`home.html`** being received as **`index.html`**.

```
attach=("home.html" name="index")
```

OUTENCODING='*encoding-value*'
specifies the resulting text encoding for the attachment to be sent. You must enclose the value in quotation marks.

Restriction Do not specify EBCDIC encoding values, because the SMTP email interface does not support EBCDIC.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

BCC='*bcc-address*'
specifies the recipient or recipients that you want to receive a blind carbon copy of the email. Individuals that are listed in the **`bcc`** field receive a copy of the email. The BCC field does not appear in the email header, so that these email addresses cannot be viewed by other recipients.

If a BCC address contains more than one word, then enclose the address in single or double quotation marks. To specify more than one address, you must enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >). Here are examples:

```
bcc="joe@site.com"
bcc=("joe@site.com" "jane@home.net")
bcc="Joe Smith <joe@site.com>"
```

Range 1–255 characters

CC='*cc-address*'
specifies the recipient or recipients to receive a carbon copy of the email message. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >). Here are examples:

```
cc='joe@site.com'
cc=("joe@site.com" "jane@home.net")
cc="Joe Smith <joe@site.com>"
```

Range 1–255 characters

CONTENT_TYPE='*content/type*'
specifies the content type for the message body. If you do not specify a content type, SAS tries to determine the correct content type. You must enclose the value in quotation marks.

If you do not specify a content type, SAS uses the default **'text/plain'**. When you use **'message/rfc822'**, SAS lets you create the entire email. SAS includes only the FROM, SUBJECT, DATE, and other email options that you specify that are not part of the MIME standard and that are expected to be in the email before the first MIME header. Creating the entire email enables you to send HTML files and to format emails in any way that you want.

Alias CT= and TYPE=

Default text/plain

See "Example 5: Using the MESSAGE/RFC822 Content Type" on page 58

DELIVERYRECEIPT
specifies that a notification be sent when the email message is delivered to the recipient.

Note If the recipient's email client does not support "delivery receipt" requests or if the recipient does not allow these requests, the sender does not receive a "delivery receipt" notification when the email is delivered.

ENCODING='*encoding-value*'
specifies the text encoding to use for the message body. For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

EXPIRES='*dd mon yyyy hh:mm*'
specifies the expiration date for the email message.

The format *dd mon hh:mm* parameters are defined as follows:

dd
is an integer from 01 to 31 that represents the day of the month.

mon
are the first three letters of the month name in English.

yyyy
is a four-digit integer that represents the year.

hh
is the number of hours that range from 00 through 23.

mm
is the number of minutes that range from 00 through 59.

Tip If the date and time have passed the current date and time, an error message occurs and no email is sent.

FROM='*from-address*'
specifies the email address of the author of the message that is being sent. Specify this option when the person who is sending the message is not the author. You must enclose an address in quotation marks. You can specify only one email address. To specify the author's real name along with the address, enclose the address in angle brackets (< >). Here are examples:

```
from='martin@home.com'
from="Brad Martin <martin@home.com>"
```

Default The default value for FROM= is the email address of the user who is running SAS. Beginning in SAS 9.4M6, if the SENDER= option is specified, the default value for FROM= is the email address that is specified in the SENDER= option.

Range 1–255 characters

Requirement The FROM option is required if the EMAILFROM system option is set. For more information, see "EMAILFROM System Option" in *SAS System Options: Reference*.

Interaction Use the SENDER= option to specify a return email address that is different from the author-specified email address in the FROM= option.

See "SENDER='*sender-address*'" on page 47

IMPORTANCE='LOW" | NORMAL" | HIGH'
specifies the priority of the email message. You must enclose the value in quotation marks. You can specify the priority in the language that matches your session encoding. However, SAS translates the priority into English because the actual message header must contain English in accordance with the RFC-2076 specification (Common Internet Message Headers). Here are examples:

```
filename inventory email 'name@mycompany.com' importance='high';
filename inventory email 'name@mycompany.com' importance='hoch';
```

Default NORMAL

LRECL=*lrecl*
where *lrecl* is the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

READRECEIPT
specifies that a notification be sent when the email message is read by the recipient.

Note If the recipient's email client does not support "read receipt" requests or if the recipient does not allow these return requests, the sender does not receive a "read receipt" notification when the recipient reads the email.

REPLYTO='*replyto-address*'
specifies the email address or addresses of who receives replies. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name along with an address, enclose the address in angle brackets (< >). Here are examples:

```
replyto='hiroshi@home.com'
replyto=('hiroshi@home.com' 'akiko@site.com')
replyto="Hiroshi Mori <mori@site.com>"
```

Range 1–255 characters

SENDER='*sender-address*'
specifies the return email address for the message that is being sent. If a message cannot be delivered, a notification is sent to the sender email address. To specify the author's real name along with the address, enclose the address in angle brackets (< >). Here are examples:

```
sender='martin@home.com'
sender='Brad Martin <martin@home.com>'
```

Default The default value for SENDER= is the email address of the user who is running SAS. Beginning in SAS 9.4M6, if the FROM= option is specified, the default value for SENDER= is the email address that is specified in the FROM= option.

Range 1–255 characters

Interaction The SENDER= address can be different from the FROM= address, which allows for the message to be sent on behalf of the email address that is specified in the FROM= option.

See "FROM='*from-address*'" on page 46

SENSITIVITY='NORMAL' | PRIVATE' | PERSONAL' | CONFIDENTIAL'' | COMPANY'

specifies the sensitivity of the email message. You must enclose the value in single quotation marks. Here is an example that results in a confidential header being added to the email message:

```
FILENAME mymail EMAIL TO='to-address' SUBJECT='a subject line'
SENSITIVITY='CONFIDENTIAL';
```

Default NORMAL

Notes If NORMAL is specified, no sensitivity header is added to the message.

The flags "company-confidential" and "confidential" are equivalent and have the same effect. The "company-confidential" flag is in compliance with RFC 2156.

SUBJECT=*subject*

specifies the subject of the message. If the subject contains special characters or more than one word (that is, it contains at least one blank space), you must enclose the text in quotation marks. Here are examples:

```
subject=Sales
subject="June Sales Report"
```

Note If you do not enclose a one-word subject in quotation marks, it is converted to uppercase.

TO='*to-address*'

specifies the primary recipient or recipients of the email message. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >). Here are examples:

```
to='joe@site.com'
to=("joe@site.com" "jane@home.net")
to="Joe Smith <joe@site.com>"
```

Range 1–255 characters

Tip Specifying TO= overrides the '*address*' argument.

PUT Statement Email Directives

The directives that you can specify in a PUT statement to change the attributes of a message are as follows:

'!EM_ABORT!'

abnormally end the current message. You can use this directive to stop SAS from automatically sending the message at the end of the DATA step. By default, SAS sends a message for each FILE statement.

'!EM_ATTACH! *filename.ext*' | '!EM_ATTACH! (*"filename.ext" attachment-options)*'
replaces the physical name of the file or files to be attached to the message and any options to modify attachment specifications. The physical name is the name that is recognized by the operating environment. The directive must be enclosed in quotation marks and contain a maximum of 256 characters.

To attach more than one file, enclose the group of files in parentheses, enclose each file in single or double quotation marks, and separate each file with either a comma or a space. To add *attachment-options*, enclose the file and the *attachment-options* in parentheses, and enclose the file in single or double quotation marks. Here is an example:

```
put '!em_attach! ("C:\Status\June2001.txt" "C:\Status\July2001.txt")';
```

In SAS 9.4M5, you can also attach more than one file using multiple ! EM_ATTACH! directives. Here is an example:

```
put '!em_attach! opinion.txt';
put '!em_attach! report.html';
```

The *attachment-options* include these values:

CONTENT_TYPE='*content/type*'
specifies the content type for the attached file. You must enclose the value in quotation marks. If you do not specify a content type, SAS tries to determine the correct content type based on the filename. For example, if you do not specify a content type, a filename of **`home.html`** is sent with a content type of text/html. Here is an example:

```
put "!em_attach! ('small.png' ct='image/png')";
```

Alias CT= and TYPE=

Default If SAS cannot determine a content type based on the filename and extension, the default value is text/plain.

ENCODING='*encoding-value*'
specifies the text encoding to use for the attachment as it is read into SAS. You must enclose the value in quotation marks. For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

EXTENSION='*extension*'
specifies a different file extension to be used for the specified attachment. You must enclose the value in quotation marks. This extension is used by the recipient's email program for selecting the appropriate utility to use for displaying the attachment. This example results in the attachment **`home.html`** being received as **`index.htm`**.

```
put '!em_attach! ("home.html" name="index" ext="htm")';
```

Alias EXT=

Default TXT

NAME='*filename*'
specifies a different name to be used for the specified attachment. You must enclose the value in quotation marks. This example results in the attachment **`home.html`** being received as **`index.html`**.

```
put '!em_attach! ("home.html" name="index")';
```

LRECL=*lrecl*
where *lrecl* is the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

OUTENCODING='*encoding-value*'
specifies the resulting text encoding for the attachment to be sent. You must enclose the value in quotation marks.

Restriction Do not specify EBCDIC encoding values, because the SMTP email interface does not support EBCDIC.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

'!EM_BCC! *bcc-address*'
specifies the recipient or recipients that you want to receive a blind carbon copy of the email. Individuals that are listed in the **`bcc`** field receive a copy of the email. The BCC field does not appear in the email header, so that these email addresses cannot be viewed by other recipients.

If a BCC address contains more than one word, then enclose the address in single or double quotation marks. To specify more than one address, you must enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >).

```
put '!em_bcc! joe@site.com';
put '!em_bcc! ("joe@site.com" "jane@home.net")';
put '!em_bcc! Joe Smith <joe@site.com>';
```

Range 1–255 characters

'!EM_CC! *cc-address*'
specifies the recipient or recipients to receive a carbon copy of the email message. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >). Here are examples:

```
put '!em_cc! joe@site.com';
```

```
put '!em_cc! ("joe@site.com" "jane@home.com")';
put '!em_cc! Joe Smith <joe@site.com>';
```

Range 1–255 characters

'!EM_CONTENTTYPE! *content/type*'
specifies the content type for the attached file. If you do not specify a content type, SAS tries to determine the correct content type based on the filename. For example, if you do not specify a content type, a filename of **`home.html`** is sent with a content type of text/html.

Alias !EM_CT! and !EM_TYPE!

Default If SAS cannot determine a content type based on the filename and extension, the default value is text/plain.

'!EM_DELIVERYRECEIPT!'
specifies that a notification be sent when the email message is delivered to the recipient.

Note If the recipient's email client does not support "delivery receipt" requests or if the recipient does not allow these requests, the sender does not receive a "delivery receipt" notification when the email is delivered.

'!EM_EXPIRES! *dd mon yyyy hh:mm*'
replaces the current expiration date for the email message. Here are examples:

```
put '!em_expires! 15 Aug 2010 08:00';
put '!em_expires! 28 Feb 2011 23:00';
```

The format *dd mon hh:mm* parameters are defined as follows:

dd
is an integer from 01 to 31 that represents the day of the month.

mon
are the first three letters of the month name in English.

yyyy
is a four-digit integer that represents the year.

hh
is the number of hours that range from 00 through 23.

mm
is the number of minutes that range from 00 through 59.

Tip If the date and time have passed the current date and time, an error message occurs and no email is sent.

'!EM_FROM! *from-address*'
replaces the current address of the author of the message being sent, which could be either the default or the address that is specified by the FROM= email option. The directive must be enclosed in quotation marks. You can specify only one email address. To specify the author's real name along with the address, enclose the address in angle brackets (< >). Here are examples:

```
put '!em_from! martin@home.com';
```

```
put '!em_from! Brad Martin <martin@home.com>';
```

Default The default value for !EM_FROM! is the email address of the user who is running SAS. Beginning in SAS 9.4M6, if the SENDER= option or the !EM_SENDER! option is specified, the default !EM_FROM! value is the email address that is specified in the SENDER= option or the !EM_SENDER! option.

Range 1–255 characters

Interaction Use the !EM_SENDER! option to specify a return email address that is different from the author-specified email address in the !EM_FROM! option.

See "'!EM_SENDER! *sender-address*'" on page 53

'!EM_IMPORTANCE! LOW | NORMAL | HIGH'
specifies the priority of the email message. The directive must be enclosed in quotation marks. You can specify the priority in the language that matches your session encoding. However, SAS translates the priority into English because the actual message header must contain English in accordance with the RFC-2076 specification (Common Internet Message Headers). Here are examples:

```
put '!em_importance! high';
put '!em_importance! haut';
```

Default NORMAL

'!EM_NEWMSG!'
clears all attributes of the current message that were set using PUT statement directives.

'!EM_READRECEIPT!'
specifies that a notification be sent when the email message is read by the recipient.

Note If the recipient's email client does not support "read receipt" requests or if the recipient does not allow these requests, the sender does not receive a "read receipt" notification when the recipient reads the email.

'!EM_REPLYTO! *replyto-address*'
specifies the email address or addresses of who receives replies. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name along with an address, enclose the address in angle brackets (< >). Here are examples:

```
put '!em_replyto! hiroshi@home.com';
put '!em_replyto! ("hiroshi@home.com" "akiko@site.com")';
put '!em_replyto! Hiroshi Mori <mori@site.com>';
```

Range 1–255 characters

'!EM_SEND!'
sends the message with the current attributes. By default, SAS sends a message when the fileref is closed. The fileref closes when the next FILE statement is encountered or the DATA step ends. If you use this directive, SAS sends the message when it encounters the directive, and again at the end of the DATA step. This directive is useful for writing DATA step programs that conditionally send messages or use a loop to send multiple messages.

'!EM_SENDER! *sender-address*'
specifies the return email address for the message that is being sent. If a message cannot be delivered, a notification is sent to the sender email address. The default value for !EM_SENDER! is the email address of the user who is running SAS. To specify the author's real name along with the address, enclose the address in angle brackets (< >). Here are examples:

```
put '!EM_SENDER! martin@home.com';
put '!EM_SENDER! Brad Martin <martin@home.com>';
```

Default The default value for !EM_SENDER! is the email address of the user who is running SAS. Beginning in SAS 9.4M6, if the FROM= option or the !EM_FROM! option is specified, the default !EM_SENDER! value is the email address that is specified in the FROM= option or the !EM_FROM! option.

Range 1–255 characters

Interaction The !EM_SENDER!= address can be different from the FROM= or !EM_FROM!= address, which allows for the message to be sent on behalf of the email address that is specified in the FROM= or !EM_FROM!= option.

See "'!EM_FROM! *from-address*'" on page 51

'!EM_SENSITIVITY! NORMAL | PRIVATE | PERSONAL | CONFIDENTIAL | COMPANY-CONFIDENTIAL'
marks the email message with the specified sensitivity. For example:

```
put '!EM_SENSITIVITY! CONFIDENTIAL';
```

Notes If NORMAL is specified, no sensitivity header is added to the message.

The flags "company-confidential" and "confidential" are equivalent and have the same effect. The "company-confidential" flag is in compliance with RFC 2156.

'!EM_SUBJECT! *subject*'
replaces the current subject of the message. The directive must be enclosed in quotation marks. If the subject contains special characters or more than one word (that is, it contains at least one blank space), you must enclose the text in quotation marks. Here are examples:

```
put '!em_subject! Sales';
put '!em_subject! "June Sales Report"';
```

'!EM_TO! *to-address*'
specifies the primary recipient or recipients of the email message. You must enclose the address in single or double quotation marks. To specify more than one address, enclose the group of addresses in parentheses, enclose each address in single or double quotation marks, and separate each address with either a comma or a space. To specify a real name as well as an address, enclose the address in angle brackets (< >). Here are examples:

```
put '!em_to! joe@site.com';
put '!em_to! ("joe@site.com" "jane@home.net")';
put '!em_to! Joe Smith <joe@site.com>';
```

Range 1–255 characters

Tip Specifying !EM_TO! overrides the '*address*' argument and the TO= email option.

Details

The Basics

You can send electronic mail programmatically from SAS using the EMAIL (SMTP) access method. To send email to an SMTP server, you first specify the SMTP email interface with the EMAILSYS system option, use the FILENAME statement to specify the EMAIL device type, and then submit SAS statements in a DATA step or in SCL code. The email access method has several advantages:

- You can use the logic of the DATA step or SCL to subset email distribution based on a large data set of email addresses.
- You can automatically send email upon completion of a SAS program that you submitted for batch processing.
- You can direct output through email based on the results of processing.

In general, DATA step or SCL code that sends email has these components:

- a FILENAME statement with the EMAIL device-type keyword
- email options specified in the FILENAME or FILE statement that indicate email recipients, subject, attached file or files, and so on
- PUT statements that define the body of the message
- PUT statements that specify email directives (of the form !EM_*directive*!) that override the email options (for example, TO=, CC=, SUBJECT=, ATTACH=) or perform actions such as send, end abnormally, or start a new message.

You can use encoded email passwords. When a password is encoded with PROC PWENCODE, the output string includes a tag that identifies the string as having been encoded. An example of a tag is {sas001}. The tag indicates the encoding method. Encoding a password enables you to avoid email access authentication with a password in plaintext. Passwords that start with "{sas" trigger an attempt to be decoded. If the decoding succeeds, then that decoded password is used. If the

decoding fails, then the password is used as is. For more information, see PROC PWENCODE in the *Base SAS Procedures Guide*.

For email messages that you send to another time zone, you can use the EMAILUTCOFFSET= system option to ensure that the email message has the UTC offset that represents your local time. You might use this option this if the time on your computer is not set to a time that uses a UTC offset or your computer does not account for Daylight Saving Time. The UTC offset specified in the EMAILUTCOFFSET= system option adds or replaces a UTC offset to the time in the email's Date: header field. For more information, see "EMAILUTCOFFSET= System Option" in *SAS System Options: Reference*.

The default amount of time that the EMAIL access method waits for the SMTP server to respond is 30 seconds. Some SMTP servers require more time before they send an acknowledgment to a command from the client. You can use the EMAILACKWAIT= system option to specify the wait time. For more information, see "EMAILACKWAIT= System Option" in *SAS System Options: Reference*.

You can use the EMAIL access method with secure SMTP servers by specifying the Transport Layer Security (TLS) protocol in the EMAILHOST= system option. TLS encrypts data between the client and the outgoing SMTP Server. This action does not guarantee an encrypted connection between the client (sender) and the recipient of the message. Message-level encryption and digital signing are currently not supported. For more information, see "EMAILHOST= System Option" in *SAS System Options: Reference*.

Note: All discussion of TLS is also applicable to the predecessor protocol, Secure Sockets Layer (SSL).

PUT Statement Syntax for EMAIL (SMTP) Access Method

In the DATA step, after using the FILE statement to define your email fileref as the output destination, use PUT statements to define the body of the message. Here is an example.

```
options emailsys=smtp;

filename mymail email 'martin@site.com' subject='Sending Email';

data _null_;
   file mymail;
   put 'Hi';
   put 'This message is sent from SAS...';
run;
```

You can also use PUT statements to specify email directives that override the attributes of your message. Examples of these attributes include TO=, CC=, SUBJECT=, CONTENT_TYPE=, and ATTACH=. Or, you can perform actions such as send, end abnormally, or start a new message. Specify only one directive in each PUT statement; each PUT statement can contain only the text that is associated with the directive that it specifies.

For a list of email directives, see "PUT Statement Email Directives" on page 48.

Examples

Example 1: Sending Email with an Attachment Using a DATA Step

In order to share a copy of your SAS configuration file with another user, you could send it by submitting this program. The email options are specified in the FILENAME statement.

```
options emailsys=smtp;

filename mymail email "JBrown@site.com"
   subject="My SAS Configuration File"
   attach="/u/sas/sasv8.cfg";
data _null_;
   file mymail;
   put 'Jim,';
   put 'This is my SAS configuration file.';
   put 'I think you might like the';
   put 'new options I added.';
run;
```

This program sends a message and two file attachments to multiple recipients. For this example, the email options are specified in the FILE statement instead of the FILENAME statement.

```
options emailsys=smtp;

filename outbox email "ron@acme.com";
data _null_;
   file outbox
      to=("ron@acme.com" "humberto@acme.com")
         /* Overrides value in */
         /* filename statement */
      cc=("miguel@acme.com" "loren@acme.com")
      subject="My SAS Output"
      attach=("C:\sas\results.out" "C:\sas\code.sas")
   ;
   put 'Folks,';
   put 'Attached is my output from the SAS';
   put 'program I ran last night.';
   put 'It worked great!';
run;
```

Example 2: Using Conditional Logic in a DATA Step

You can use conditional logic in a DATA step in order to send multiple messages and control which recipients get which message. For example, in order to send customized reports to members of two different departments, this program produces an email message and attachments that are dependent on the department to which the recipient belongs. In the program, these actions occur:

- In the first PUT statement, the !EM_TO! directive assigns the TO attribute.
- The second PUT statement assigns the SUBJECT attribute using the ! EM_SUBJECT! directive.

- The !EM_SEND! directive sends the message.
- The !EM_NEWMSG! directive clears the message attributes, which must be used to clear message attributes between recipients.
- The !EM_ABORT! directive abnormally ends the message before the RUN statement causes it to be sent again. The !EM_ABORT! directive prevents the message from being automatically sent at the end of the DATA step.

```
options emailsys=smtp;

filename reports email "Jim.Smith@work.com";
data _null_;
   file reports;
   length name dept $ 21;
   input name dept;
   put '!EM_TO! ' name;
   put '!EM_SUBJECT! Report for ' dept;
   put name ',';
   put 'Here is the latest report for ' dept '.' ;
   if dept='marketing' then
      put '!EM_ATTACH! c:\mktrept.txt';
   else  /* ATTACH the appropriate report */
      put '!EM_ATTACH! c:\devrept.txt';
      put '!EM_SEND!';
      put '!EM_NEWMSG!';
      put '!EM_ABORT!';
   datalines;
Susan          marketing
Peter          marketing
Alma           development
Andre          development
;
run;
```

Example 3: Sending Procedure Output in Email

You can use email to send procedure output. This example illustrates how to send ODS HTML in the body of an email message. The ODS HTML procedure output must be sent with the RECORD_SEPARATOR (RS) option set to NONE.

```
options emailsys=smtp;

filename outbox email
   to='susan@site.com'
   type='text/html'
   subject='Temperature Conversions';
data temperatures;
   do centigrade = -40 to 100 by 10;
      fahrenheit = centigrade*9/5+32;
      output;
   end;
run;
ods html
   body=outbox /* Mail it! */
   rs=none;
title 'Centigrade to Fahrenheit Conversion Table';
```

```
   proc print;
      id centigrade;
      var fahrenheit;
   run;
```

Example 4: Creating and Emailing an Image

This example illustrates how to create a GIF image and send it from SAS as an attachment to an email message.

```
options emailsys=smtp;

filename gsasfile email
   to='Jim@acme.com'
   type='image/gif'
   subject="SAS/GRAPH Output";
goptions dev=gif gsfname=gsasfile;
proc gtestit pic=1;
run;
```

Example 5: Using the MESSAGE/RFC822 Content Type

This example sends an email from an .mht file.

```
options emailsys=smtp;

filename myemail email
to="Jim@acme.com"
from="Wiley <wcoyote@acme.com>"
sender="Wiley <wcoyote@acme.com>"
subject="Message/RFC822 Example"
content_type="message/rfc822";
data _null_;
  file myemail;
  infile 'C:\temp\customer.mht';
  input @;
  put _infile_;
run;
```

Example 6: Creating an Email with an Embedded Image

This example creates an email that contains an embedded image.

```
options emailsys=smtp;

filename myemail email
to="Jim@acme.com"
from="Wiley <wcoyote@acme.com>"
sender="Wiley <wcoyote@sas.com>"
attach=('C:\Public\Pictures\Sample Pictures\sasLogo.gif' NAME="sasLogo" INLINED="logo"
   'C:\temp\reportToEmail.html' NAME='myreport')
subject="Embedded Image Example"
content_type="text/html";

data _null_;
file myemail;
   put 'Dear customer,<br><br>';
```

```
  put 'This is an example email with content type text/html, an attached report ';
  put 'and an embedded image.<br><br>';
  put 'Sincerely,<br><br>';
  put 'Wiley Coyote<br>';
  put 'Developer<br>';
  put 'SAS Research & Development<br>';
  put '1234 Any Street<br> ';
  put 'Sampletown';
  put 'NC 45678<br><br><br>';
  put '<IMG SRC="cid:logo" height="40" width="160">';
run;
```

See Also

- "How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?" in *SAS Language Reference: Concepts*
- "The SMTP E-Mail Interface" in *SAS Language Reference: Concepts*
- "Transport Layer Security (TLS) " in *Encryption in SAS*

Statements:

- "FILENAME Statement" on page 19
- "LOCKDOWN Statement" in SAS Intelligence Platform: Application Server Administration Guide

System Options:

- "EMAILACKWAIT= System Option" in *SAS System Options: Reference*
- "EMAILHOST= System Option" in *SAS System Options: Reference*
- "EMAILUTCOFFSET= System Option" in *SAS System Options: Reference*.

FILENAME Statement: FILESRVC Access Method

Enables you to store and retrieve user content using the SAS Viya Files service.

Valid in:	Anywhere
Category:	Data Access
Restriction:	You cannot access data in a SAS Content folder that includes an ampersand (&) in the folder name.
Requirement:	See "Requirements for the FILESRVC Access Method" on page 64.

Syntax

Form 1: Use this form to access SAS Viya files using a file Uniform Resource Identifier (URI).

FILENAME *fileref* FILESRVC '*file-uri*' <*filesrvc-options*>;

Form 2: Use this form to access a SAS Viya file using a collection option.

FILENAME *fileref* FILESRVC *collection-option* <*filesrvc-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 characters

FILESRVC
specifies the access method that enables you to create and access user content (such as reports) stored within the SAS Viya system.

file-uri
is a valid Uniform Resource Identifier (URI) of the content in the Files service. Here is an example. Note the file identifier contains a universally unique identifier (UUID) that is generated by the Files service when the content is created.

```
/files/files/<UUID>
```

Collection Options

The *collection-option* is one of these values:

FOLDERPATH='*path*'
specifies a path to a SAS Viya folder for directory access.

The fileref is used to access files that are members of the folder.

Here is an example:

```
filename jobout filesrvc folderpath='/output/user6/log';
```

Restrictions The maximum *path* length is 1024.

This option cannot be used with the FOLDERURI or PARENTURI options.

Interaction If the FILENAME= option is specified, the fileref accesses the named file that is a member of the provided folder. If the FILENAME= option is not specified, the fileref is a directory fileref that presents all file members of the folder as directory members.

FOLDERURI='*uri*'
specifies a path to a SAS Viya folder by URI.

The fileref is used to access files that are members of the folder. Here is an example:

```
filename jobout filesrvc
   folderuri='/folders/folders/5a308aa7-1c3a-4465-a14c-fd69a9091926';
```

Restrictions The maximum *uri* length is 2048 bytes.

This option cannot be used with the FOLDERPATH or PARENTURI options.

Interaction If the FILENAME= option is specified, the fileref accesses the named file that is a member of the provided folder. If FILENAME= is not specified, the fileref is a directory fileref that presents all file members of the folder as directory members.

PARENTURI='*uri*'
specifies a SAS Viya object by Uniform Resource Identifier (URI).

The fileref is used to access files that are associated with the parent object. Here is an example:

```
filename jobout filesrvc
   parenturi='/jobExecution/jobs/5a308aa7-1c3a-4465-a14c-fd69a9091926';
```

Restrictions The maximum *uri* length is 2048 bytes.

This option cannot be used with the FOLDERPATH or FOLDERURI options.

Interaction If the FILENAME= option is specified, the fileref accesses the named file that has the provided PARENTURI. If FILENAME= is not specified, the fileref is a directory fileref that presents all file associations of the object as directory members.

Note Files with the PARENTURI association are deleted when the parent object is deleted.

FILESRVC Options

The *filesrvc-options* can include these values:

CONTENTTYPE='*content-type*'
specifies a default HTTP **Content-Type** header to be returned with the file when the file's content is retrieved from the Files service.

Alias CT, CTYPE

Default The provided file extension is mapped to a default content type. If a mapping is not found for a provided file extension, the default content type is set to **application/octet-stream**.

Restriction The maximum *string* length is 64 bytes.

CONTENTDISP='*string*'
specifies how to deliver the file.

Here is an example that displays content as a file attachment:

```
CD='attachment; filename="name.ext"'
```

Alias CD, CDISP

Restriction The maximum *string* length is 64 bytes.

DEBUG=ERROR | HTTP
writes debugging information to the SAS log.

ERROR
indicates to write debugging information when HTTP calls fail.

HTTP
indicates to write debugging information for all HTTP calls.

DESCRIPTION='*string*'
specifies a description for the file.

Alias DESC

Restriction The maximum *string* length is 256 bytes.

DOCUMENTTYPE='*document-type*'
sets the document type attribute of the file.

Alias DT

Restriction The maximum *document-type* length is 64 bytes.

ENCODING='*encoding-value*'
specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

Default SAS assumes that an external file is in the same encoding as the session encoding.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

FILENAME='*name*'
specifies the name of a file.

Alias NAME

Restriction The maximum *name* length is 128 bytes.

Requirement When specifying a FILENAME, a file collection option must also be specified using the FOLDERPATH, FOLDERURI, or PARENTURI option.

Interaction Default content-type for a file is mapped using the file extension specified in the provided file name. If no mapping is found for a

provided file extension, the content-type is **application/octet-stream** unless specified differently with CONTENTTYPE.

LRECL=
specifies the logical record length. The LRECL value specifies the maximum number of bytes that can be accessed in one read or write call. Records that are longer than the specified LRECL might be truncated.

Beginning in SAS Viya 3.4, if files are created in the Files Service using the FILESRVC file name, LRECL and RECFM attributes are saved in the Files Service file. If you write a file using the Files Service with an LRECL= value and then read the file without specifying a value for LRECL=, the file is read using the value specified during the write.

Default 32,767 bytes

Range 1 to 1G

Interaction You can specify the record format using the RECFM= argument.

MOD
indicates that data written to the file should be appended to the file instead of overwriting the file.

RECFM=
specifies the record format. The RECFM= option is used for both input and output. The RECFM= option can be one of these values:

F
is a fixed record format. That is, each record has the same length.

N
is a binary format. The file consists of a stream of bytes with no record boundaries.

Alias S

V
is a variable record format. Each record ends with a newline character. You can specify the record delimiter using the TERMSTR= option.

Default V

Interaction You can specify the logical record length using the LRECL= argument.

TERMSTR=
specifies the record delimiter used for variable record format files (RECFM=V). Accepted values are CR, CRLF, LF, and NULL.

Defaults CRLF for Windows.

LF for UNIX.

Details

Requirements for the FILESRVC Access Method

The FILESRVC access method enables you to store and retrieve user content using the SAS Viya Files service. The FILESRVC access method stores files and associates files to folders, but it does not create or delete the folders. The FILESRVC access method can support all file encodings that are supported by the Files service. For more information, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

In order to use the FILESRVC access method, you must

- set the SERVICESBASEURL= system option before the invocation of your SAS session. This option specifies the host and port for Files service requests. The FILENAME statement fails at assignment time if the SERVICESBASEURL= option is not specified.
- define the SAS_VIYA_TOKEN environment variable. The SAS_VIYA_TOKEN environment variable contains a valid CAS OAuth access token and enables you to access SAS Viya services from SAS 9.4. For information about obtaining the access token, see "Obtain an Access Token Using Password Credentials" in *SAS Viya Administration*.

Interacting with the Files Service

The Files service enables you to store, retrieve, and delete user content that is maintained in the SAS Infrastructure Data Server database repository. The repository is not considered a complete 'file system.' Rather, the repository contains individual files that are directly accessible by their file identifier. This file identifier contains a universally unique identifier (UUID) that is generated by the Files service when a file is created.

The Files service also assigns a unique name to each file in the repository, but the name is not human-friendly. A user can change the name, but the name might not be unique within the repository.

You can access a file in the Files service by using the file identifier. The file identifier is contained in the URI that is stored in the file information. Use the system-generated name or user-assigned name to find the file URI and the file identifier. After you find the file identifier, you can use it to access the file directly in the Files service.

The Files service does not have a concept of 'folders' in its repository. However, you can associate files by using a PARENTURI. A PARENTURI is a relative URI for any object in SAS Viya. You can create a collection of files by specifying the same PARENTURI for each file.

For information about the SAS Infrastructure Data Server database, see "SAS Infrastructure Data Server" in *SAS Viya Administration*.

For more information about the Files service, see the SAS Viya Files API.

SAS Viya Folders Service

Unlike the Files service, the SAS Viya Folders service does provide support for folders. The Folders service also supports associating objects from other SAS Viya services (including the file objects in the Files service) with the Folders service folders as members.

The FILESRVC access method provides file operations for interacting with the Files service. Part of this interaction involves using the Folders service to provide grouping and directory operations and member operations. The access method can create files, associate files to a folder, and delete them from a folder. However, the access method does not support the creation or deletion of folders in the Folders service.

For more information about the Folders service, see the SAS Viya Folders API.

Generating a FILESRVC Macro Variable

When a single file or a folder is assigned to a FILESRVC fileref, a macro variable is generated. This macro variable allows for an easy way to access the file or folder. The macro variable name is _FILESRVC_*fileref*_URI. For a single file, this macro variable value is the Uniform Resource Identifier (URI) for the file (for example, /files/files/*UUID*). If the file does not exist, the value is blank. When you create a file using the FILESRVC access method, the macro variable is updated with the file's URI. For a folder, the macro variable value is the value of the folder URI.

Note: The FILESRVC access method does not create folders. The folder must already exist when the folder is specified using the FOLDERPATH= or FOLDERURI= collection options.

FILESRVC macro variables exist until the end of the SAS session. The value of the variable is changed only if the *fileref* is successfully re-assigned to a different Files service file. If the file is deleted using the FILESRVC access method, the macro variable remains, but the value of the variable is set to null.

Examples

Example 1: Accessing a File by Name and Folder

This example accesses the **sales.csv** file in the **/Shared Data/Sales** folder.

```
filename myfldr2 filesrvc folderpath='/Shared Data/Sales'
filename='sales.csv';
```

This example accesses the **sales.csv** file in the folder denoted by the provided folder URI.

```
filename myfldr filesrvc
   folderuri='/folders/folders/5a308aa7-1c3a-4465-a14c-fd69a9091926'
   filename='sales.csv';
```

Example 2: Listing Members of a Folder

This example shows how you can write out a list of members of a folder.

Note: You can use "MOPEN Function" in *SAS Functions and CALL Routines: Reference* to open a file. If an argument is invalid, then MOPEN returns 0. You can obtain the text of the corresponding error message from the SYSMSG function. Invalid arguments do not produce a message in the SAS log and do not set the _ERROR_ automatic variable.

If a folder contains a file that is not of type text, then MOPEN should only use binary-record format to read and download the member. If the internal format of the binary content is unknown, MOPEN should not be used to read and interpret the content.

```
filename myfldr filesrvc folderPath='/Users/Test/My Folder';

data _null_;
   did = dopen('myfldr');
   mcount = dnum(did);
   put 'MYFLDR contains ' mcount 'member(s)...';
   do i=1 to mcount;
      memname = dread(did, i);
      put i @5 memname;
   end;
   rc = dclose(did);
run;
```

Here is the partial log:

```
MYFLDR contains 35 member(s)...
1   member01.txt
2   member02.txt
...
33  member33.txt
34  mytest.dat
35  mytest.txt
```

Example 3: Creating a File and Associating It with a Parent URI

This example creates the file **class.csv** and associates the file with a job object referenced in the fileref **jobout**.

```
filename jobout filesrvc
   parenturi='/jobExecution/jobs/5a308aa7-1c3a-4465-a14c-fd69a9091926';
data _null_;
   set sashelp.class;
   file jobout('class.csv');
   put name "," sex "," age "," height "," weight;
run;
```

Example 4: Accessing an Associated File Using a Parent URI

This example accesses the **class.csv** file by using a parent object URI.

```
filename jobout filesrvc
   parenturi='/jobExecution/jobs/5a308aa7-1c3a-4465-a14c-fd69a9091926';
data _null_;
   length name $8 sex $1;
   infile jobout('class.csv') dlm=',';
   input name sex age height weight;
run;
```

See Also

Environment Variables:

- "SAS_VIYA_TOKEN Environment Variable" in *Encryption in SAS*

Statements:

- "FILENAME Statement" on page 19

System Options:

- "SERVICESBASEURL= System Option" in *SAS System Options: Reference*

FILENAME Statement: FTP Access Method

Enables you to access remote files by using the FTP protocol.

Valid in:	Anywhere
Category:	Data Access
Restrictions:	The FILENAME FTP access method does not support implicit FTPS When SAS is in a locked-down state, the FILENAME statement, FTP access method is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Language Reference: Concepts*.
Supports:	Explicit FTPES (FTP/TLS)

Syntax

FILENAME *fileref* FTP '*external-file*' <*ftp-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Note In SAS 9.4M3, the locale in which the SAS job is being run must support UTF-8 characters or your transcoded characters might not be what you expect. If the SAS program is running on z/OS, the OPTS UTF8 ON protocol command is not issued and no UTF-8 transcoding occurs.

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

FTP
specifies the access method that enables you to use File Transfer Protocol (FTP) to read from or write to a file from any host computer that you can connect to on a network with an FTP server running.

Tip Use FILENAME with FTP when you want to connect to the host computer, to log on to the FTP server, to make records in the specified file available for reading or writing, and to disconnect from the host computer.

'external-file'
specifies the physical name of an external file that you want to read from or write to. The physical name is the name that is recognized by the operating environment.

If the file has an IBM 370 format and a record format of FB or FBA, and if the ENCODING= option is specified, you must also specify the LRECL= option. If the length of a record is shorter than the value of LRECL, then SAS pads the record with blanks until the record length is equal to the value of LRECL.

Operating environment For details about specifying the physical names of external files, see the SAS documentation for your operating environment.

Tips If you are not transferring a file but performing a task such as retrieving a directory listing, then you do not need to specify a filename. Instead, put empty quotation marks in the statement. See "Example 1: Retrieving a Directory Listing" on page 81.

You can associate a fileref with a single file or with an aggregate file storage location.

If you use the DIR option, specify the directory in this argument.

ftp-options
specifies details that are specific to your operating environment such as file attributes and processing attributes.

Operating environment For more information about some of these FTP options, see the SAS documentation for your operating environment.

See "FTP Options" on page 69

FTP Options

AUTHDOMAIN="*auth-domain*"
specifies the name of an authentication domain metadata object in order to connect to the FTP server. The authentication domain references credentials (user ID and password) without your having to explicitly specify the credentials. The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

An 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 FTP server and is resolved by the BASE engine calling the SAS Metadata Server and returning the authentication credentials.

Requirement The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Interaction If you specify AUTHDOMAIN=, you do not need to specify USER= and PASS=.

See For more information about creating and using authentication domains, see the discussion on credential management in *SAS Intelligence Platform: Security Administration Guide*.

AUTHTLS
issues the FTP AUTH TLS command to the FTP server that requests TLS authentication. The Secure Command Channel Mode is entered by issuing the AUTH TLS command.

Requirement The AUTH TLS command must be issued to set up Control Channel protection before the Data Channel can be protected.

Interactions If you specify either the AUTHTLS, PROT=, or PBSZ= option, the AUTH TLS command is issued. An attempt to negotiate the TLS security with the FTP server occurs to protect the FTP Control Channel. If you specify the PROT= or PBSZ= option either independently or in conjunction with the AUTHTLS option, then an attempt to negotiate TLS security with the FTP server occurs to protect the FTP Control and Data Channels.

Instead of using the FILENAME FTP statement to turn TLS authentication on, you can define the SAS_FTP_AUTHTLS environment variable. For more information, see "SAS_FTP_AUTHTLS Environment Variable" on page 225.

See Problem Note 56154: File Transfer Protocol Secure (FTPS, FTPES, and FTP/TLS) support

Usage Note 61222: Differences between the FILENAME Access Methods: FTP, FTP/TLS, and SFTP

"PBSZ=*protection-buffer-size*" on page 75

"PROT=*protection-level*" on page 76

BINARY
is fixed-record format. Thus, all records are of size LRECL with no line delimiters. Data is transferred in image (binary) mode.

The BINARY option overrides the value of RECFM= in the FILENAME FTP statement, if specified, and forces a binary transfer.

Alias RECFM=F

Interaction If you specify the BINARY option and the S370V or S370VS option, then SAS ignores the BINARY option.

BLOCKSIZE=*blocksize*
where *blocksize* is the size of the data buffer in bytes.

Default 32768

CD='*directory*'
issues a command that changes the working directory for the file transfer to the *directory* that you specify.

Restriction The CD option cannot be used to specify a directory to delete files. Use the FDELETE= function:

```
filename delfile ftp '/mydir/myfile.txt' host='myhost.sas.com'
   user='user1' prompt;
data _null_;
   rc=fdelete('delfile');
   put rc=;
run;
```

Interaction The CD and DIR options are mutually exclusive. If both are specified, FTP ignores the CD option and SAS writes an informational note to the log.

CONNHOST='*url*'
specifies the Uniform Resource Locator (URL) for the web proxy server when you are accessing an FTP server through a proxy server. The CONNHOST= option creates a connection between the client and the proxy server and between the proxy server and the FTP server. The CONNHOST= option is equivalent to the PROXY= option in the FILENAME URL access method.

Requirement You must use the CONNHOST= option with the CONNPORT= option.

Example

```
filename myfile ftp "list.txt"
   connhost="proxsrv.abc.com"
   connport=3128
```

```
   host="ftp.myFTPdataSite.com"
   user="anonymous"
   pass=XXXXXXXXXXXXXXX;
```

CONNPASS='*password*'
specifies the password to use for proxy server authentication when you are accessing an FTP server through a proxy server. The CONNPASS= option contains the password that is paired with the user name that is specified in the CONNUSER= option.

Example

```
filename myfile ftp "list.txt"
   connhost="proxsrv.abc.com"
   connport=3128
   host="ftp.myFTPdataSite.com"
   user="anonymous"
   pass=XXXXXXXXXXXXXXX;
```

CONNPORT=
specifies the port number of the web proxy server that you are connecting to an FTP server through a proxy server.

Requirements You must use the CONNHOST= option with the CONNPORT= option.

You must use the CONNPORT= option with the CONNHOST= option.

Example

```
filename myfile ftp "list.txt"
   connhost="proxsrv.abc.com"
   connport=3128
   host="ftp.myFTPdataSite.com"
   user="anonymous"
   pass=XXXXXXXXXXXXXXX;
```

CONNUSER='*username*'
specifies the user name to use for proxy server authentication when you are accessing an FTP server through a proxy server. The CONNUSER= option contains the user name that is paired with the password that is specified in the CONNPASS= option.

See "PROXY=*url* " on page 111

DEBUG
writes to the SAS log informational messages that are sent to and received from the FTP server.

DIR
enables you to access directory files, PDS members, or PDSE members. Specify the directory name in the *external-file* argument. You must use valid directory syntax for the specified host.

Interaction The CD and DIR options are mutually exclusive. If both are specified, FTP ignores the CD option and SAS writes an informational note to the log.

Tips If you want FTP to append a file extension of DATA to the member name that is specified in the FILE or INFILE statement, then use the FILEEXT option in conjunction with the DIR option. The FILEEXT option is ignored if you specify a file extension in the FILE or INFILE statement.

If you want FTP to create the directory, then use the NEW option in conjunction with the DIR option. The NEW option is ignored if the directory exists.

If the NEW option is omitted and you specify an invalid directory, then a new directory is not created and you receive an error message.

The maximum number of directory or z/OS PDSE members that can be open simultaneously is limited by the number of sockets that can be open simultaneously on an FTP server. The number of sockets that can be open simultaneously is proportional to the number of connections that are set up during the installation of the FTP server. You might want to limit the number of sockets that are open simultaneously to avoid performance degradation.

Example "Example 10: Reading and Writing from Directories" on page 84

ENCODING=*encoding-value*
specifies the encoding to use when reading from or writing to the external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

Default SAS assumes that an external file is in the same encoding as the session encoding.

Tip The data is transferred in image or binary format and is in local data format. Thus, you must use appropriate SAS informats to read the data correctly.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

FILEEXT
specifies that the member type of DATA is automatically appended to the member name in the FILE or INFILE statement when you use the DIR option.

Tip The FILEEXT option is ignored if you specify a file extension in the FILE or INFILE statement.

See LOWCASE_MEMNAME option on page 73

Example "Example 10: Reading and Writing from Directories" on page 84

HOST='*host*'
where *host* is the network name of the remote host with the FTP server running.

You can specify either the name of the host (for example, `server.pc.mydomain.com`) or the IP address of the computer (for example, `2001:db8::`).

HOSTRESPONSELEN='*size*'
where *size* is the length of the FTP server response message.

Default 2048 bytes

Range 2048 to 16384 bytes

Restriction If you specify a *size* that is less than 2048 or is greater than 16384, the *size* is set to 2048.

LIST
issues the LIST command to the FTP server. LIST returns the contents of the working directory as records that contain all of the file attributes that are listed for each file.

Tip The file attributes that are returned vary, depending on the FTP server that is being accessed.

LOWCASE_MEMNAME
enables autocall macro retrieval of lowercase directory or member names from FTP servers.

Restriction SAS autocall macro retrieval always searches for uppercase directory member names. Mixed case directory or member names are not supported.

Interaction If you access files off FTP servers by using the %INCLUDE, FILE, INFILE, or other DATA step I/O statements, case sensitivity is preserved.

See FILEEXT option on page 72

LRECL=*lrecl*
where *lrecl* is the logical record length of the data.

Default 32767

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

LS
issues the LS command to the FTP server. LS returns the contents of the working directory as records with no file attributes.

Tips The file attributes that are returned vary, depending on the FTP server that is being accessed.

To return a listing of a subset of files, use the LSFILE= option in addition to LS.

LSFILE='*character-string*'
in combination with the LS option, specifies a character string that enables you to request a listing of a subset of files from the working directory. Enclose the character string in quotation marks.

Restriction LSFILE= can be used only if LS is specified.

Tips You can specify a wildcard as part of '*character-string* '.

The file attributes that are returned vary, depending on the FTP server that is being accessed.

Example This statement lists all of the files that start with *sales* and end with *sas*:

```
filename myfile ftp '' ls lsfile='sales*.sas'
   other-ftp-options;
```

MGET
transfers multiple files, similar to the FTP command MGET.

Tips The whole transfer is treated as one file. However, as the transfer of each new file is started, the EOV= variable is set to 1.

Specify MPROMPT to prompt the user before each file is sent.

MPROMPT
specifies whether to prompt for confirmation that a file is to be read, if necessary, when the user executes the MGET option.

Restriction The MPROMPT option is not available on z/OS for batch processing.

NEW
specifies that you want FTP to create the directory when you use the DIR option.

Restriction The NEW option is not available under z/OS.

Tip The NEW option is ignored if the directory exists.

PASS='*password*'
where *password* is the password to use with the user name specified in the USER= option.

Tips You can specify the PROMPT option instead of the PASS option, which tells the system to prompt you for the password.

If the user name is *anonymous*, then the remote host might require that you specify your email address as the password.

To use an encoded password, use the PWENCODE procedure in order to disguise the text string, and then enter the encoded password for the PASS= option. For more information, see "PWENCODE Procedure" in *Base SAS Procedures Guide*.

Example "Example 6: Using an Encoded Password" on page 83

PASSIVE
specifies that an attempt is made for passive mode FTP.

In passive mode FTP, the client initiates the control and data connections to the server. This action solves the problem of firewalls filtering the incoming data port connection to the client from the server.

Note Not all FTP servers support the passive mode. If an attempt is made by the FILENAME statement FTP access method to issue the PASV command and the command fails or the server does not accept the command, then active mode FTP is used for the connection.

PBSZ=*protection-buffer-size*
specifies the FTP data channel Protection Buffer Size.

Default 0

Range 0–2147483647 bytes

Interactions If you specify either the AUTHTLS, PROT=, or PBSZ= option, the AUTH TLS command is issued. An attempt to negotiate the TLS security with the FTP server occurs to protect the FTP Control Channel. If you specify the PROT= or PBSZ= option either independently or in conjunction with the AUTHTLS option, then an attempt to negotiate TLS security with the FTP server occurs to protect the FTP Control and Data Channels.

Instead of using the FILENAME FTP statement to turn TLS authentication on, you can define the SAS_FTP_AUTHTLS environment variable. For more information, see "SAS_FTP_AUTHTLS Environment Variable" on page 225.

Note IBM Mainframe FTP servers typically change whatever value you specify with the PBSZ= option to 0.

Tip If you specify a buffer size that is not within the range, the default value of 0 is used.

See "AUTHTLS" on page 69

"PROT=*protection-level*" on page 76

PORT=*portno*
where *portno* is the port that the FTP daemon monitors on the respective host.

The *portno* can be any number between 0 and 65535 that uniquely identifies a service.

Tip In the internet community, there is a list of predefined port numbers for specific services. For example, the default port for FTP is 21. A partial list of port numbers is usually available in the */etc/services* file on any UNIX computer.

PROMPT
specifies to prompt for the user login password, if necessary.

Restriction The PROMPT option is not available for batch processing under z/OS.

Interaction If PROMPT is specified without USER=, then the user is prompted for an ID, as well as a password.

Tip You can use the SAVEUSER option on page 78 to save the user ID and password after the user ID and password prompt is successfully executed.

PROT=*protection-level*
issues the FTP Data Channel protection level command. *protection-level* can be one of these values:

C
provides only FTP Control Channel security. The Data Channel is not protected.

E
provides Confidentiality Protection.

S
provides Integrity checking.

P
provides both Integrity checking and Confidentiality Protection.

Default P

Interactions If you specify either the AUTHTLS, PROT=, or PBSZ= option, the AUTH TLS command is issued. An attempt to negotiate the TLS security with the FTP server occurs to protect the FTP Control Channel. If you specify the PROT= or PBSZ= option either independently or in conjunction with the AUTHTLS option, then an attempt to negotiate TLS security with the FTP server occurs to protect the FTP Control and Data Channels.

Instead of using the FILENAME FTP statement to turn TLS authentication on, you can define the SAS_FTP_AUTHTLS environment variable. For more information, see "SAS_FTP_AUTHTLS Environment Variable" on page 225.

See "AUTHTLS" on page 69

"PBSZ=*protection-buffer-size*" on page 75

RCMD= *'command'*

where *command* is the FTP 'SITE' or 'service' command to send to the FTP server.

FTP servers use SITE commands to provide services that are specific to a system and are essential to file transfer but not common enough to be included in the protocol.

For example, **`rcmd='site rdw'`** preserves the record descriptor word (RDW) of a z/OS variable blocked data set as a part of the data. For more information, see "S370V" on page 78 and "S370VS" on page 79.

Interaction Some FTP service commands might not run at a particular client site depending on the security permissions and the availability of the commands.

Tips If you transfer a file with the FTP access method and then cannot read the file, you might need to change the FTP server's UMASK setting.

If the FTP server supports a SITE UMASK setting, you can change the permissions of the file as shown in this example:

```
filename in ftp '/mydir/accounting/file2.dat'
   host="xxx.fyi.xxx.com"
   user="john"
   rcmd='site umask 022'
   prompt;
data _null;
file in;
put a $80;
run;
```

You can specify multiple FTP service commands if you separate them by semicolons. Here are examples:

```
rcmd='ascii;site umask 002'
rcmd='stat;site chmod 0400 ~mydir/abc.txt'
```

RECFM=*recfm*

where *recfm* is one of three record formats:

F

is a fixed-record format. Thus, all records are of size LRECL with no line delimiters. Data is transferred in image (binary) mode.

Aliases BINARY

The BINARY option overrides the value of RECFM= in the FILENAME FTP statement, if specified, and forces a binary transfer.

Interaction The default value for the LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

S

is a stream-record format. Data is transferred in image (binary) mode.

Interaction The amount of data that is read is controlled by the current LRECL value or by the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL.

See The NBYTE= option in the INFILE statement.

V
is a variable-record format (the default). In this format, records have varying lengths, and the records are transferred in text mode.

Interaction Any record larger than LRECL is truncated.

Tip If you are using files with the IBM 370 Variable format or the IBM 370 Spanned Variable format, then you might want to use the S370V or S370VS options instead of the RECFM= option. For more information, see "S370V" on page 78 and "S370VS" on page 79.

Default V

Interaction If you specify the RECFM= option and the S370V or S370VS option, then SAS ignores the RECFM= option.

RHELP
issues the HELP command to the FTP server. The results of this command are returned as records.

RSTAT
issues the RSTAT command to the FTP server. The results of this command are returned as records.

SAVEUSER
saves the user ID and password after the user ID and password prompt are successfully executed.

Interaction The user ID and password are saved only for the duration of the SAS session or until you change the association between the fileref and the external file, or discontinue it with another FILENAME statement.

S370V
indicates that the file being read is in IBM 370 variable format.

Interaction If you specify this option and the RECFM= option, then SAS ignores the RECFM= option.

Tips The data is transferred in image or binary format and is in local data format. Thus, you must use appropriate SAS informats to read the data correctly on non-EBCDIC hosts.

Use the *rcmd='site rdw'* option when you transfer a z/OS data set with a variable-record format to another z/OS data set with a variable-record format to preserve the record descriptor word (rdw)

of each record. By default, most FTP servers remove the rdw that exists in each record before it is transferred.

Typically, the 'SITE RDW' command is not necessary when you transfer a data set with a z/OS variable-record format to ASCII, or when you transfer an ASCII file to a z/OS variable-record format.

S370VS
indicates that the file that is being read is in IBM 370 variable-spanned format.

Interaction If you specify this option and the RECFM= option, then SAS ignores the RECFM= option.

Tips The data is transferred in image or binary format and is in local data format. Thus, you must use appropriate SAS informats to read the data correctly on non-EBCDIC hosts.

Use the *rcmd='site rdw'* option when you transfer a z/OS data set with a variable-record format to another z/OS data set with a variable-record format to preserve the record descriptor word (rdw) of each record. By default, most FTP servers remove the rdw that exists in each record before it is transferred.

Typically, the 'SITE RDW' command is not necessary when you transfer a data set with a z/OS variable-record format to ASCII, or when you transfer an ASCII file to a z/OS variable-record format.

TERMSTR='*eol-char*'
where *eol-char* is the line delimiter to use when RECFM=V. There are three valid values:

CRLF carriage return (CR) followed by line feed (LF).

LF line feed only (the default).

NULL NULL character (0x00).

Default LF

Restriction Use this option only when RECFM=V.

USER='*username*'
where *username* is used to log on to the FTP server.

Restriction The FTP access method does not support FTP proxy servers that require user ID authentication.

Interaction If PROMPT is specified, but USER= is not, then the user is prompted for an ID.

Tip You can specify a proxy server and credentials for an FTP server when using the FTP access method. The user ID and password that you need to log on to the FTP server is sent via the proxy server by using the `user="userid@ftpservername" pass="password"`

host="*proxy.server.xxx.com*" syntax. Both anonymous and user ID validation are supported.

Example "Example 1: Retrieving a Directory Listing" on page 81

WAIT_MILLISECONDS=*milliseconds*
specifies the FTP response time in milliseconds.

Default 1,000 milliseconds

Tips If you receive a "connection closed; transfer aborted" or "network name is no longer available" message in the log, use the WAIT_MILLISECONDS option to increase the response time.

If you use the FEXIST function and the function reports a data set not found but the data set is cataloged, use the WAIT_MILLISECONDS option to increase the response time.

Details

The FTP access method lets you download and upload files. This method directly reads files into your SAS session without first storing them on your system.

In SAS 9.4M3, the FTP access method also supports explicit FTPES (FTP/TLS). Explicit FTPES includes full support for the Transport Layer Security (TLS) and Secure Socket Layer (SSL) cryptographic protocols, including the use of server-side public key authentication certificates and client-side authorization certificates.

The Transport Layer Security (TLS) protocol is used when the URL begins with "ftps" instead of "ftp". TLS and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that are designed to provide communication security over the internet. TLS and SSL are protocols that provide network data privacy, data integrity, and authentication. In addition to providing encryption services, TLS performs client and server authentication, and it uses message authentication codes to ensure data integrity. The TLS protocol allows client/server applications to communicate across a network in a way that is designed to prevent eavesdropping and tampering. TLS is supported by all major browser software.

The name of the FTP server being accessed must match the TLS or SSL certificate name created for that server. For UNIX and z/OS operating environments, the TLS or SSL certificate must be stored in an ASCII file and referred to by the SSLCALISTLOC= system option. The SSLCALISTLOC= system option specifies the location of a single file that contains the public certificate or certificates for all of the trusted certification authorities (CA) in the trust chain. On Windows operating environments, the TLS or SSL certificate needs to be imported to the certificate store for the computer.

Note: All discussion of TLS is also applicable to the predecessor protocol, SSL.

Note: If you do not specify the AUTHTLS, PROT=, or PBSZ= option in the FILENAME statement, FTP Access Method, TLS authentication is attempted. If the FTP server does not accept TLS authentication, then basic FTP authentication is used.

Comparisons

As with the FTP **get** and **put** commands, the FTP access method enables you to download and upload files. However, this method directly reads files into your SAS session without first storing them on your system.

Examples

Example 1: Retrieving a Directory Listing

This example retrieves a directory listing from a host named **mvshost1** for user **smythe**, and prompts **smythe** for a password:

```
filename dir ftp '' ls user='smythe'
   host='mvshost1.mvs.sas.com' prompt;
data _null_;
   infile dir;
   input;
   put _INFILE_;
run;
```

Note: The quotation marks are empty because no file is being transferred. However, because quotation marks are required by the syntax, you must include them.

Example 2: Reading a File from a Remote Host

This example reads a file called **sales** in the directory **/u/kudzu/mydata** from the remote UNIX host **hp720**.

```
filename myfile ftp 'sales' cd='/u/kudzu/mydata'
   user='guest' host='hp720.hp.sas.com'
   recfm=v prompt;
data mydata / view=mydata;   /* Create a view */
   infile myfile;
   input x $10. y 4.;
run;
proc print data=mydata;     /* Print the data */
run;
```

Example 3: Creating a File on a Remote Host

This example creates a file called **test.dat** in a directory called **c:\remote** for the user **bbailey** on the host **winnt.pc**.

```
filename create ftp 'c:\remote\test.dat'
   host='winnt.pc'
   user='bbailey' prompt recfm=v;
data _null_;
   file create;
   do i=1 to 10;
      put i=;
   end;
run;
```

Example 4: Reading an S370V-Format File on z/OS

This example reads an S370V-format file from a z/OS system. For more information about RCMD='**site rdw**', see RCMD= option on page 77 .

```
filename viewdata ftp 'sluggo.stat.data'
   user='sluggo' host='zoshost1'
   s370v prompt rcmd='site rdw';
data mydata / view=mydata;   /* Create a view */
   infile viewdata;
   input x $ebcdic8.;
run;
proc print data=mydata;     /* Print the data */
run;
```

Example 5: Anonymously Logging In to FTP

This example shows how to log on to FTP anonymously, if the host accepts anonymous logins.

Note: Some anonymous FTP servers require a password. If required, your email address is usually used. See PASS= option on page 74 under "FTP Options."

```
filename anon ftp '' ls host='130.96.6.1'
   user='anonymous';
data _null_;
   infile anon;
   input;
   list;
run;
```

Note: The quotation marks following the argument FTP are empty. A filename is needed only when transferring a file, not when routing a command. However, the quotation marks are required.

Example 6: Using an Encoded Password

This example shows you how to use an encoded password in the FILENAME statement.

In a separate SAS session, use the PWENCODE procedure to encode your password and make note of the output.

```
proc pwencode in= "MyPass1";
run;
```

This output appears in the SAS log:

```
(sas001)TX1QYXNzMQ==
```

You can now use the entire encoded password string in your batch program.

```
filename myfile ftp 'sales' cd='/u/kudzu/mydata'
   user='tjbarry' host='hp720.hp.mycompany.com'
   pass="(sas001)TX1QYXMZ==";
```

Example 7: Importing a Transport Data Set

This example uses the CIMPORT procedure to import a transport data set from a host named **myshost1** for user **calvin**. The new data set resides locally in the Sasuser library. Note that user and password can be SAS macro variables. If you specify a fully qualified data set name, then use double quotation marks and single quotation marks. Otherwise, the system appends the profile prefix to the name that you specify.

```
%let user=calvin;
%let pw=xxxxx;
filename inp ftp "'calvin.mat1.cpo'" user="&user"
   pass="&pw" rcmd='binary'
   host='mvshost1';
proc cimport library=sasuser infile=inp;
run;
```

Example 8: Transporting a SAS Library

This example uses the CPORT procedure to transport a SAS library to a host named **mvshost1** for user **calvin**. It creates a new sequential file on the host called **userid.mat64.cpo** with the record format of **fb**, lrecl of 80, and blocksize of 8000.

```
filename inp ftp 'mat64.cpo' user='calvin'
   pass="xxxx" host='mvshost1'
   lrecl=80 recfm=f blocksize=8000
   rcmd='site blocksize=800 recfm=fb lrecl=80';
proc cport library=mylib file=inp;
run;
```

Example 9: Creating a Transport Library with Transport Engine

This example creates a new SAS library on host **mvshost1**. The FILENAME statement assigns a fileref to the new data set. Note the use of the RCMD= option to specify important file attributes. The LIBNAME statement uses a libref that is the same as the fileref and assigns it to the XPORT engine. The PROC COPY step copies all data sets from the SAS library that are referenced by MYLIB to the

XPORT engine. Output from the PROC CONTENTS step confirms that the copy was successful:

```
filename inp ftp  'mat65.cpo' user='calvin'
   pass="xxxx"  host='mvshost1'
   lrecl=80 recfm=f blocksize=8000
   rcmd='site blocksize=8000 recfm=fb lrecl=80';
libname mylib 'SAS-library';
libname inp xport;
proc copy in=mylib out=inp mt=data;
run;
proc contents data=inp._all_;
run;
```

Note: For more information about the XPORT engine, see "Transport Engine" in *SAS Programmer's Guide: Essentials* and "XPORT Engine Limitations" in *Moving and Accessing SAS Files*.

Example 10: Reading and Writing from Directories

This example reads the file **ftpmem1** from a directory on a UNIX host, and writes the file **ftpout1** to a different directory on another UNIX host.

```
filename indir ftp '/usr/proj2/dir1' DIR
   host="host1.mycompany.com"
   user="xxxx" prompt;
filename outdir ftp '/usr/proj2/dir2' DIR FILEEXT
   host="host2.mycompany.com"
   user="xxxx" prompt;
data _null_;
   infile indir(ftpmem1) truncover;
   input;
   file outdir(ftpout1);
   put _infile_;
run;
```

The file **ftpout1** is written to **/usr/proj2/dir2/ftpout1.DATA**. Note that a member type of DATA is appended to the **ftpout1** file because the FILEEXT option was specified in the output file's FILENAME statement. For more information, see FILEEXT option on page 72 .

Note: The DIR option is not needed for some ODS destinations.

This example writes an output file and transfers it to an ODS-specified destination. The DIR option is not needed.

```
filename output ftp "~user/ftpdir/" host="host.fyi.company.com" user="userid"
pass="userpass" recfm=s debug;
ods html body='body.html' path=output;
proc print data=sashelp.class;run;
```

To export multiple graph files to a remote directory location, the DIR option must be specified in the FILENAME statement. Accordingly, when creating external graph files with the ODS HTML destination, two FILENAME statements are

needed: one for the HTML files, and one for the graph files. This example illustrates the need for two FILENAME statements.

```
filename output1 ftp "~user/dir" fileext host="host.unx.company.com"
   user="userid" pass="userpass" recfm=s debug;
filename output2 ftp "~user/dir" dir fileext host="host.unx.company.com"
   user="userid" pass="userpass" recfm=s debug;
ods html body='body.html' path=output1 gpath=output2
   frame='frames.html' contents='contents.html';
proc gtestit;
run;
quit;
;
```

Example 11: Using a Proxy Server

This example uses a proxy server with the FTP access method. The user ID and password are sent via the proxy server.

```
filename test ftp ' ' ls
   host='proxy.server.xxx.com'
   user='userid@ftpservername'
   pass='xxxxxx'
   cd='pubsdir/';
data _null_;
   infile test truncover;
   input a $256.;
   put a=;
run;
```

See Also

Environment Variables:

- “SAS_FTP_AUTHTLS Environment Variable” on page 225

Statements:

- “FILENAME Statement” on page 19
- “LIBNAME Statement” on page 139
- “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide

System Options:

- “SSLCALISTLOC= System Option” in *Encryption in SAS*

FILENAME Statement: Hadoop Access Method

Enables you to access files on a Hadoop Distributed File System (HDFS) whose location is specified in a configuration file.

Valid in: Anywhere

Category: Data Access

Restrictions: Access is restricted to Hadoop configurations on systems based on UNIX

When SAS is in a locked-down state, the FILENAME statement, Hadoop access method is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. If the FILENAME statement, Hadoop access method is re-enabled using the LOCKDOWN ENABLE_AMS= statement, the HADOOP procedure is automatically re-enabled. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Programmer's Guide: Essentials*.

Requirements: In SAS 9.4M3, to connect to the Hadoop cluster, the Hadoop configuration files must be copied from the specific Hadoop cluster to a physical location that is accessible to the SAS client machine. The SAS environment variable SAS_HADOOP_CONFIG_PATH must be set to the location of the Hadoop configuration files.

To use the FILENAME statement, Hadoop access method by using a Java native API, the Hadoop distribution JAR files must be copied to a physical location that is accessible to the SAS client machine. The SAS environment variable SAS_HADOOP_JAR_PATH must be defined and point to the location of the Hadoop JAR files.

To use the FILENAME statement, Hadoop access method through WebHDFS by using the REST API, the SAS environment variable SAS_HADOOP_RESTFUL 1 must be defined. In addition, the Hadoop configuration file hdfs-site.xml must include the properties for the WebHDFS location.

Syntax

FILENAME *fileref* HADOOP '*external-file*' <*hadoop-options*>;

Required Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement.

HADOOP
specifies the access method that enables you to use Hadoop to read from or write to a file from any host machine that you can connect to on a Hadoop configuration.

'external-file'
specifies the physical name of the file that you want to read from or write in an HDFS system. The physical name is the name that is recognized by the operating environment.

Operating environment: For more information about specifying the physical names of external files, see the SAS documentation for your operating environment.

Tip: Specify *external-file* when you assign a fileref to an external file. You can associate a fileref with a single file or with an aggregate file storage location.

Hadoop Options

hadoop-options can be any of these values:

BUFFERLEN=*bufferlen*
specifies the maximum buffer length of the data that is passed to Hadoop for its I/O operations.

Default: 503808

Restriction: The maximum buffer length is 1000000.

Tip: Specifying a buffer length that is larger than the default could result in performance improvements.

CFG="*physical-pathname-of-hadoop-configuration-file*" | *fileref-that-references-a-hadoop-configuration-file*
specifies the configuration file that contains the connections settings for a specific Hadoop cluster.

Note: If a file is specified, it is the only file that is used to obtain configuration information.

Note: If a directory is specified, the directory is used to obtain the required configuration files.

Note: The CFG= option is required for a configuration file that is specific to Apache Oozie. You must also set the SAS environment variable SAS_HADOOP_CONFIG_PATH. For other uses, specify the location of configuration files by setting the SAS_HADOOP_CONFIG_PATH environment variable only. The environment variable is used by several SAS components.

See HADOOP Configuration Guide

CONCAT
specifies that the HDFS directory name that is specified on the FILENAME HADOOP statement is considered a wildcard specification. The concatenation of all the files in the directory is treated as a single logical file and read as one file.

Restriction This works for input only.

Interaction The CONCAT and DIR options are mutually exclusive. If both options are specified, Hadoop ignores the DIR option and SAS writes an informational note to the log.

Tip For best results, do not concatenate text and binary files.

DEBUG
enables additional messages that are displayed on the SAS log.

DIR
enables you to access files in an HDFS directory.

Requirement You must use valid directory syntax for the specified host.

Interactions The CONCAT and DIR options are mutually exclusive. If both options are specified, Hadoop ignores the DIR option and SAS writes an informational note to the log.

Specify the HDFS directory name in the *external-file* argument.

If you want to create the directory, use the NEW option in conjunction with the DIR option. The NEW option is ignored if the directory exists. If the NEW option is omitted and you specify an invalid directory, then a new directory is not created and you receive an error message.

See "FILEEXT" on page 89

"NEW" on page 89

ENCODING='*encoding-value*'
specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

Default SAS assumes that an external file is in the same encoding as the session encoding.

Note When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

FILEEXT
specifies that a file extension is automatically appended to the filename when you use the DIR option

Interaction The autocall macro facility always passes the extension .SAS to the file access method as the extension to use when opening files in the autocall library. The DATA step always passes the extension .DATA. If you define a fileref for an autocall macro library and the files in that library have a file extension of .SAS, use the FILEEXT option. If the files in that library do not have an extension, do not use the FILEEXT option. For example, if you define a fileref for an input file in the DATA step and the file X has an extension of .DATA, you would use the FILEEXT option to read the file X.DATA. If you use the INFILE or FILE statement, enclose the member name and extension in quotation marks to preserve case.

Tip The FILEEXT option is ignored if you specify a file extension on the FILE or INFILE statement.

See "LOWCASE_MEMNAME" on page 89

LOWCASE_MEMNAME
enables autocall macro retrieval of lowercase directory or member names from HDFS systems.

Restriction SAS autocall macro retrieval always searches for uppercase directory member names. Mixed-case directory or member names are not supported.

See "FILEEXT" on page 89

LRECL=*logical-record-length*
specifies the logical record length of the data.

Default 65536

MOD
places the file in Update mode and appends updates to the bottom of the file.

MAXWAIT=*wait-interval*
specifies the HTTP status response time when using WebHDFS.

Default 40000 milliseconds

Requirement The environment variable SAS_HADOOP_RESTFUL 1 must be set.

Tip If you receive a time-out message in the log, use the MAXWAIT to increase the wait period.

NEW
specifies that you want to create the directory when you use the DIR option.

Interaction If you want to create the directory, use the NEW option in conjunction with the DIR option. The NEW option is ignored if the directory exists. If the NEW option is omitted and you specify an

invalid directory, then a new directory is not created and you receive an error message.

See "DIR" on page 88

PASS='*password*'
specifies the password to use with the user name that is specified in the USER option.

Requirement The password is case sensitive and it must be enclosed in single or double quotation marks.

Tip To use an encoded password, use the PWENCODE procedure in order to disguise the text string, and then enter the encoded password for the PASS= option. For more information, see "PWENCODE Procedure" in *Base SAS Procedures Guide*.

PROMPT
specifies to prompt for the user login, the password, or both, if necessary.

Interaction The USER= and PASS= options override the PROMPT option if all three options are specified. If you specify the PROMPT option and do not specify the USER= or PASS= option, you are prompted for a user ID and password.

RECFM=*record-format*
where *record-format* is one of three record formats:

F
is a fixed-record format. In this format, records have fixed lengths, and they are read in binary mode.

S
is a binary-record format. The file consists of a series of bytes with no record boundaries.

Tip The amount of data that is read is controlled by the current LRECL value or the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL. To avoid problems when you read large binary files like PDF or GIF, set NBYTE=1 to read one byte at a time.

See "NBYTE=variable" in *SAS DATA Step Statements: Reference*

V
is a variable-record format (the default). In this format, records in the file have varying lengths.

Tip Any record larger than LRECL is truncated.

Default V

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

USER='*username*'
where *username* is used to log on to the Hadoop system.

Requirements If you connect to Hadoop without specifying the USER= option, you must start SAS with this option:

```
-jreoptions "(-Djavax.security.auth.useSubjectCredsOnly=false)"
```

The user name is case sensitive, and it must be enclosed in single or double quotation marks.

Details

An HDFS system has levels of permissions at both the directory and file level. The Hadoop access method honors those permissions. For example, if a file is available as read-only, you cannot modify it.

Operating Environment Information: Using the FILENAME statement requires information that is specific to your operating environment. The Hadoop access method is fully documented here. For more information about how to specify filenames, see the SAS documentation for your operating environment.

Examples

Example 1: Writing to a New Member of a Directory

This example writes the file **shoes** to the directory **testing**.

```
set=SAS_HADOOP_CONFIG_PATH="/u/hadoopcfg/cdh52p1";

filename out hadoop '/user/testing/' user='xxxx'
  pass='xxxx' recfm=v  lrecl=32167 dir ;

data _null_;
   file out(shoes) ;
   put 'write data to shoes file';
run;
```

Example 2: Buffering 1MB of Data during a File Read

This example uses the BUFFERLEN option to buffer 1MB of data at a time during the file read. The records of length 1024 are read from this buffer.

```
set=SAS_HADOOP_CONFIG_PATH="/u/hadoopcfg/cdh52p1";

filename foo hadoop 'file1.dat'
   user='user' pass='apass' recfm=s
```

```
   lrecl=1024 bufferlen=1000000;

data _null_;
   infile foo truncover;
input a $1024.;
put a;
run;
```

Example 3: Using the CONCAT Option

This example uses the CONCAT option to read all members of DIRECTORY1 as if they are one file.

```
set=SAS_HADOOP_CONFIG_PATH="/u/hadoopcfg/cdh52p1";

filename foo hadoop '/directory1/' user='user' pass='apass'
   recfm=s lrecl=1024 concat;

data _null_;
   infile foo truncover;
input a $1024.;
put a;
run;
```

See Also

Statements:

- “FILENAME Statement” on page 19
- “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide

FILENAME Statement: S3 Access Method

Enables you to access Amazon S3 files.

Valid in:	Anywhere
Category:	Data Access
Restrictions:	Support for the S3 access method in SAS 9 begins in SAS 9.4M8. The S3 access method is not supported on z/OS platforms.
Supports:	PROC S3 statement options
See:	“S3 Procedure” in *Base SAS Procedures Guide*

Syntax

FILENAME *fileref* S3 *"object path"*<*s3-options*>;

Required Arguments

fileref
is a valid fileref.

S3
specifies the S3 access method.

object-path
specifies the S3 object that you want to access.

Details

Using the S3 Access Method

The S3 access method enables you to access objects in the Simple Storage Service (S3) of Amazon Web Services (AWS).

Before you can use the S3 access method, you need an AWS access key ID and a secret access key. When using temporary credentials, you also need a security token. For more information, see the Amazon S3 documentation.

Support for the ENCKEY Statement

The S3 access method supports the ENCKEY statement of PROC S3. The ENCKEY statement supports server-side encryption in an Amazon Web Services (AWS) S3 environment. For an example that shows how to encrypt data in an AWS S3 environment, see "Example 4: Using the S3 Access Method with the ENCKEY Statement" on page 94. For information about the ENCKEY statement, see "ENCKEY Statement" in *Base SAS Procedures Guide*.

Examples

Example 1: Reading a File by Using the S3 Access Method

This example uses the S3 access method to read a file in an AWS S3 environment. The FILENAME statement creates an alias (fileref) called `myfile`, which points to the location of the file, `filename.txt`, on the AWS S3 server. The DATA step reads the file and writes its contents to the SAS log.

```
filename myfile s3 '/directory1/filename.txt';

data _null_;
  infile myfile;
  input;
```

```
  put _infile_;
run;
```

Example 2: Sending a Log Output to a New Destination by Using the S3 Access Method

This example uses the S3 access method with PROC PRINTTO to route a log output to a file, `log.txt`, on the AWS S3 server. The NEW option removes any information that is in the file and prepares the file to receive the output. For more information, see "PROC PRINTTO Statement" in *Base SAS Procedures Guide*.

The S3 access method in the FILENAME statement provides an alias and pointer to the location of the `log.txt` file on AWS. The LOG= option in PROC PRINTTO specifies that all log output is sent to the external file on AWS. You can also specify the PRINT= option in the PRINTTO statement so that all procedure output is sent to an external file.

```
filename mylog s3 "/directory1/userid/log.txt" ;

proc printto log=mylog new;
run;

proc print data=sashelp.cars;
run;

proc printto;
run;
```

Example 3: Importing a File by Using the S3 Access Method

This example uses PROC IMPORT with the S3 access method to import a file to a SAS data set from an AWS S3 environment.

```
filename i s3 '/directory1/i.cvs' ;

proc import datafile=i out=work.i replace dbms=csv debug;
  getnames=yes;
  datarow=2;
  guessingrows=all;
run;
```

Example 4: Using the S3 Access Method with the ENCKEY Statement

This example uses the SAS DATA step with PROC S3 and the ENCKEY statement to encrypt a server-side file in an AWS S3 environment.

```
proc s3;
  enckey add name="mykey"                                                    /* 1 */
  hexkey="7468697349734d6f726546726565666f726d49735550506f7365736f79656168";
run;

filename myfile s3 "/directory1/filename.txt" enckey=mykey;              /* 2 */

data _null_;
```

```
  file myfile;                                                    /* 3 */
  put 'Write with enckey';
run;
```

1 The ENCKEY statement in PROC S3 creates the encryption key and stores it in the name `mykey`.

2 The FILENAME statement creates an alias named myfile and uses the S3 access method to associate the alias with the S3 server file, filename.txt. The S3 access method uses the ENCKEY= option to send the encryption information to the S3 server where the data in filename.txt is encrypted.

3 The FILE statement in the DATA step uses the alias for the S3 server file to specify that the PUT statement output is written to the server file.

FILENAME Statement: SFTP Access Method

Enables you to access remote files by using the SFTP protocol.

Valid in: Anywhere

Category: Data Access

Syntax

FILENAME *fileref* SFTP '*external-file*' <*sftp-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

SFTP
specifies the access method that enables you to use Secure File Transfer Protocol (SFTP) to read from or write to a file from any host computer that you can connect to on a network with an OpenSSH SSHD server running.

'external-file'
specifies the physical name of an external file that you want to read from or write to. The physical name is the name that is recognized by the operating environment.

Operating environment: For more information about specifying the physical names of external files, see the SAS documentation for your operating environment.

Tips: If you are not transferring a file but performing a task such as retrieving a directory listing, then you do not need to specify an external filename. Instead, put empty quotation marks in the statement.

You can associate a fileref with a single file or with an aggregate file storage location.

sftp-options
specifies details that are specific to your operating environment such as file attributes and processing attributes.

Operating environment: For more information about some of these SFTP options, see the SAS documentation for your operating environment.

See: "SFTP Options" on page 96

SFTP Options

sftp-options can be any of these values:

BATCHFILE='*path*'
specifies the fully qualified pathname and the filename of the batch file that contains the SFTP commands. These commands are submitted when the SFTP access method is executed. After the batch file processing ends, the SFTP connection is closed.

Requirement: The path must be enclosed in quotation marks.

Tip: After the batch file processing ends, the SFTP connection is closed and the filename assignment is no longer available. If subsequent DATA step processing requires the FILENAME SFTP statement, then another FILENAME SFTP statement is required.

Example: "Example 5: Using a Batch File" on page 102

CD='*directory*'
issues a command that changes the working directory for the file transfer to the *directory* that you specify.

DEBUG
writes informational messages to the SAS log.

DIR
enables you to access directory files. Specify the directory name in the external-file argument. You must use valid directory syntax for the specified host.

Interaction: The CD and DIR options are mutually exclusive. If both are specified, SFTP ignores the CD option and SAS writes an informational note to the log.

Tips If you want SFTP to create the directory, then use the NEW option in conjunction with the DIR option. The NEW option is ignored if the directory exists.

If the NEW option is omitted and you specify an invalid directory, then a new directory is not created and you receive an error message.

HOST='*host*'
where *host* is the network name of the remote host with the OpenSSH SSHD server running.

You can specify either the name of the host (for example, `server.pc.mydomain.com`) or the IP address of the computer (for example, `2001:db8::`).

LRECL=*lrecl*
where *lrecl* is the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

LS
issues the LS command to the SFTP server. LS returns the contents of the working directory as records with no file attributes.

Restriction The LS option does display files with leading periods. An example is *.xAuthority*.

Interaction The LS and LSA options are mutually exclusive. If you specify both options, the LSA option takes precedence.

Tip To return a listing of a subset of files, use the LSFILE= option in addition to LS.

LSA
issues the LS command to the SFTP server. LSA returns all the contents of the working directory as records with no file attributes.

Interactions The LS and LSA options are mutually exclusive. If you specify both options, the LSA option takes precedence.

To display files without leading periods. For example, use the LS= option with *.xAuthority*.

Tip To return a listing of a subset of files, use the LSFILE= option in addition to LSA.

LSFILE='*character-string*'
in combination with the LS option, specifies a character string that enables you to request a listing of a subset of files from the working directory. Enclose the character string in quotation marks.

Restriction LSFILE= can be used only if LS or LSA is specified.

Tip You can specify a wildcard as part of '*character-string* '.

Example This statement lists all of the files that start with *sales* and end with *sas*:

```
filename myfile sftp '' ls lsfile='sales*.sas'
   other-sftp-options;
```

MGET
transfers multiple files, similar to the SFTP command MGET.

Tip The whole transfer is treated as one file. However, as the transfer of each new file is started, the EOV= variable is set to 1.

NEW
specifies that you want SFTP to create the directory when you use the DIR option.

Restriction The NEW option is not available under z/OS.

Tip The NEW option is ignored if the directory exists.

OPTIONS='*option-string*'
specifies SFTP configuration options such as port numbers and verbose.

Requirement When you submit code that contains a FILENAME SFTP statement from SAS Enterprise Guide that runs on a Windows workspace server, you must specify authentication by using the OPTIONS or OPTIONSX option.

Note If you need to blot any information in the OPTIONS string, use the OPTIONSX option.

See "Example 6: Connecting a Windows PUTTY Client to an SSHD Server By Using Authentication Specified on the OPTIONSX Parameter in SAS Enterprise Guide" on page 102

OPTIONSX='*option-string*'
specifies SFTP configuration options such as private keys and passphrases. All information in the *option-string* is blotted when written to the SAS log.

Requirements When you submit code that contains a FILENAME SFTP statement from SAS Enterprise Guide that runs on a Windows workspace server, you must specify authentication by using the OPTIONS or OPTIONSX option.

If the passphrase in the OPTIONSX string contains one or more spaces, then the passphrase must be enclosed in double

quotation marks and the OPTIONSX string must be enclosed in single quotation marks.

Tip The passphrase is passed using the **-pw** parameter.

See "Example 6: Connecting a Windows PUTTY Client to an SSHD Server By Using Authentication Specified on the OPTIONSX Parameter in SAS Enterprise Guide" on page 102

PATH
specifies the location of the SFTP executable if it is not installed in the PATH or $PATH search path.

Tip It is recommended that the OpenSSH "SFTP" executable or PUTTY "PSFTP" executable be installed in a directory that is accessible via the PATH or $PATH search path.

RECFM=*recfm*
where *recfm* is one of three record formats:

F
is a fixed-record format. Thus, all records are of size LRECL with no line delimiters.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256.

S
is a stream-record format. Data is transferred in image (binary) mode.

Interaction The amount of data that is read is controlled by the current LRECL value or by the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL.

See The NBYTE= option in the INFILE statement.

V
is variable-record format (the default). In this format, records have varying lengths, and they are separated by newlines.

Default V

USER='*username*'
specifies the user name.

Requirement The *username* is required by the PUTTY client on the Windows host.

Tips The *username* is not typically required on LINUX or UNIX hosts when using public key authentication.

Public key authentication using an SSH agent is the recommended way to connect to a remote SSHD server.

WAIT_MILLISECONDS=*milliseconds*
specifies the SFTP response time in milliseconds.

Default 1,500 milliseconds

Tip If you receive a time-out message in the log, use the WAIT_MILLISECONDS option to increase the response time.

Details

The Basics

The Secure File Transfer Protocol (SFTP) provides a secure connection and file transfers between two hosts (client and server) over a network. Both commands and data are encrypted. The client machine initiates a connection with the remote host (OpenSSH SSHD server).

With the SFTP access method, you can read from or write to any host computer that you can connect to on a network with an OpenSSH SSHD server running. The client and server applications can reside on the same computer or on different computers that are connected by a network.

Specific implementation details are dependent on the OpenSSH SSHD server version and how that site is configured.

The SFTP access method relies on default send and reply messages to OpenSSH commands. Custom installs of OpenSSH that modify these messages disable the SFTP access method.

To use the SFTP access method, the applicable client software must be installed. The SFTP access method supports only these SSH clients.

- OpenSSH – UNIX
- PUTTY – Windows

Note: Password validation is not supported for the SFTP access method.

Note: Public key authentication using an SSH agent is the recommended way to connect to a remote SSHD server.

Note: If you have trouble running the SFTP access method, try to manually validate SFTP client access to an OpenSSH SSHD server without involving the SAS system. Manually validating SFTP client access without involving the SAS system ensures that your SSH or SSHD configuration and key authentication is setup correctly.

SFTP Access Methods and SFTP Prompts

The SFTP access method supports only these prompts. Changing the prompt disables the SFTP access method.

- For OpenSSH:
  - *sftp>*
- For PUTTY:
  - *psftp>*

Comparisons

As with the SFTP **get** and **put** commands, the SFTP access method lets you download and upload files. However, this method directly reads files into your SAS session without first storing them on your system.

Examples

Example 1: Connecting to an SSHD Server at a Standard Port

This example reads a file called **test.dat** using the SFTP access method after connecting to the SSHD server a standard port:

```
filename myfile sftp '/users/xxxx/test.dat' host="unixhost1";
data _null_;
   infile myfile truncover;
   input a $25.;
run;
```

Example 2: Connecting to an SSHD Server at a Nonstandard Port

This example reads a file called **test.dat** using the SFTP access method after connecting to the SSHD server at port 4117:

```
filename myfile sftp '/users/xxxx/test.dat' host="unixhost1" options="-
oPort=4117";
data _null_;
   infile myfile truncover;
   input a $25.;;
run;
```

Example 3: Connecting a Windows PUTTY Client to an SSHD Server

This example writes a file called **test.dat** using the SFTP access method after connecting a Windows PUTTY client to the SSHD server with a user ID of **userid**:

```
filename outfile sftp '/users/xxxx/test.dat' host="unixhost1"
user="userid";
data _null_;
   file outfile;
   do i=1 to 10;
      put i=;
   end;
run;
```

Example 4: Reading Files from a Directory on the Remote Host

This example reads the files **test.dat** and **test2.dat** from a directory on the remote host.

```
filename infile sftp '/users/xxxx/' host="unixhost1" dir;
data _null_;
   infile infile(test.dat) truncover;
   input a $25.;
   infile infile(test2.dat) truncover;
   input b $25.;
run;
```

Example 5: Using a Batch File

In this example, when the INFILE statement is processed, the batch file associated with the FILENAME SFTP statement, **sftpcmds**, is executed.

```
filename process sftp ' ' host="unixhost1" user="userid"
   batchfile="c:/stfpdir/sftpcmds.bat";
data _null_;
   infile process;
run;
```

Example 6: Connecting a Windows PUTTY Client to an SSHD Server By Using Authentication Specified on the OPTIONSX Parameter in SAS Enterprise Guide

This example writes a file, **test.dat**, by using the SFTP access method after connecting a Windows PUTTY client to the SSHD server. Public key authentication occurs with the private key and passphrase specified on the OPTIONSX parameter. The OPTIONSX string values are blotted with X characters in the SAS log. The passphrase is passed using the **-pw** parameter. If the passphrase contains spaces, then the passphrase must be enclosed in double quotation marks and the OPTIONSX string must be enclosed in single quotation marks.

```
filename outfile sftp '/users/xxxx/test.dat' host="unixhost1"
  optionsx='-i C:\privatekey.ppk -pw "pass phrase"'   user="userid" ;
data _null_;
   file outfile;
   do i=1 to 10;
      put i=;
   end;
run;
```

See Also

- Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. 2005. "SSH, The Secure Shell: The Definitive Guide." Sebastopol, CA: O'Reilly

Statements:

- "FILENAME Statement" on page 19
- "LIBNAME Statement" on page 139

FILENAME Statement: SOCKET Access Method

Enables you to read from or write to a TCP/IP socket.

Valid in: Anywhere

Category: Data Access

Restriction: When SAS is in a locked-down state, the FILENAME statement, SOCKET access method is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Language Reference: Concepts*.

Syntax

Form 1: **FILENAME** *fileref* SOCKET '*hostname*:*portno*' <*tcpip-options*>;

Form 2: **FILENAME** *fileref* SOCKET ':*portno*' SERVER <*tcpip-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

SOCKET
specifies the access method that enables you to read from or write to a Transmission Control Protocol/Internet Protocol (TCP/IP) socket.

'hostname:portno'
is the name or IP address of the host and the TCP/IP port number to connect to.

Tip Use this specification for client access to the socket.

':portno'
is the port number to create for listening.

Tips Use this specification for server mode.

If you specify *:0*, the system chooses a number.

SERVER
sets the TCP/IP socket to be a listening socket, thereby enabling the system to act as a server that is waiting for a connection.

Tip The system accepts all connections serially; only one connection is active at any one time.

See The RECONN= option on page 105 under *TCPIP Options*.

tcpip-options
specifies details that are specific to your operating system such as the number of connections that the server accepts.

Operating Environment Information: For more information about some of these TCP/IP options, see the SAS documentation for your operating environment

See "TCP/IP Options" on page 104

TCP/IP Options

tcpip-options can be any of these values:

BLOCKSIZE=*blocksize*
where *blocksize* is the size of the socket data buffer in bytes.

Default 8192

ENCODING=*encoding-value*
specifies the encoding to use when reading from or writing to the socket. The value for ENCODING= indicates that the socket has a different encoding from the current session encoding.

When you read data from a socket, SAS transcodes the data from the specified encoding to the session encoding. When you write data to a socket, SAS transcodes the data from the session encoding to the specified encoding.

For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

LRECL=*lrecl*
where *lrecl* is the logical record length.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

RECFM=*recfm*

where *recfm* is one of three record formats:

F

is a fixed-record format. Thus, all records are of size LRECL with no line delimiters. Data is transferred in image (binary) mode.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

S

is a stream-record format.

Interaction The amount of data that is read is controlled by the current LRECL value or the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable equal to the amount of data to be read. This amount must be less than or equal to LRECL.

Tip Data is transferred in image (binary) mode.

See The NBYTE= option in the INFILE statement.

V

is a variable-record format (the default).

Tips In this format, records have varying lengths, and they are transferred in text mode.

Any record larger than LRECL is truncated.

Default V

RECONN=*conn-limit*

where *conn-limit* is the maximum number of connections that the server accepts.

Note Because only one connection can be active at a time, a connection must be disconnected before the server can accept another connection. When a new connection is accepted, the EOV= variable is set to 1. The server continues to accept connections, one at a time, until *conn-limit* has been reached.

TERMSTR='*eol-char*'

where *eol-char* is the line delimiter to use when RECFM=V. There are three valid values:

CRLF
carriage return (CR) followed by line feed (LF).

LF
line feed only (the default).

NULL
NULL character (0x00).

Default LF

Restriction Use this option only when RECFM=V.

Details

The Basics

A TCP/IP socket is a communication link between two applications. The *server* application creates the socket and waits for a connection. The *client* application connects to the socket. With the SOCKET access method, you can use SAS to communicate with another application over a socket in either client or server mode. The client and server applications can reside on the same computer or on different computers that are connected by a network.

For example, you can develop an application using Microsoft Visual Basic that communicates with a SAS session that uses the TCP/IP sockets. Note that Visual Basic does not provide inherent TCP/IP support. You can obtain a custom control (VBX) from SAS Technical Support (free of charge) that allows a Visual Basic application to communicate through the sockets.

Using the SOCKET Access Method in Client Mode (*Form 1*)

In client mode, a local SAS application can use the SOCKET access method to communicate with a remote application that acts as a server (and waits for a connection). Before you can connect to a server, you must know this information:

- the network name or IP address of the host computer running the server
- the port number that the remote application is listening to for new connections

The remote application can be another SAS application, but it does not need to be. When the local SAS application connects to the remote application through the TCP/IP socket, the two applications can communicate by reading from and writing to the socket as if it were an external file. If at any time the remote side of the socket is disconnected, the local side also automatically terminates.

Using the SOCKET Access Method in Server Mode (*Form 2*)

When the local SAS application is in server mode, it remains in a wait state until a remote application connects to it. To use the SOCKET access method in server mode, you need to know only the port number that you want the server to listen to for a connection. Typically, servers use *well-known ports* to listen for connections. These port numbers are reserved by the system for specific server applications. For

more information about how well-known ports are defined on your system, see the documentation for your TCP/IP software or ask your system administrator.

If the server application does not use a well-known port, then the system assigns a port number when it establishes the socket from the local application. However, because any client application that waits to connect to the server must know the port number, you should try to use a well-known port.

While a local SAS server application is waiting for a connection, SAS is in a wait state. Each time a new connection is established, the EOV= variable in the DATA step is set to 1. Because the server accepts only one connection at a time, no new connections can be established until the current connection is closed. The connection closes automatically when the remote client application disconnects. The SOCKET access method continues to accept new connections until it reaches the limit set in the RECONN option.

Example: Communicating between Two SAS Applications over a TCP/IP Socket

This example shows how two SAS applications can talk over a TCP/IP socket. The local application is in server mode; the remote application is the client that connects to the server. This example assumes that the server host name is **hp720.unx.sas.com**, that the well-known port number is 5000, and that the server allows a maximum of three connections before closing the socket.

Here is the program for the server application:

```
filename local socket ':5000' server reconn=3;
   /*The server is using a reserved */
   /*port number of 5000.           */
data tcpip;
   infile local eov=v;
   input x $10;
   if v=1 then
      do;              /* new connection when v=1 */
         put 'new connection received';
      end;
   output;
run;
```

Here is the program for the remote client application:

```
filename remote socket 'hp720.unx.sas.com:5000';
data _null_;
   file remote;
   do i=1 to 10;
      put i;
   end;
run;
```

See Also

Statements:

- “FILENAME Statement” on page 19
- “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide

FILENAME Statement: URL Access Method

Enables you to access remote files by using the URL access method.

Valid in: Anywhere

Category: Data Access

Restriction: When SAS is in a locked-down state, the FILENAME statement, URL access method is not available. Your server administrator can re-enable this access method so that it is accessible in the locked-down state. If the FILENAME statement, URL access method is re-enabled, the SOAP procedure is automatically re-enabled. For more information, see “SAS Processing Restrictions for Servers in a Locked-Down State” in *SAS Language Reference: Concepts*.

Syntax

FILENAME *fileref* URL '*external-file*' <*url-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

URL
specifies the access method that enables you to read a file from any host computer that you can connect to on a network with a URL server running.

Alias HTTP

'*external-file*'
specifies the name of the file that you want to read from on a URL server. The Transport Layer Security (TLS) protocol, https, can also be used to access the files. The file must be specified in one of these formats:

- http://*hostname/file*
- https://*hostname/file*
- http://*hostname:portno/file*
- https://*hostname:portno/file*

Operating environment: For more information about specifying the physical names of external files, see the SAS documentation for your operating environment.

URL Options

url-options can be any of these values:

ACCEPT='*header-type*'
specifies the Accept: header. The Accept: header can be used to specify certain media types, which are acceptable for the response.

Default: */*

Requirement: *header-type* must be enclosed in either single or double quotation marks.

AUTHDOMAIN="*auth-domain*"
specifies the name of an authentication domain in order to connect to the proxy or web server. The authentication domain references credentials (user ID and password) without your having to explicitly specify the credentials. The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

An 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 proxy or web server and is resolved by the BASE engine calling the SAS Metadata Server and returning the authentication credentials.

Requirement: The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Interaction: If you specify AUTHDOMAIN=, you do not need to specify USER= and PASS=.

See: For more information about creating and using authentication domains, see the discussion on credential management in *SAS Intelligence Platform: Security Administration Guide*.

BLOCKSIZE=*blocksize*
specifies the size of the URL data buffer in bytes.

Default 8K

CONNECT
creates a connection between the client and the proxy and between the proxy and the server when accessing a URL through a proxy.

Requirement You must use the PROXY= option with the CONNECT option. No connection is made if the CONNECT option is used without the PROXY= option.

Interaction If you use "http" in the *external-file* argument, a connection is made, but the TLS protocol is not used.

See "PROXY=*url* " on page 111

DEBUG
writes debugging information to the SAS log.

Tip The result of the HELP command is returned as records.

HEADERS=*fileref*
specifies the fileref to which the header information is written when a file is opened by using the URL access method. The header information is the same information that is written to the SAS log.

Requirement The fileref must be defined in a previous FILENAME statement.

Interactions If you specify the HEADERS= option without specifying the DEBUG option, the DEBUG option is automatically turned on.

By default, log information is overwritten. To append the log information, you must specify the MOD option in the FILENAME statement that creates the fileref.

LRECL=*lrecl*
specifies the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value is 256.

PASS='*password*'
where *password* is the password to use with the user name that is specified in the USER option.

Tips You can specify the PROMPT option instead of the PASS option, which tells the system to prompt you for the password.

To use an encoded password, use the PWENCODE procedure in order to disguise the text string, and then enter the encoded password for the PASS= option. For more information, see "PWENCODE Procedure" in *Base SAS Procedures Guide*.

PPASS='*password*'
where *password* is the password to use with the user name that is specified in the PUSER option. The PPASS option is used to access the proxy server.

Tips You can specify the PROMPT option instead of the PPASS option, which tells the system to prompt you for the password.

To use an encoded password, use the PWENCODE procedure to disguise the text string, and then enter the encoded password for the PASS= option. For more information, see "PWENCODE Procedure" in *Base SAS Procedures Guide*.

PROMPT
specifies to prompt for the user login password if necessary.

Tip If you specify PROMPT, you do not need to specify PASS= or PPASS=.

PROXY=*url*
specifies the Uniform Resource Locator (URL) for the proxy server in one of these forms:

http://*hostname*/

http://*hostname:portno*/

See "CONNECT" on page 110

PUSER='*username*'
where *username* is used to log on to the URL proxy server.

Interactions If you specify the PUSER option, the USER option goes to the web server regardless of whether you specify a proxy server.

If PROMPT is specified, but PUSER is not, the user is prompted for an ID as well as a password.

Tip If you specify puser='*', then the user is prompted for an ID.

RECFM=*recfm*
specifies one of three record formats:

F
is a fixed-record format. Thus, all records are of size LRECL with no line delimiters. Data is transferred in image (binary) mode.

Interaction In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

S
is a stream-record format. Data is transferred in image (binary) mode.

Alias N

Tip The amount of data that is read is controlled by the current LRECL value or the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL.

See The NBYTE= option in the INFILE statement.

V
is a variable-record format (the default). In this format, records have varying lengths, and the records are transferred in text mode.

Tip Any record larger than LRECL is truncated.

Default V

TERMSTR='*eol-char*'
specifies the line delimiter to use when RECFM=V. There are four valid values:

CR carriage return (CR).

CRLF carriage return (CR) followed by line feed (LF).

LF line feed only (the default).

NULL NULL character (0x00).

Default LF

Restriction Use this option only when RECFM=V.

USER='*username*'
specifies the *username* that is used to log on to the URL server.

Interactions If you specify the USER option but do not specify the PUSER option, where the USER option goes depends on whether you specify a proxy server. If you do not specify a proxy server, USER goes to the web server. If you do specify a proxy server, USER goes to the proxy server.

If you specify the PUSER option, the USER option goes to the web server regardless of whether you specify a proxy server.

If PROMPT is specified, but USER or PUSER is not, the user is prompted for an ID as well as a password.

Tip If you specify user='*', then the user is prompted for an ID.

Details

The Transport Layer Security (TLS) protocol is used when the URL begins with "https" instead of "http". TLS and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that are designed to provide communication security over the internet. TLS and SSL are protocols that provide network data privacy, data integrity, and authentication. In addition to providing encryption services, TLS performs client and server authentication, and it uses message authentication codes to ensure data integrity. The TLS protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. TLS is supported by all major browser software.

Note: All discussion of TLS is also applicable to the predecessor protocol, SSL.

Operating Environment Information: Using the FILENAME statement requires information that is specific to your operating environment. The URL access method is fully documented here, but for more information about how to specify filenames, see the SAS documentation for your operating environment.

Examples

Example 1: Accessing a File at a Website

This example accesses document **test.dat** at site **www.a.com**:

```
filename foo url 'http://www.a.com/test.dat'
   proxy='http://www.gt.sas.com';
```

Example 2: Specifying a User ID and a Password

This example accesses document **file1.html** at site **www.b.com** using the TLS protocol and requires a user ID and password:

```
filename foo url 'https://www.b.com/file1.html'
    user='jones' prompt;
```

Example 3: Reading Records from a URL File

This example reads records from lines 228 through 248 from a URL file and writes the records to the SAS log with a PUT statement:

```
filename foo url "http://support.sas.com/publishing/cert/sampdata.txt";

data _null_;
   infile foo length=len;
   input record $varying200. len;
   if _n_>=228 and _n_<=248 then do;
      put record $varying200. len;
   end;
run;
```

See Also

- "Transport Layer Security (TLS) " in *Encryption in SAS*

Statements:

- "FILENAME Statement" on page 19
- "LOCKDOWN Statement" in SAS Intelligence Platform: Application Server Administration Guide

FILENAME Statement: WebDAV Access Method

Enables you to access remote files by using the WebDAV protocol.

Valid in: Anywhere

Category: Data Access

Syntax

FILENAME *filref* WEBDAV '*external-file*' <*webdav-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

WEBDAV
specifies the access method that enables you to use WebDAV (Web Distributed Authoring and Versioning) to read from or write to a file from any host machine that you can connect to on a network with a WebDAV server running.

'external-file'
specifies the name of the file that you want to read from or write to a WebDAV server. The external file must be in one of these forms:

http://*hostname/path-to-the-file*

https://*hostname/path-to-the-file*

http://*hostname*:*port/path-to-the-file*

https://*hostname*:*port/path-to-the-file*

Requirement When using the HTTPS communication protocol, you must use the TLS or SSL protocol that provides secure network communications. For more information, see *Encryption in SAS*.

Operating environment For more information about specifying the physical names of external files, see the SAS documentation for your operating environment.

WebDAV Options

webdav-options can be any of these values:

AUTHDOMAIN="*auth-domain*"
specifies the name of an authentication domain metadata object in order to connect to the WebDAV server. The authentication domain references credentials (user ID and password) without your explicitly specifying the credentials.

An 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 proxy or web server. The authentication domain is resolved by the BASE engine calling the SAS Metadata Server and returning the authentication credentials.

Requirements The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

Interaction If you specify AUTHDOMAIN=, you do not need to specify USER= and PASS=.

See For more information about creating and using authentication domains, see the section on credential management in *SAS Intelligence Platform: Security Administration Guide*.

DEBUG
writes debugging information to the SAS log.

DEL_ALL
enables you to delete a directory and all its members.

Requirement The DIR option is required when you use the DEL_ALL option.

Note The default behavior of the WebDAV access method is that only empty directories can be deleted. Use the DEL_ALL option to delete directories that are not empty.

See "DIR" on page 116

DIR
enables you to access directory files. Specify the directory name in the external-file argument. You must use valid directory syntax for the specified host.

Tip See FILEEXT option on page 116 for information about specifying file extensions.

ENCODING='*encoding-value*'
specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

Default SAS assumes that an external file is in the same encoding as the session encoding.

See "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

FILEEXT
specifies that a file extension is automatically appended to the filename when you use the DIR option.

Interaction The autocall macro facility always passes the extension .SAS to the file access method as the extension to use when opening files in the autocall library. The DATA step always passes the extension .DATA. If you define a fileref for an autocall macro library and the files in that library have a file extension of .SAS, use the FILEEXT option. If the files in that library do not have an extension, do not use the FILEEXT option. For example, if you define a fileref for an input file in the DATA step and the file *X* has an extension of .DATA, you would use the FILEEXT option to read the file X.DATA. If you use the INFILE or FILE statement, enclose the member name and extension in quotation marks to preserve case.

Tip The FILEEXT option is ignored if you specify a file extension in the FILE or INFILE statement.

See LOWCASE_MEMNAME option on page 117

LOCALCACHE="*directory name*"
specifies a directory where a temporary subdirectory is created to hold local copies of the server files. Each fileref has its own unique subdirectory. If a directory is not specified, then the subdirectories are created in the SAS Work directory. SAS deletes the temporary files when the SAS program completes.

Default SAS Work directory

LOCKDURATION=*n*
specifies the number of minutes that the files that are written through the WebDAV fileref are locked. SAS unlocks the files when the SAS program successfully finishes executing. If the SAS program fails, then the locks expire after the time allotted.

Default 30 minutes

LOWCASE_MEMNAME
enables autocall macro retrieval of lowercase directory or member names from WebDAV servers.

Restriction SAS autocall macro retrieval always searches for uppercase directory member names. Mixed-case directory or member names are not supported.

See FILEEXT option on page 116

LRECL=*lrecl*
specifies the logical record length of the data.

Default 256

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767. If you are using fixed-length records (RECFM=F), the default value for LRECL is 256.

MKDIR="*new-directory-name*"
specifies a new directory that is created from the parent directory that was specified in the *external-file* option.

Requirements You must use valid directory syntax for the specified host.

You must use the DIR option with the MKDIR option.

Example
```
filename bankname webdav "http://webserver.com/parentdir/"
   dir mkdir="testdir1" user="myid" pass="xxxx";
```

MOD
Places the file in Update mode and appends updates to the bottom of the file.

PASS='*password*'
where *password* is the password to use with the user name that is specified in the USER option. The password is case sensitive and it must be enclosed in single or double quotation marks.

Alias PASSWORD=, PW=, PWD=

Tip To use an encoded password, use the PWENCODE procedure in order to disguise the text string, and then enter the encoded password for the PASS= option. For more information, see "PWENCODE Procedure" in *Base SAS Procedures Guide* .

PROMPT
specifies to prompt for the user logon password, if necessary.

Interaction The USER= and PASS= options override the PROMPT option if all three options are specified. If you specify the PROMPT option and do not specify the USER= or PASS= option, you are prompted for a user ID and password.

PROXY=*url*
specifies the Uniform Resource Locator (URL) for the proxy server in one of these forms:

http://*hostname*/

http://*hostname:port*/

RECFM=*recfm*
where *recfm* is one of two record formats:

S
is a stream-record format. Data is transferred in image (binary) mode.

Note It is recommended that you specify RECFM=S for PDF or any other binary files, especially if ENCODING=UTF8 or any other Unicode encoding. Otherwise, a byte-order mark (BOM) is written at the beginning of the file and an incorrect content type is generated.

Tip The amount of data that is read is controlled by the current LRECL value or the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to the LRECL. To avoid problems when you transfer large binary files such as PDF or GIF, set NBYTE=1 to transfer one byte at a time.

See The NBYTE= option in the INFILE statement.

V
is a variable-record format (the default). In this format, records have varying lengths, and they are transferred in text mode.

Tip Any record larger than LRECL is truncated.

Default V

USER='*username*'
where *username* is used to log on to the URL server. The user ID is case sensitive and it must be enclosed in single or double quotation marks.

Alias UID=

Details

The Basics

Web Distributed Authoring and Versioning (WebDAV) is an extension of the HTTP protocol that enables users to manage files on a remote server, access documents over the web, and collaboratively work on them.

When you access a WebDAV server to update a file, the file is pulled from the WebDAV server to your local disk storage for processing. When this processing is complete, the file is pushed back to the WebDAV server for storage. The file is removed from the local disk storage when it is pushed back.

The Transport Layer Security (TLS) protocol is used when the URL begins with "https" instead of "http". TLS and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that are designed to provide communication security over the internet. TLS and SSL are protocols that provide network data privacy, data integrity, and authentication. In addition to providing encryption services, TLS performs client and server authentication, and it uses message authentication codes to ensure data integrity. The TLS protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. TLS is supported by all major browser software.

The name of the WebDAV server being accessed must match the TLS or SSL certificate name created for that server. For UNIX and z/OS operating environments, the TLS or SSL certificate must be stored in an ASCII file and referred to by the SSLCALISTLOC= system option. The SSLCALISTLOC= system option specifies the location of a single file that contains the public certificate or certificates for all of the trusted certification authorities (CA) in the trust chain. On Windows operating environments, the TLS or SSL certificate needs to be imported to the certificate store for the computer.

Note: All discussion of TLS is also applicable to the predecessor protocol, SSL.

Note: WebDAV servers have levels of permissions at both the directory and file level. The WebDAV access method honors those permissions. For example, if a file is available as read-only, the user cannot modify it.

Operating Environment Information: Using the FILENAME statement requires information that is specific to your operating environment. The WebDAV access method is fully documented here, but for more information about how to specify file names, see the SAS documentation for your operating environment.

Examples

Example 1: Accessing a File at a Website

This example accesses the file **`rawFile.txt`** at site **`www.mycompany.com`**.

```
filename foo webdav 'https://www.mycompany.com/production/files/
rawFile.txt'
   user='wong' pass='jd75ld';
data _null_;
   infile foo;
   input a $80.;
run;
```

Alternatively, you can use the FILENAME function in a SAS DATA step to perform the same task shown in the preceding example. In this example, the FILENAME function uses the WebDAV protocol to access the file `rawFile.txt`.

```
data _null_;
rc = filename(
   'foo',                                                    /*1*/
   'https://www.mycompany.com/production/files/rawFile.txt', /*2*/
   'webdav',                                                 /*3*/
   'user="wong" pass="jd75ld"');                             /*4*/
run;

data _null_;
   infile foo;
   input a $80.;
run;
```

1 In the FILENAME function, specify a fileref (`foo`).

2 Specify the name of the external file that you want to read (`rawFile.txt`). Use the fully qualified domain name.

3 Specify the device-type (`webdav`).

4 Specify the username and password. These are host-options in the FILENAME function.

Example 2: Using a Proxy Server

This example accesses the file **`acctgfile.dat`** by using the proxy server **`otherwebsvr:80`**.

```
filename foo webdav 'https://webserver.com/webdav/acctgfile.dat'
   user='sanchez' pass='239sk349exz'
   proxy='http://otherwebsvr.com:80';
data _null_;
   infile foo;
   input a $80.;
run;
```

Example 3: Writing to a New Member of a Directory

This example writes the file **`SHOES`** to the directory **`TESTING`**.

```
filename writeit webdav
   "https://webserver.com:8443/webdav/testing/"
    dir user="webuser" pass=XXXXXXXXX;
    data _null_;
       file writeit(shoes);
```

```
      set sashelp.shoes;
      put region $25. product $14.;
   run;
```

Example 4: Reading from a Member of a Directory

This example reads the file **SHOES** from the directory **TESTING1**.

```
filename readit webdav
   "https://webserver.com:8443/webdav/testing1/"
   dir user="webuser" pass=XXXXXXXXX;
   data shoes;
      length region $25  product $14;
      infile readit(shoes);
      input region $25. product $14.;
   run;
```

Example 5: Using a WebDAV Location as an Autocall Macro Library

By default, the autocall macro facility expects uppercase file names. This example accesses the file **MYTEST** in the autocall macro library **WRITEIT**.

```
filename writeit webdav
   "https://webserver.com/webdav/macrolib"
      dir fileext user="webuser" pass=XXXXXXXXX;
options SASAUTOS=(writeit);
/* expects a file called MYTEST.SAS */
%MYTEST;
```

Example 6: Accessing a Lowercased Autocall Macro Member

This example accesses the file **testmem.sas** in the autocall macro library **LIST**. The LOWCASE_MEMNAME option is used to access the file, which is in lowercase.

```
filename list webdav "https://webserver.com:8443/accounting/"
   dir fileext user="xxxxx" pass="xxxxx" LOWCASE_MEMNAME;
options sasautos=(list);
%testmem;
```

Example 7: Using a %INCLUDE Statement and Macro Invocation to Access a Lowercased Autocall Macro Member

This example accesses the file **testmem.sas** in the autocall macro library **MYTEST**. Because the file is accessed by using the %INCLUDE statement, case sensitivity is preserved.

```
filename mytest webdav "https://webserver.com:8443/payroll/"
   dir user="xxxxxx" pass="xxxxx";
%include mytest(testmem.sas) /source2;
%testmem;
```

If the file name was in uppercase, the reference to the file name in the %INCLUDE statement and macro call needs to be uppercase.

```
%include mytest(TESTMEM.SAS) /source2;
```

```
%TESTMEM;
```

Example 8: Accessing a File with a Mixed-Case Name

This example accesses the file **fileNOext** from the **production** directory. Because the file is quoted in the INFILE statement, case sensitivity is preserved and the file extension is ignored.

```
filename test webdav "https://webserver.com:8443/production"
     dir user="xxxxxx" pass="xxxxx";
data _null_;
   infile test('fileNOext');
   input;
   list;
run;
```

Example 9: Using the FILEEXT Option to Automatically Attach a File Extension

This example accesses the file **testmem.sas** from the **sales** directory. The FILEEXT option automatically adds **.DATA** as the file extension. The member name that is read is **testmem.DATA**.

```
filename listing webdav "https://webserver.com:8443/sales"
     dir fileext user="xxxxxx" pass="xxxxx";
data _null_;
   infile listing(testmem);
   input;
   list;
run;
```

Example 10: Deleting a Directory That Contains Members

In this example, the **newusers** directory contains members, and the deletion of the directory fails.

```
filename newusers webdav "https://webserver.com:8443/production"
     dir user="xxxxxx" pass="xxxxx";

/**** cannot delete newusers because it has members ****/
data _null_;
   rc=fdelete("newusers");
   put rc=;
run;

/ **** can delete newusers because del_all in filename statement ****/
```

In this example, the FILENAME statement contains the DEL_ALL option, which enables the **newusers** directory to be deleted.

```
filename newusers webdav "https://webserver.com:8443/production"
     dir user="xxxxxx" pass="xxxxx" del_all;

/ **** can delete newusers because del_all option ****/
data _null_;
   rc=fdelete("newusers");
   put rc=;
```

```
run;
```

See Also

- "Transport Layer Security (TLS) " in *Encryption in SAS*

Statements:

- "FILENAME Statement" on page 19
- "LIBNAME Statement: WebDAV Server Access" on page 184

System Options:

- "SSLCALISTLOC= System Option" in *Encryption in SAS*

FILENAME Statement: ZIP Access Method

Enables you to access ZIP files.

Valid in: Anywhere

Category: Data Access

Note: The ZIP access method reads and writes only files created with the WinZip or GZIP file compression.

Syntax

FILENAME *fileref* ZIP '*external-file*' <*zip-options*>;

Arguments

fileref
is a valid fileref.

Range 1 to 8 bytes

Tip The association between a fileref and an external file lasts only for the duration of the SAS session or until you change the fileref or discontinue it with another FILENAME statement. You can change the fileref for a file as often as you want.

ZIP
specifies the access method that enables you to use ZIP files.

'external-file'
specifies the name of the ZIP file that you want to read from or write to.

Operating environment: For information about specifying the physical names of external files, see the SAS documentation for your operating environment.

Notes: If you write multiple entries of the same ZIP file in a DATA step, an error occurs. Multiple entries overlay each other with unpredictable results.

All ZIP filenames and ZIP file entries are case sensitive.

Tip: Specify *external-file* when you assign a fileref to an external file. You can associate a fileref with a single file by using the MEMBER= syntax or with an aggregate file storage location that uses the *fileref*(*member*) syntax.

ZIP Options

zip-options can be any of these values:

COMMENT="*comment-string*"
writes an informative comment in a ZIP file.

COMPRESSION='*compression-level*'
specifies the compression level that is used to write to the ZIP file member. Valid values for the compression level are 0 through 9. A value of 0 stores the file with no compression. A value of 9 indicates maximum compression.

Default: 6

Restriction: COMPRESSION= is used only when opening a file for writing.

DEBUG
writes debugging information to the SAS log.

ENCODING=*encoding-value*
specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

Default: SAS assumes that an external file is in the same encoding as the session encoding.

See: "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*

FILEEXT
specifies that a file extension is automatically appended to the filename member if the extension does not exist.

Interaction The autocall macro facility always passes the extension .SAS to the file access method as the extension to use when opening files in the autocall library. The DATA step always passes the extension .DATA. If you define a fileref for an autocall macro library and the files in that library have a file extension of .SAS, use the FILEEXT option. If the files in that library do not have an extension, do not use the FILEEXT option. For example, if you define a fileref for an input file in the DATA step and the file X has an extension of .DATA, you would use the FILEEXT option to read the file X.DATA. If you use the INFILE or FILE statement, enclose the member name and extension in quotation marks to preserve the casing.

Tip The FILEEXT option is ignored if you specify a file extension on the FILE or INFILE statement.

See "LOWCASE_MEMNAME" on page 125

GZIP
specifies that the external file is a GZIP file.

Interaction The GZIP and MEMBER= options are mutually exclusive. If both options appear, GZIP takes precedence and the MEMBER= option is ignored.

Note Support for this option was added in SAS 9.4M5.

LOWCASE_MEMNAME
enables autocall macro retrieval of lowercase directory or member names from ZIP files.

Restriction SAS autocall macro retrieval always searches for uppercase directory member names. Mixed-case directory or member names are not supported.

See "FILEEXT" on page 124

LRECL=*lrecl*
specifies the logical record length of the data.

Default 32767

Interaction Alternatively, you can specify a global logical record length by using the "LRECL= System Option" in *SAS System Options: Reference*. In SAS 9.4, the default value for the global LRECL system option is 32767.

MEMBER="*member-file*"
associates the fileref with a single file found inside the ZIP file.

Interaction The MEMBER= and GZIP options are mutually exclusive. If both options appear, GZIP takes precedence and the MEMBER= option is ignored.

Tip You can use a wildcard in the MEMBER= syntax. An asterisk (*) matches zero or more characters. A question mark (?) matches one character. Wildcards are supported when reading entries and for exist actions. Wildcards are not supported for write or delete actions. For example, writing entry “A*” creates an entry “A*”. Deleting an entry named “A*” deletes “A*” but not any entry that starts with “A”. Calling an exist function with “A*” returns True as long as one or more entries that start with “A” exist.

NAMEENCODING=*encoding-value*
specifies the encoding to use for ZIP file entry names and comments. The value for NAMEENCODING= indicates that the entry name and comment have a different encoding from the current session encoding.

Default Code Page 437

Example
```
filename zs zip "yxz.zip" nameencoding=sjis member="s" termstr=lf;
```

RECFM=*recfm*
where *recfm* is one of four record formats:

F
is a fixed-record format. Each record has the same length.

N
is a binary format. The file consists of stream bytes with no record boundaries.

S
is a stream-record format.

Interaction The amount of data that is read is controlled by the current LRECL value or by the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL.

See The NBYTE= option in the INFILE statement.

V
is a variable-record format (the default). In this format, records have varying lengths, the records are transferred in text mode.

Interaction Any record larger than LRECL is truncated.

Default V

TERMSTR='*eol-termination-character*'
specifies the terminating character, which is the line character for Read operations and the terminating character for Write operations. There are four valid values:

CR carriage return (CR).

CRLF carriage return (CR) followed by line feed (LF).

LF line feed only (the default).

NULL NULL character (0x00).

Default CRLF for Windows. LF for all other operating environments.

Operating environment Using the FILENAME statement requires information that is specific to your operating environment. For more information about how to specify filenames, see the SAS documentation for your operating environment.

Examples

Example 1: Reading a ZIP File Member from a Directory

This example reads the *test1.txt* ZIP file member from the **testzip** ZIP file.

```
filename foo ZIP 'U:\directory1\testzip.zip' member="test1.txt" ;

data _null_;
infile foo;
input a $80.;
run;
```

Example 2: Writing a ZIP File to a New Member of a Directory

This example writes the *shoes* file to the **testzip** ZIP file.

```
filename foo ZIP 'U:\directory1\testzip.zip';

data _null_;
  file foo(shoes);
  set sashelp.shoes;
  put region $25. product $14.;
run;
```

Example 3: Reading from a Member of a Directory

This example reads the file *shoes* from the **testzip** ZIP file.

```
filename foo ZIP 'U:\directory1\testzip.zip';

data shoes;
  length region $25  product $14;
  infile foo(shoes);
  input region $25. product $14.;
run;
```

See Also

Statements:

- "FILE Statement" in *SAS DATA Step Statements: Reference*
- "FILENAME Statement" on page 19
- "INFILE Statement" in *SAS DATA Step Statements: Reference*

FOOTNOTE Statement

Writes up to 10 lines of text at the bottom of the procedure or DATA step output.

Valid in: Anywhere

Category: Output Control

Restriction: The FOOTNOTE statement does not support Unicode.

Requirement: You must specify the FOOTNOTE option if you use a FILE statement.

See: FOOTNOTE Statement under Windows, UNIX, and z/OS

Syntax

FOOTNOTE<*n* > <*ods-format-options*> <'*text*' | "*text*">;

Without Arguments

Using FOOTNOTE without arguments cancels all existing footnotes.

Arguments

n

specifies the relative line to be occupied by the footnote.

Default If you omit *n*, SAS assumes a value of 1.

Range *n* can range from 1 to 10.

Tip For footnotes, lines are pushed up from the bottom. The FOOTNOTE statement with the highest number appears on the bottom line.

ods-format-options

specifies formatting options for the ODS HTML, RTF, and PRINTER(PDF) destinations.

BOLD

specifies that the footnote text is bold font weight.

ODS destination HTML, RTF, PRINTER

COLOR=*color*

specifies the footnote text color.

Alias C

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

BCOLOR=*color*
specifies the background color of the footnote block.

ODS destination HTML, RTF, PRINTER

FONT=*font-face*
specifies the font to use. If you supply multiple fonts, then the destination device uses the first one that is installed on your system.

Alias F

ODS destination HTML, RTF, PRINTER

HEIGHT=*size*
specifies the point size.

Alias H

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

ITALIC
specifies that the footnote text is in italic style.

ODS destination HTML, RTF, PRINTER

JUSTIFY= CENTER | LEFT | RIGHT
specifies justification.

CENTER
specifies center justification.

Alias C

LEFT
specifies left justification.

Alias L

RIGHT
specifies right justification.

Alias R

Alias J

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

LINK='*url*'
specifies a hyperlink.

ODS destination HTML, RTF, PRINTER

Tip The visual properties for LINK= always come from the current style.

UNDERLIN= 0 | 1 | 2 | 3
specifies whether the subsequent text is underlined. 0 indicates no underlining. 1, 2, and 3 indicates underlining.

Alias U

ODS destination HTML, RTF, PRINTER

Tip ODS generates the same type of underline for values 1, 2, and 3. However, SAS/GRAPH uses values 1, 2, and 3 to generate increasingly thicker underlines.

Note The defaults for how ODS renders the FOOTNOTE statement come from style elements that relate to system footnotes in the current style. The FOOTNOTE statement syntax with *ods-format-options* is a way to override the settings that are provided by the current style. The current style varies according to the ODS destination. For more information about how to determine the current style, see "Understanding Styles, Style Elements, and Style Attributes" in *SAS Output Delivery System: Procedures Guide* and "Concepts: TEMPLATE Procedure" in *SAS Output Delivery System: Procedures Guide*.

Tip You can specify these options by letter, word, or words by preceding each letter or word of the *text* by the option. For example, this code makes the footnote "Red, White, and Blue" appear in different colors.

```
footnote color=red "Red," color=white "White, and" color=blue "Blue";
```

***'text'* | "*text*"**
specifies the text of the footnote in single or double quotation marks

Tips For compatibility with previous releases, SAS accepts some text without quotation marks. When you write new programs or update existing programs, *always* enclose text in quotation marks.

You can use macro variables and macros to change the information in FOOTNOTE statements. If the footnote is enclosed in double quotation marks (""), the text indicated is substituted into the footnote. If the footnote is enclosed in single quotation marks ('), the text is not substituted.

If you use single quotation marks (') or double quotation marks ("") together (with no space in between them) as the string of text, SAS writes a single quotation mark (') or double quotation mark (''), respectively.

Details

A FOOTNOTE statement takes effect when the step or RUN group with which it is associated executes. After you specify a footnote for a line, SAS repeats the same footnote on all pages until you cancel or redefine the footnote for that line. When a FOOTNOTE statement is specified for a given line, it cancels the previous FOOTNOTE statement for that line and for all footnote lines with higher numbers.

Operating Environment Information: The maximum footnote length that is allowed depends on the operating environment and the value of the LINESIZE= system option. For more information, see the SAS documentation for your operating environment.

Comparisons

You can also create footnotes with the FOOTNOTES window. For more information, refer to the online Help for the window.

You can modify footnotes with the Output Delivery System. See "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218.

Example: Using the FOOTNOTE Statement

These examples of a FOOTNOTE statement result in the same footnote:

- `footnote8 "Managers' Meeting";`
- `footnote8 'Managers'' Meeting';`

These are examples of FOOTNOTE statements that use some of the formatting options for the ODS HTML, RTF, and PRINTER(PDF) destinations. For the complete example, see "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218.

```
footnote j=left height=20pt
    color=red "Prepared "
    c='#FF9900' "on";
footnote2 j=center color=blue
   height=24pt "&SYSDATE9";
footnote3 link='http://support.sas.com/documentation/' "SAS";
```

See Also

Statements:

- "TITLE Statement" on page 212

%INCLUDE Statement

Brings a SAS programming statement, data lines, or both, into a current SAS program.

Valid in: Anywhere

Category: Program Control

Alias: %INC

See: %INCLUDE Statement under Windows , UNIX , and z/OS

Syntax

%INCLUDE *source(s)*
</<SOURCE2> <S2=*length*> <*operating-environment-options*> >;

Arguments

source(s)
describes the location of the information that you want to access with the %INCLUDE statement. There are three possible sources:

Source	Definition
file-specification	specifies an external file
internal-lines	specifies lines that are entered earlier in the same SAS job or session
keyboard-entry	specifies statements or data lines that you enter directly from the keyboard

file-specification
identifies an entire external file that you want to bring into your program.

File-specification can have these forms:

'external-file'
specifies the physical name of an external file that is enclosed in quotation marks. The physical name is the name by which the operating environment recognizes the file.

fileref
specifies a fileref that has previously been associated with an external file.

Range 1 to 8 bytes

Tip You can use a FILENAME statement or function or an operating environment command to make the association.

fileref (filename-1 <, "filename–2.xxx", ... filename-n>)
specifies a fileref that has previously been associated with an aggregate storage location. Follow the fileref with one or more filenames that reside in that location. Enclose the filenames in one set of parentheses, and separate each filename with a comma followed by a space.

This example instructs SAS to include the files testcode1.sas, testcode2.sas, and testcode3.txt. These files are located in the aggregate storage location **mylib**. You do not need to specify the file extension for testcode1 and testcode2 because they are the default .SAS extension. You must enclose testcode3.txt in quotation marks with the whole filename specified because it has an extension other than .SAS:

```
%include mylib(testcode1, testcode2,
               "testcode3.txt");
```

Operating environment Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, a text library, or a partitioned data set. For information about accessing files from a storage location that contains several files, see the SAS documentation for your operating environment.

Note A file that is located in an aggregate storage location and has a name that is not a valid SAS name must have its name enclosed in quotation marks.

Tip You can use a FILENAME statement or function or an operating environment command to make the association.

Restriction You cannot selectively include lines from an external file.

Operating environment The character length allowed for filenames is operating environment specific. For complete details about specifying the physical names of external files, see the SAS documentation for your operating environment.

Tips You can verify the existence of *file-specification* by using the SYSERR macro variable if the ERRORCHECK option is set to STRICT.

Including external sources is useful in all types of SAS processing: batch, windowing, interactive line, and noninteractive.

internal-lines

includes lines that were entered earlier in the same SAS job or session.

To include internal lines, use any of these values:

n includes line *n*.

n-m or *n:m* includes lines *n* through *m*.

Note The SPOOL system option controls internal access to previously submitted lines when you run SAS in interactive line mode, noninteractive mode, and batch mode. By default, the SPOOL system option is set to NOSPOOL. The SPOOL system option must be in effect in order to use %INCLUDE statements with internal line references. Use the OPTIONS procedure to determine the current setting of the SPOOL system option on your system.

Tips Including internal lines is most useful in interactive line mode processing.

Use a %LIST statement to determine the line numbers that you want to include.

Although you can use the %INCLUDE statement to access previously submitted lines when you run SAS in a windowing environment, it might be more practical to recall lines in the Program Editor with the RECALL command and then submit the lines with the SUBMIT command.

keyboard-entry

is a method for preparing a program so that you can interrupt the current program's execution, enter statements or data lines from the keyboard, and then resume program processing.

*

prompts you to enter data from the keyboard. Place an asterisk (*) after the %INCLUDE statement in your code: To resume processing the original source program, enter a %RUN statement from the keyboard.

```
proc print;
   %include *;
run;
```

Restriction The asterisk (*) cannot be used to specify keyboard entry if you use the Enhanced Editor in the Microsoft Windows operating environment.

Note The fileref SASTERM must have been previously associated with an external file in a FILENAME statement or function or an operating environment command.

Tips Use this method when you run SAS in noninteractive or interactive line mode. SAS pauses during processing and prompts you to enter statements from the keyboard.

Use this argument to include source from the keyboard:

You can use a %INCLUDE * statement in a batch job by creating a file with the fileref SASTERM that contains the statements that you would otherwise enter from the keyboard. The %INCLUDE * statement causes SAS to read from the file that is referenced by SASTERM. Insert a %RUN statement into the file that is referenced by SASTERM where you want SAS to resume reading from the original source.

SOURCE2

causes the SAS log to show the source statements that are being included in your SAS program.

Tips The SAS log also displays the fileref and the filename of the source and the level of nesting (1, 2, 3, and so on).

The SOURCE2 system option produces the same results. When you specify SOURCE2 in a %INCLUDE statement, it overrides the setting of the SOURCE2 system option for the duration of the include operation.

S2=*length*

specifies the length of the record to be used for input. *Length* can have these values:

S sets S2 equal to the current setting of the S= SAS system option.

0 tells SAS to use the setting of the SEQ= system option to determine whether the line contains a sequence field. If the line does contain a sequence field, SAS determines line length by excluding the sequence field from the total length.

n specifies a number greater than zero that corresponds to the length of the line to be read, when the file contains fixed-length records. When the file contains variable-length records, *n* specifies the column in which to begin reading data.

Interaction The S2= system option also specifies the length of secondary source statements that are accessed by the %INCLUDE statement, and it is effective for the duration of your SAS session. The S2= option in the %INCLUDE statement affects only the current include operation. If you use the S2= option in the %INCLUDE statement, it overrides the S2= system option setting for the duration of the include operation.

Tips Text input from the %INCLUDE statement can be either fixed or variable length.

Fixed-length records are either unsequenced or sequenced at the end of each record. For fixed-length records, the value given in S2= is the ending column of the data.

Variable-length records are either unsequenced or sequenced at the beginning of each record. For variable-length records, the value given in S2= is the starting column of the data.

See For a detailed discussion of fixed-length and variable-length input records, see "S= System Option" in *SAS System Options: Reference* and "S2= System Option" in *SAS System Options: Reference*.

operating-environment-options

Operating environment Operating environments can support various options for the %INCLUDE statement. See the documentation for your operating environment for a list of these options and their functions.

Details

What %INCLUDE Does

When you execute a program that contains the %INCLUDE statement, SAS executes your code, including any statements or data lines that you bring into the program with %INCLUDE.

Operating Environment Information: Use of the %INCLUDE statement is dependent on your operating environment. Before you run the examples for this statement and for more information about additional software features and methods of referring to and accessing your files, see the documentation for your operating environment.

Three Sources of Data

The %INCLUDE statement accesses SAS statements and data lines from three possible sources:

- external files
- lines entered earlier in the same job or session
- lines entered from the keyboard

Uses of %INCLUDE

The %INCLUDE statement is most often used when running SAS in interactive line mode, noninteractive mode, or batch mode.

TIP You can use the INCLUDE and RECALL commands to quickly and efficiently access and re-submit data lines and program statements.

Rules for Using %INCLUDE

- You can specify any number of sources in a %INCLUDE statement, and you can mix the types of included sources. However, although it is possible to include information from multiple sources in one %INCLUDE statement, it might be easier to understand a program that uses separately coded %INCLUDE statements for each source.
- When used in the DATA step, the %INCLUDE statement must be the first statement or it must immediately follow a semicolon that ends another statement. This restriction does not apply if the %INCLUDE statement is used with the macro facility.
- When used in the DATA step, the %INCLUDE statement cannot be used in conditional logic. However, you can use the %INCLUDE statement with conditional logic when used with the macro facility. For example, you can specify the following %IF-%THEN macro statement:

```
%if &error=1 %then %do;
   %include "myfile";
%end;
```

- The maximum line length is 32K bytes.

Comparisons

The %INCLUDE statement executes statements immediately. The INCLUDE command brings the included lines into the Program Editor window but does not execute them. You must issue the SUBMIT command to execute those lines.

Examples

Example 1: Including an External File

- This example stores a portion of a program in a file named MYFILE. The file can be included in a program that is written later.

```
data monthly;
   input x y month $;
   datalines;
1 1 January
2 2 February
3 3 March
4 4 April
;
```

This program includes an external file named MYFILE and submits the DATA step that it contains before the PROC PRINT step executes.

```
%include 'MYFILE';
proc print;
run;
```

- To reference a file by using a fileref rather than the actual filename, you can use the FILENAME statement (or a command recognized by your operating environment) to assign a fileref.

  ```
  filename in1 'MYFILE';
  ```

  Later, you can access MYFILE with the fileref IN1.

  ```
  %inc in1;
  ```

- If you want to use many files that are stored in a directory, you can assign a fileref to the directory location and then specify the filename in the %INCLUDE statement. For example, this FILENAME statement assigns the fileref `storage` to an aggregate storage location.

  ```
  filename storage
     'aggregate-storage-location';
  ```

  Later, you can include a file using this statement.

  ```
  %inc storage(MYFILE);
  ```

- You can also access several files or members from this storage location by listing them in parentheses after the fileref in a single %INCLUDE statement. Separate filenames with a comma or a blank space. This %INCLUDE statement demonstrates this method.

  ```
  %inc storage(file-1,file-2,file-3);
  ```

  When the file does not have the default .SAS extension, you can access it using quotation marks around the complete filename listed inside the parentheses.

-

  ```
  %inc storage("file-1.txt","file-2.dat",
     "file-3.cat");
  ```

Example 2: Including Previously Submitted Lines

This %INCLUDE statement causes SAS to process lines 1, 5, 9 through 12, and 13 through 16 as if you had entered them again from your keyboard.

```
%include 1 5 9-12 13:16;
```

Example 3: Including Input from the Keyboard

The method shown in this example is valid only when you run SAS in noninteractive mode or interactive line mode.

Restriction: The asterisk (*) cannot be used to specify keyboard entry if you use the Enhanced Editor in the Microsoft Windows operating environment.

This example uses %INCLUDE to add a customized TITLE statement when PROC PRINT executes.

```
data report;
   infile file-specification;
   input month $ salesamt $;
run;
proc print;
   %include *;
```

```
run;
```

When this DATA step executes, %INCLUDE with the asterisk causes SAS to issue a prompt for statements that are entered at the keyboard. For example:

```
where month= 'January';
title 'Data for month of January';
```

After you enter statements, you can use %RUN to resume processing by entering `%run;`.

The %RUN statement signals to SAS to leave keyboard-entry mode and resume reading and executing the remaining SAS statements from the original program.

Example 4: Using %INCLUDE with Several Entries in a Single Catalog

This example submits the source code from three entries in the catalog MYLIB.INCLUDE. When no entry type is specified, the default is CATAMS.

```
filename dir catalog 'mylib.include';
%include dir(mem1);
%include dir(mem2);
%include dir(mem3);
```

See Also

Statements:

- "%LIST Statement" on page 189
- "%RUN Statement" on page 203

LIBNAME Statement

Associates or disassociates a SAS library with a libref (a shortcut name), clears one or all librefs, lists the characteristics of a SAS library, concatenates SAS libraries, or concatenates SAS catalogs.

Valid in: Anywhere

Category: Data Access

Restriction: When SAS is in a locked-down state, the LIBNAME statement is not available for files that are not in the lockdown path list. For more information, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Programmer's Guide: Essentials*.

See: LIBNAME Statement under Windows, UNIX, and z/OS

Syntax

Form 1: **LIBNAME** *libref* <*engine*> '*SAS-library*'
< *libname-options* > <*engine-host-options*>;

Form 2: **LIBNAME** *libref* CLEAR | _ALL_ CLEAR ;

Form 3: **LIBNAME** *libref* LIST | _ALL_ LIST;

Form 4: **LIBNAME** *libref* <*engine*> (*library-specification-1* <...*library-specification-n*>)
<*libname-options*>;

Required Arguments

libref
is a shortcut name or a "nickname" for the aggregate storage location where your SAS files are stored. It is any SAS name when you are assigning a new libref. When you are disassociating a libref from a SAS library or when you are listing attributes, specify a libref that was previously assigned.

Range 1 to 8 bytes

Tip The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change it or discontinue it with another LIBNAME statement.

See A libref follows the same rules of syntax as any SAS name. For more information about SAS naming conventions, see "Words and Names" in *SAS Programmer's Guide: Essentials*.

'SAS-library'
must be the physical name for the SAS library. The physical name is the name that is recognized by the operating environment. Enclose the physical name in single or double quotation marks.

Operating environment For more information about specifying the physical names of files, see the SAS documentation for your operating environment.

library-specification
is two or more SAS libraries that are specified by physical names, previously assigned librefs, or a combination of the two. Separate each specification with either a blank or a comma and enclose the entire list in parentheses.

'SAS-library'
is the physical name of a SAS library, enclosed in quotation marks.

libref
is the name of a previously assigned libref.

Restriction When concatenating libraries, you cannot specify options that are specific to an engine or an operating environment.

See "Rules for Library Concatenation" on page 153

Example "Example 2: Logically Concatenating SAS Libraries" on page 155

CLEAR
disassociates one or more currently assigned librefs.

Tip Specify *libref* to disassociate a single libref. Specify _ALL_ to disassociate all currently assigned librefs.

ALL
specifies that the CLEAR or LIST argument applies to all currently assigned librefs.

LIST
writes the attributes of one or more SAS libraries to the SAS log.

Tip Specify *libref* to list the attributes of a single SAS library. Specify _ALL_ to list the attributes of all SAS libraries that have librefs in your current session.

Optional Arguments

libname-options can be any of these values:

ACCESS=READONLY | TEMP

READONLY
assigns a read-only attribute to an entire SAS library. SAS does not allow you to open a data set in the library in order to update information or write new information.

TEMP
specifies that the SAS library be treated as a scratch library. That is, the system does not consume CPU cycles to ensure that the files in a Temp library do not become corrupted.

Tip Use ACCESS=TEMP to save resources only when the data is recoverable.

Operating environment Some operating environments support LIBNAME statement options that have similar functions to the ACCESS= option. See the SAS documentation for your operating environment.

AUTHADMIN= YES | NO
specifies whether an administrator can access a metadata-bound library for which corresponding metadata is corrupted, misconfigured, or missing.

Default NO

Restriction This LIBNAME option can be used only by administrators of metadata-bound libraries.

Interactions If the administrator specifies AUTHADMIN=YES in a LIBNAME statement and knows the password (or passwords) for the target

data, the administrator can access that data by explicitly supplying the password (or passwords).

An administrator can choose to specify the AUTHPW= option in the LIBNAME statement as an additional method for making the metadata-bound library password available to later requests.

Note The use of AUTHADMIN=YES is intended for the administrator to correct misaligned location and metadata information. To ensure that the user who is issuing the LIBNAME statement has administrator rights to correct the misalignments, the user must have the same permissions that are needed to run the AUTHLIB procedure statements and must supply the metadata-bound data passwords when accessing the data sets.

Tip The AUTHLIB REPAIR statement is preproduction. It is recommended that you use AUTHADMIN=YES when performing any AUTHLIB REPAIR action.. As a best practice, do not use AUTHADMIN=YES in any other circumstance.

See "AUTHPW=password"

"Metadata-Bound Libraries" on page 154

SAS Guide to Metadata-Bound Libraries

PROC AUTHLIB in *Base SAS Procedures Guide*

AUTHALTER=*alter-password*

Specifies an ALTER password to use only in data access requests where both of these conditions exist:

- AUTHADMIN=YES is specified in the LIBNAME statement that is referenced in the request.
- The correct password for the target metadata-bound data set or library is not otherwise available or is invalid.

Requirement The AUTHADMIN option must be set to YES for this option to have an effect.

Interaction You can use the AUTHALTER= option in the same way as the AUTHPW= option if all three of the passwords (ALTER, READ, and WRITE) are the same.

See *SAS Guide to Metadata-Bound Libraries*

AUTHPW=*password*

Specifies a password to use only in data access requests where both of these conditions exist:

- AUTHADMIN=YES is specified in the LIBNAME statement that is referenced in the request or is invalid.
- The correct password for the target metadata-bound library is not otherwise available.

Requirement The AUTHADMIN option must be set to YES for this option to have an effect. However, the use of AUTHAMDIN=YES does not require that you use AUTHPW. You are not required to specify metadata-bound library passwords in a LIBNAME statement.

Interactions If the metadata-bound library has two or three distinct passwords, you must specify each individual password with the AUTHALTER=, AUTHREAD=, and AUTHWRITE= options as appropriate instead of using the AUTHPW= option on its own.

You can use the AUTHALTER= option in the same way as the AUTHPW= option if all three of the passwords (ALTER, READ, and WRITE) are the same and you are in a SAS language context where ALTER= can be used.

Tip An error occurs if the AUTHPW password does not match the password that is within the referenced secured library object.

See *SAS Guide to Metadata-Bound Libraries*

AUTHREAD=*read-password*

Specifies a READ password to use only in data access requests where both of these conditions exist:

- AUTHADMIN=YES is specified in the LIBNAME statement that is referenced in the request.
- The correct password for the target metadata-bound library is not otherwise available or is invalid.

Requirement The AUTHADMIN option must be set to YES for this option to have an effect.

See *SAS Guide to Metadata-Bound Libraries*

AUTHWRITE=*write-password*

Assigns a WRITE password to a metadata-bound library that prevents users from writing to a library, unless they enter the password.

Requirement The AUTHADMIN option must be set to YES for this option to have an effect.

See *SAS Guide to Metadata-Bound Libraries*

COMPRESS=NO | CHAR | BINARY

controls the compression of observations in output SAS data sets for a SAS library.

NO

specifies that the observations in a newly created SAS data set be uncompressed (fixed-length records).

CHAR

specifies that the observations in a newly created SAS data set be compressed (variable-length records) by SAS using RLE (Run Length

Encoding). RLE compresses observations by reducing repeated consecutive characters (including blanks) to two-byte or three-byte representations.

Alias YES

Tip Use this compression algorithm for character data.

BINARY
specifies that the observations in a newly created SAS data set be 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 variables). 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.

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 in the COMPRESS= option. For more information about CLONE and NOCLONE, see COPY Statement in the DATASETS procedure. This interaction does not apply when using SAS/SHARE or SAS/CONNECT.

See Compression in *SAS® V9 LIBNAME Engine: Reference.*

CVPBYTES=*bytes*
specifies the number of bytes by which to expand character variable lengths when processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by adding the specified value to the current length. You can specify a value from 0 to 32,766.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPBYTES= option:

```
libname expand 'SAS data-library' cvpbytes=5;
```

Character variable lengths are increased by adding 5 bytes. A character variable with a length of 10 is increased to 15, and a character variable with a length of 100 is increased to 105.

Default If you specify CVPBYTES=, SAS automatically uses the CVP engine to expand the character variable lengths according to your specification. If you explicitly assign the CVP engine but do not specify either CVPBYTES= or CVPMULTIPLIER=, then SAS uses CVPMULTIPLIER=1.5 to increase the lengths of the character variables.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error message in the SAS log.

Interaction You cannot specify both the CVPBYTES= option and the CVPMULTIPLIER= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPENGINE=*engine*

specifies the engine to use to process a SAS data file that requires transcoding. The CVP engine expands the character variable lengths to transcoding so that character data truncation does not occur. Then the specified engine processes the actual file.

Alias CVPENG

Default SAS uses the default SAS engine.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPFORMATWIDTH=YES | NO

specifics whether to expand the character format width.

If CVPVARCHAR= is not specified, the new format width is determined by the CVPMULTIPILER= and CVPBYTES= options.

If CVPVARCHAR= is specified, the CVP engine automatically adjusts the format width to meet the maximum-byte length of a converted character variable. For example, in a UTF-8 session, the format width is multiplied by 4.

Alias CVPFMTW

Default YES

CVPMULTIPLIER=*multiplier*

specifies a multiplier value that expands character variable lengths when you are processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by multiplying the current length by the specified value. You can specify a multiplier value from 1 to 5 or you can specify 0 and then the CVP engine determines the multiplier automatically.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPMULTIPLIER= option:

```
libname expand 'SAS data-library' cvpmultiplier=2.5;
```

Character variable lengths are increased by multiplying the lengths by 2.5. A character variable with a length of 10 is increased to 25, and a character variable with a length of 100 is increased to 250.

Alias CVPMULT

Default If you specify the CVPMULTIPLIER= option, SAS automatically uses the CVP engine to expand the character variable lengths according to your specification. If you explicitly specify the CVP engine but do not specify either the CVPMULTIPLIER= option or the CVPBYTES= option, then SAS uses CVPMULTIPLIER=AUTO(0) to increase the lengths. AUTO(0) sets the value of the CVP engine based on the encoding of the SAS session and input data set.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error in the SAS log.

Interaction You cannot specify both the CVPMULTIPLIER= option and the CVPBYTES= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPVARCHAR=YES | NO

specifies whether to convert fixed-width character variables to variable-width characters during input file processing. The byte length of the new-width character variable is the maximum number of bytes per character from the SAS session encoding multiplied by the specified fixed-width character length.

Default No

Interaction If you specify CVPVARCHAR=YES, the CVPMULTIPLIER= and CVPBYTES= options are ignored.

Notes Trailing blanks are removed from string data that is under CHAR columns.

Fixed-width character variables with a format of TRANSCODE=NO are excluded during conversion.

EXTENDOBSCOUNTER=YES | NO

specifies whether to extend the maximum observation count in output SAS data files for a SAS library.

YES
specifies to use the default maximum observation count of 2^{63}–1 or approximately 9.2 quintillion observations in a newly created SAS data file.

Restriction A SAS data file that has the extended observation count attribute cannot be used by releases prior to SAS 9.3. To remove the extended observation count attribute, the file must be re-created. If you plan to use the file in SAS 9.2 or earlier releases, then set EXTENDOBSCOUNTER=NO when you create the file.

Interaction EXTENDOBSCOUNTER=YES is ignored if the data representation is a 64-bit UNIX operating environment.

NO
is a compatibility setting for a newly created SAS data file, to enable its use in releases prior to SAS 9.3.

- Under UNIX, if you specify OUTREP= and plan to use the file in SAS 9.2 or earlier releases, specify EXTENDOBSCOUNTER=NO. If you do not specify OUTREP=, then you do not need to specify EXTENDOBSCOUNTER=NO.
- Under Windows or z/OS, if you plan to use the file in SAS 9.2 or earlier releases, specify EXTENDOBSCOUNTER=NO.

Alias EOC=

Defaults Under UNIX environments, by default the EXTENDOBSCOUNTER= option is not set. The extended observation count feature is not necessary under 64-bit UNIX. However, if you specify the OUTREP= option, and the data representation is not a 64-bit UNIX operating environment, then SAS automatically sets EXTENDOBSCOUNTER=YES. SAS adds the extended observation count feature for compatibility with environments other than UNIX where it might be necessary.

Under Windows and z/OS, by default EXTENDOBSCOUNTER=YES. Files are created with the enhanced file format and the extended observation count attribute.

Restrictions Use with output data files only.

Use with the BASE engine only.

If you copy a file, the extended observation count attribute is not inherited.

See Understanding the Observation Count in *SAS® V9 LIBNAME Engine: Reference.*

INENCODING=ANY | ASCIIANY | EBCDICANY | *encoding-value*
overrides the encoding when you are reading (input processing) SAS data sets in the SAS library.

See "INENCODING=, OUTENCODING= Options Statements" in *SAS National Language Support (NLS): Reference Guide*

LIBRARYDEFINITION=*source-definition-URI*
retrieves pre-defined engine LIBNAME information from the Data Sources microservice and performs a LIBNAME assignment using that information.

Here is an example:

```
libname x
   LIBDEF="/dataSources/providers/Compute/
      sourceDefinitions/a22d983d-c324-442c-a605-06758af9aa6d"
   access=readonly;
```

Note: The SERVICESBASEURL= system option must be set during the invocation of your SAS session. This option specifies the host and port for file service requests. The access method fails at assignment time if the SERVICESBASEURL= option is not specified. For more information, see "SERVICESBASEURL= System Option" in *SAS System Options: Reference*.

Note: The SAS_VIYA_TOKEN environment variable must be defined. For more information about defining this variable, see "SAS_VIYA_TOKEN Environment Variable" in *Encryption in SAS*.

Alias LIBDEF=

Restriction This is option is available only in SAS Viya 3.4.

Requirement LIBDEF= must be the first option specified after the libref.

OUTENCODING=
OUTENCODING=ANY | ASCIIANY | EBCDICANY | *encoding-value*

overrides the encoding when you are creating (output processing) SAS data sets in the SAS library.

See "INENCODING=, OUTENCODING= Options Statements" in *SAS National Language Support (NLS): Reference Guide*

OUTREP=*format*
specifies the data representation, which is the form in which data is stored in a particular operating environment. Different operating environments use different standards or conventions for storing data.

- Floating-point numbers can be represented in IEEE floating-point format or IBM floating-point format.
- Data alignment can be on a 1-byte, 4-byte, or 8-byte boundary, depending on data type requirements for the operating environment.
- Data type lengths can be 8 bits or more for a character data type, 16 bit, 32 bit, or 64 bit for an integer data type, 32 bit for a single-precision floating-point data type, and 64 bit for a double-precision floating-point data type.

- The ordering of bytes can be big Endian or little Endian.

By default, SAS creates a new SAS data set by using the data representation of the CPU that is running SAS. Specifying the OUTREP= option enables you to create a SAS data set with a different data representation. For example, in a UNIX environment, you can create a SAS data set that uses a Windows data representation. For more information about compatibility and data representation, see "Cross-Environment Data Access" in *SAS Programmer's Guide: Essentials*.

Values for OUTREP= are listed in this table.

Table 2.2 *Data Representation Values for OUTREP= Option*

OUTREP= Value	Alias[1]	Environment
ALPHA_VMS_64		OpenVMS Alpha
HP_IA64	HP_ITANIUM	HP-UX for the Itanium Processor Family Architecture
HP_UX_32	HP_UX	HP-UX for PA-RISC
HP_UX_64		HP-UX for PA-RISC, 64-bit
LINUX_32	LINUX	Linux for Intel architecture
LINUX_X86_64		Linux for x64
LINUX_POWER_64		Linux on the Power Architecture[2]
MIPS_ABI		MIPS ABI
MVS_32	MVS	31-bit SAS on z/OS
MVS_64_BFP		64-bit SAS on z/OS
RS_6000_AIX_32	RS_6000_AIX	AIX
RS_6000_AIX_64		AIX
SOLARIS_32	SOLARIS	Solaris for SPARC
SOLARIS_64		Solaris for SPARC
SOLARIS_X86_64		Solaris for x64
VMS_IA64		OpenVMS on HP Integrity
WINDOWS_32	WINDOWS	32-bit SAS on Microsoft Windows

OUTREP= Value	Alias[1]	Environment
WINDOWS_64		64-bit SAS on Microsoft Windows (for both Itanium-based systems and x64)

1 It is recommended that you use the current values. The aliases are available for compatibility only.
2 LINUX_POWER_64 is added in SAS Viya 3.5. It is not supported in SAS 9.

Interactions By default, PROC COPY uses the data representation of the file from the source library. If, instead, you want to use the data representation of the current SAS session, specify the NOCLONE option. If you want to use a different data representation, specify the NOCLONE option and the OUTREP= option. When you use PROC COPY with SAS/SHARE or SAS/CONNECT, the default behavior is to use the data representation of the current SAS session. For more information about CLONE and NOCLONE, see COPY Statement in the *Base SAS Procedures Guide*.

The COPY procedure (with NOCLONE) and the MIGRATE procedure can use the LIBNAME option OUTREP= for DATA, VIEW, ACCESS, MDDB, and DMDB member types. Otherwise, only DATA member types are affected by the OUTREP= LIBNAME option.

When you use the OUTREP= LIBNAME statement option, the default encoding is based on the operating environment that is represented by the OUTREP= value and the locale of the current SAS session. To assign a nondefault encoding such as UTF-8, you must also specify the OUTENCODING= LIBNAME statement option. For more information about locale and encoding, see *SAS National Language Support (NLS): Reference Guide*.

Transcoding could result in character data loss when encodings are incompatible. For more information, see *SAS National Language Support (NLS): Reference Guide*.

If you specify the OUTREP= option, and you plan to use the file in an earlier release of SAS, you might also want to specify the EXTENDOBSCOUNTER= option. See "EXTENDOBSCOUNTER=YES | NO" on page 146.

POINTOBS=YES | NO
specifies whether SAS creates compressed data sets whose observations can be randomly accessed or sequentially accessed.

YES
causes SAS software to produce a compressed data set that might be randomly accessed by observation number.

Note For an individual data set, the POINTOBS= data set option overrides the setting of the POINTOBS= option in the LIBNAME statement.

Tip Specifying POINTOBS=YES does not affect the efficiency of retrieving information from a data set. It does increase CPU usage by approximately 10% when creating a compressed data set and when updating or adding information to it.

NO
suppresses the ability to randomly access observations in a compressed data set by observation number.

Tip Specifying POINTOBS=NO is desirable for applications where the ability to point directly to an observation by number within a compressed data set is not important. If you do not need to access data by observation number, then you can improve performance by approximately 10% by specifying POINTOBS=NO when creating a compressed data set or when updating or adding observations to it.

Default YES

REPEMPTY=YES | NO
controls replacement of SAS data sets when the new one is empty.

YES
specifies that a new empty data set replaces an existing data set. This is the default.

Interaction If REPEMPTY=YES and REPLACE=NO, then the data set is not replaced.

NO
specifies that a new empty data set does not replace an existing data set.

Here is an example where setting REPEMPTY=NO prevents the empty data set B from replacing the data set MYLIB.A:

```
libname mylib 'c:\mydata' repempty=no;
data mylib.a;
   i=1;
run;

data b; /* create an empty data set B */
run;

data mylib.a;
   set b;
run;
```

Here is the log:

```
WARNING: Data set MYLIB.A was not replaced because REPEMPTY=NO and the replacement file is e
```

Tip For both the convenience of replacing existing data sets with new ones that contain data and the protection of not overwriting existing data sets with new empty ones that are created by mistake, set REPLACE=YES and REPEMPTY=NO.

Note For an individual data set, the REPEMPTY= data set option overrides the setting of the REPEMPTY= option in the LIBNAME statement.

See "REPEMPTY= Data Set Option" in *SAS Data Set Options: Reference*

Engines

engine
is an engine name.

Tip Usually, SAS automatically determines the appropriate engine to use for accessing the files in the library. If you want to create a new library with an engine other than the default engine, then you can override the automatic selection.

See LIBNAME Statement under Windows, UNIX, and z/OS

For more information, see "SAS Engines" in *SAS Programmer's Guide: Essentials*.

Engine Host Options

engine-host-options
are one or more options that are listed in the general form *keyword=value*. For a list of engines, see the SAS documentation for your operating system.

Restriction When concatenating libraries, you cannot specify options that are specific to an engine or an operating environment.

See LIBNAME Statement under Windows, UNIX, and z/OS

Details

Associating a Libref with a SAS Library (*Form 1*)

The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change the libref or discontinue it with another LIBNAME statement. The simplest form of the LIBNAME statement specifies only a libref and the physical name of a SAS library:

LIBNAME *libref 'SAS-library'*;

See "Example 1: Assigning and Using a Libref" on page 155.

An engine specification is usually not necessary. If the situation is ambiguous, SAS uses the setting of the ENGINE= system option to determine the default engine. If all data sets in the library are associated with a single engine, then SAS uses that engine as the default. In either situation, you can override the default by specifying another engine with the ENGINE= system option:

LIBNAME *libref engine 'SAS-library'*

<*options*> <*engine/host-options*>;

Operating Environment Information: Using the LIBNAME statement requires host-specific information. See the SAS documentation for your operating environment before using this statement.

Disassociating a Libref from a SAS Library (*Form 2*)

To disassociate a libref from a SAS library, use a LIBNAME statement by specifying the libref and the CLEAR option. You can clear a single, specified libref or all current librefs.

LIBNAME *libref* CLEAR | _ALL_ CLEAR;

Writing SAS Library Attributes to the SAS Log (*Form 3*)

Use a LIBNAME statement to write the attributes of one or more SAS libraries to the SAS log. Specify *libref* to list the attributes of one SAS library; use _ALL_ to list the attributes of all SAS libraries that have been assigned librefs in your current SAS session.

LIBNAME *libref* LIST | _ALL_ LIST;

Concatenating SAS Libraries (*Form 4*)

When you logically concatenate two or more SAS libraries, you can reference them all with one libref. You can specify a library with its physical filename or its previously assigned libref.

LIBNAME *libref* <*engine*> (*library–specification-1* <*...library-specification-n*>)
< *options*>;

In the same LIBNAME statement, you can use any combination of specifications: librefs, physical filenames, or a combination of librefs and physical filenames. See "Example 2: Logically Concatenating SAS Libraries" on page 155.

Concatenating SAS Catalogs (*Form 4*)

When you logically concatenate two or more SAS libraries, you also concatenate the SAS catalogs that have the same name. For example, if three SAS libraries each contain a catalog named CATALOG1, then when you concatenate them, you create a catalog concatenation for the catalogs that have the same name. See "Example 3: Concatenating SAS Catalogs" on page 156.

LIBNAME *libref* <*engine*> (*library–specification-1* <*...library-specification-n*>)
< *options* >;

Rules for Library Concatenation

After you create a library concatenation, you can specify the libref in any context that accepts a simple (non-concatenated) libref. These rules determine how SAS files (that is, members of SAS libraries) are located among the concatenated libraries:

- When a SAS file is opened for input or update, the concatenated libraries are searched and the first occurrence of the specified file is used.
- When a SAS file is opened for output, it is created in the first library that is listed in the concatenation.

  Note: A new SAS file is created in the first library even if there is a file with the same name in another part of the concatenation.

- When you delete or rename a SAS file, only the first occurrence of the file is affected.
- Anytime a list of SAS files is displayed, only one occurrence of a filename is shown.

  Note: Even if the name occurs multiple times in the concatenation, only the first occurrence is shown.

- A SAS file that is logically connected to another file (such as an index to a data set) is listed only if the parent file resides in that same library. For example, if library One contains A.DATA, and library Two contains A.DATA and A.INDEX, only A.DATA from library One is listed. (See the previous rule.)
- If any library in the concatenation is sequential, then all of the libraries are treated as sequential.
- The attributes of the first library that is specified determine the attributes of the concatenation. For example, if the first SAS library that is listed is "read only," then the entire concatenated library is "read only."
- If you specify any options or engines, they apply only to the libraries that you specified with the complete physical name, not to any library that you specified with a libref.
- If you alter a libref after it has been assigned in a concatenation, it does not affect the concatenation.

Automatically Creating the Library Directory

You can set the DLCREATEDIR system option to create the directory for the SAS library that is specified in the LIBNAME statement if that directory does not exist. For more information, see "DLCREATEDIR System Option" in *SAS System Options: Reference*.

z/OS Specifics: For more information, see "DLCREATEDIR System Option: z/OS" in *SAS Companion for z/OS*.

Metadata-Bound Libraries

The Base SAS LIBNAME engine can enforce permissions on a user and group basis to SAS data sets that are bound to secured table objects in the metadata server. Metadata-bound libraries provide enhanced protection for Base SAS data (SAS data sets and SAS views). A connection to the metadata server is required in order to access metadata-bound data.

For more information, see *SAS Guide to Metadata-Bound Libraries* and the AUTHLIB procedure in *Base SAS Procedures Guide*.

If you have questions or need assistance accessing your data, contact your local SAS Administrator.

Comparisons

- Use the LIBNAME statement to reference a SAS library. Use the FILENAME statement to reference an external file. Use the LIBNAME, SAS/ACCESS statement to access DBMS tables.
- Use the CATNAME statement to concatenate SAS catalogs. Use the LIBNAME statement to concatenate SAS catalogs. The CATNAME statement enables you to specify the names of the catalogs that you want to concatenate. The LIBNAME statement concatenates all like-named catalogs in the specified SAS libraries.

Examples

Example 1: Assigning and Using a Libref

This example assigns the libref SALES to an aggregate storage location that is specified in quotation marks as a physical filename. The DATA step creates SALES.QUARTER1 and stores it in that location. The PROC PRINT step references it by its two-level name, SALES.QUARTER1.

```
libname sales 'SAS-library';
data sales.quarter1;
infile 'your-input-file';
input salesrep $20. +6 jansales febsales
   marsales;
run;
proc print data=sales.quarter1;
run;
```

Example 2: Logically Concatenating SAS Libraries

- This example concatenates three SAS libraries by specifying the physical filename of each:

```
libname allmine ('file-1' 'file-2' 'file-3');
```

- This example assigns librefs to two SAS libraries, one that contains SAS 6 files and one that contains SAS 9 files. This technique is useful for updating your files and applications from SAS 6 to SAS 9 and enables you to have convenient access to both sets of files:

```
libname v6 'v6-SAS-library';
libname v9 'v9-SAS-library';
libname allmine (v9 v6);
```

- This example shows that you can specify both librefs and physical filenames in the same concatenation specification:

```
libname allmine (v9 v6 'some-filename');
```

Example 3: Concatenating SAS Catalogs

This example concatenates three SAS libraries by specifying the physical filename of each and assigns the libref ALLMINE to the concatenated libraries:

```
libname allmine ('file-1' 'file-2' 'file-3');
```

If each library contains a SAS catalog named MYCAT, then using ALLMINE.MYCAT as a libref.catref provides access to the catalog entries that are stored in all three catalogs named MYCAT. To logically concatenate SAS catalogs with different names, see "CATNAME Statement" on page 9.

Example 4: Permanently Storing Data Sets with One-Level Names

If you want the convenience of specifying only a one-level name for permanent, not temporary, SAS files, then use the USER= system option. This example stores the data set QUARTER1 permanently without using a LIBNAME statement first to assign a libref to a storage location:

```
options user='SAS-library';
data quarter1;
infile 'your-input-file';
input salesrep $20. +6 jansales febsales
   marsales;
run;
proc print data=quarter1;
run;
```

See Also

- To determine the maximum number of characters for a SAS name that is measure in bytes, see "Extending SAS Names" in *SAS Programmer's Guide: Essentials*.

Data Set Options:

- "ENCODING= Data Set Option" in *SAS National Language Support (NLS): Reference Guide*

Statements:

- "CATNAME Statement" on page 9 for a discussion of concatenating SAS catalogs
- "FILENAME Statement" on page 19

- LIBNAME option character variable attributes used to transcode SAS files
- "LIBNAME Statement: SASEDOC" in *SAS Output Delivery System: User's Guide*
- LIBNAME Statement for SAS metadata
- LIBNAME Statement for Scalable Performance Data (SPD)
- LIBNAME statement for XML documents
- LIBNAME Statement for SAS/ACCESS
- LIBNAME Statement for SAS/CONNECT
- "LIBNAME: SASESOCK Engine" in *SAS/CONNECT User's Guide*
- LIBNAME Statement for SAS/SHARE
- "LOCKDOWN Statement" in SAS Intelligence Platform: Application Server Administration Guide

System Options:

- "DLCREATEDIR System Option" in *SAS System Options: Reference*
- "LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide
- "USER= System Option" in *SAS System Options: Reference*

LIBNAME Statement: CVP Engine

Associates a libref for the character variable padding (CVP) engine to expand character variable lengths so that character data truncation does not occur when a file requires transcoding.

Valid in: Anywhere

Category: Data Access

Restrictions: The CVP engine is available for input (read) processing only.

The CVP engine supports SAS data files, but the engine does not support SAS views, catalogs, or item stores.

The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error message in the SAS log.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Syntax

LIBNAME *libref* CVP '*SAS-library*' <*libname-options*>;

Required Arguments

libref
is a character constant, variable, or expression that specifies the libref that is assigned to a SAS library.

Range 1 to 8 bytes

SAS-library
is the physical name for the SAS library. The physical name is recognized by the operating environment. Enclose the physical name in single or double quotation marks.

Optional Argument

libname-options
libname-options can be any of these values:

CVPBYTES=*bytes*
specifies the number of bytes by which to expand character variable lengths when processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by adding the specified value to the current length. You can specify a value from 0 to 32,766.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPBYTES= option:

```
libname expand 'SAS data-library' cvpbytes=5;
```

Character variable lengths are increased by adding 5 bytes. A character variable with a length of 10 is increased to 15, and a character variable with a length of 100 is increased to 105.

Default If you specify CVPBYTES=, SAS automatically uses the CVP engine to expand the character variable lengths according to your specification. If you explicitly assign the CVP engine but do not specify either CVPBYTES= or CVPMULTIPLIER=, then SAS uses CVPMULTIPLIER=1.5 to increase the lengths of the character variables.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error message in the SAS log.

Interaction You cannot specify both the CVPBYTES= option and the CVPMULTIPLIER= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPENGINE=*engine*
specifies the engine to use to process a SAS data file that requires transcoding. The CVP engine expands the character variable lengths to transcoding so that character data truncation does not occur. Then the specified engine processes the actual file.

Alias CVPENG

Default SAS uses the default SAS engine.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPFORMATWIDTH=YES | NO
specifics whether to expand the character format width.

If CVPVARCHAR= is not specified, the new format width is determined by the CVPMULTIPILER= and CVPBYTES= options.

If CVPVARCHAR= is specified, the CVP engine automatically adjusts the format width to meet the maximum-byte length of a converted character variable. For example, in a UTF-8 session, the format width is multiplied by 4.

Alias CVPFMTW

Default YES

CVPMULTIPLIER=*multiplier*
specifies a multiplier value that expands character variable lengths when you are processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by multiplying the current length by the specified value. You can specify a multiplier value from 1 to 5 or you can specify 0 and then the CVP engine determines the multiplier automatically.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPMULTIPLIER= option:

```
libname expand 'SAS data-library' cvpmultiplier=2.5;
```

Character variable lengths are increased by multiplying the lengths by 2.5. A character variable with a length of 10 is increased to 25, and a character variable with a length of 100 is increased to 250.

Alias CVPMULT

Default If you specify the CVPMULTIPLIER= option, SAS automatically uses the CVP engine to expand the character variable lengths

according to your specification. If you explicitly specify the CVP engine but do not specify either the CVPMULTIPLIER= option or the CVPBYTES= option, then SAS uses CVPMULTIPLIER=AUTO(0) to increase the lengths. AUTO(0) sets the value of the CVP engine based on the encoding of the SAS session and input data set.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error in the SAS log.

Interaction You cannot specify both the CVPMULTIPLIER= option and the CVPBYTES= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*

CVPVARCHAR=YES | NO

specifies whether to convert fixed-width character variables to variable-width characters during input file processing. The byte length of the new-width character variable is the maximum number of bytes per character from the SAS session encoding multiplied by the specified fixed-width character length.

Default No

Interaction If you specify CVPVARCHAR=YES, the CVPMULTIPLIER= and CVPBYTES= options are ignored.

Notes Trailing blanks are removed from string data that is under CHAR columns.

Fixed-width character variables with a format of TRANSCODE=NO are excluded during conversion.

Details

The character variable padding (CVP) engine expands character variable lengths so that character data truncation does not occur when a file requires transcoding. Character data truncation can occur when the number of bytes for a character in one encoding is different from the number of bytes for the same character in another encoding. An example is a single-byte character set (SBCS) such as Wlatin1

or a double-byte character set (DBCS) such as Shift-jis that is transcoded to a multi-byte character set (MBCS) such as UTF-8.

By explicitly specifying the CVP engine with the LIBNAME statement, the default character expansion is 1.5 times the character variable lengths. To specify a different expansion amount, use the CVPBYTES= or CVPMULTIPLIER= option.

Note: The expansion amount must be large enough to accommodate any expansion. Otherwise, truncation still occurs.

TIP The CVP engine might affect performance for processing that conditionally selects a subset of observations using a WHERE expression. Processing the file without using the CVP engine might be faster than processing the file using the CVP engine. For example, if you use the CVP engine with a data set that has indexes, the indexes are not used to optimize the WHERE expression.

Examples

Example 1

This LIBNAME statement explicitly assigns the CVP engine. Character variable lengths are increased using the default expansion, which multiples the lengths by 1.5. For example, a character variable with a length of 10 has a new length of 15. A character variable with a length of 100 has a new length of 150.

```
libname expand cvp 'SAS-library';
```

Example 2

In this example, the CVP engine is used to expand character variable lengths by adding two bytes to each length.

```
libname expand cvp 'SAS-library' cvpbytes=2;
```

See Also

- "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS National Language Support (NLS): Reference Guide*
- "Special-Purpose Engines" in *SAS Language Reference: Concepts*

Statements:

■ “LIBNAME Statement” on page 139

LIBNAME Statement: JMP Engine

Associates a libref with a JMP data table and enables you to read and write JMP data tables.

Valid in: Anywhere

Category: Data Access

See: Base SAS LIBNAME Statement

Syntax

LIBNAME *libref* JMP '*path*' <FMTLIB=*libref.format-catalog*>;

Arguments

libref
is a character constant, variable, or expression that specifies the libref that is assigned to a SAS library.

Range 1 to 8 bytes

path
is the physical name for the SAS library. The physical name is the name that is recognized by the operating environment. Enclose the physical name in single or double quotation marks.

FMTLIB=*libref.format-catalog*
specifies where the formats are stored when a JMP data table is read and where the formats come from when a JMP data table is created.

Requirement The library that is specified in the FMTLIB argument must be a SAS data set LIBNAME statement.

Example
```
libname inv jmp "." fmtlib=seform.formats;
libname seform '.';
   data work.mine;
   set inv.suri2011;
run;
```

Details

A JMP file is a file format that the JMP software program creates. JMP is an interactive statistics package that is available for Microsoft Windows and

Macintosh. For more information, see the JMP documentation that is packaged with your system.

A JMP file contains data that is organized in a tabular format of fields and records. Each field can contain one type of data, and each record can hold one data value for each field.

SAS supports access to JMP files. You can access JMP files by either of these two methods:

- the IMPORT and EXPORT procedures and the Import and Export Wizard without a license for SAS/ACCESS Interface to PC Files. SAS imports data from JMP files that are saved with version 7 or later formats, and it exports data to JMP files with version 7 or later formats. SAS no longer supports JMP files with versions 3 through 6 formats.

  For more information, see *SAS/ACCESS Interface to PC Files: Reference*.

- the LIBNAME statement for the JMP engine

Note: The JMP LIBNAME engine does not support extended attributes. If you want extended attributes, either use the IMPORT procedure or use the EXPORT procedure with **dbms=jmp**.

Examples

Example 1: Using the LIBNAME Statement to Read a JMP Data Table

This example reads and prints five observations from the **bank** JMP data table.

```
libname b jmp 'c:/temp/national';
proc contents data=b.bank(drop=edlevel id age);
run;
proc print data=b.bank(obs=5 drop=edlevel id age);
run;
```

Example 2: Reading and Sorting a JMP Data Table

This example reads a JMP data table, sorts it, and stores it in a SAS data set. The formats stored on the JMP data set are put in **a.formats**.

```
libname a 'c:/temp/field';
libname b jmp '.' fmtlib=a.formats;

proc sort data=b.cars out=a.sorted;
   by category_ic;
run;
```

LIBNAME Statement: JSON Engine

Provides read-only sequential access to JSON data.

Valid in: Anywhere

Category: Data Access

Notes: The JSON file is read only once, when the JSON engine LIBNAME statement is assigned. To read the JSON file again, you must reassign the JSON libref.

Map files generated by the JSON engine are created with the SAS session encoding by default. To create the map files with a different session encoding, see "Example 3: Re-Create an Existing JSON Map File in a Different Encoding" on page 184.To read JSON that is not in your session encoding, see "Example 4: Read JSON Created in a Different Encoding" on page 184.

Tip: If you want to create a JSON file from a SAS data set, SAS provides the JSON procedure. For more information, see *Base SAS Procedures Guide*.

Syntax

LIBNAME *libref* JSON < '*JSON-document-path*' > <*options*>;

Arguments

libref
specifies a logical name, or libref, to associate with the SAS library.

Range 1 to 8 bytes

JSON-document-path
is the physical location of the JSON document. Enclose the physical location in single or double quotation marks.

Note: If the physical location is not supplied, the JSON engine tries to access a fileref with the same name as the libref that is supplied. The fileref can be a disk file, a URL access method fileref, an FTP access method fileref, or other valid fileref.

LIBNAME Options

ALLDATA= "*name*"
renames the ALLDATA data set to the specified name. A data set named ALLDATA is created by default when the JSON engine runs. The ALLDATA data set contains all of the information in the JSON file. This option creates the data set with the specified name instead. The new name must meet the conventions for SAS names.

Note This option is available beginning with SAS 9.4M5 and SAS Viya 3.4.

AUTOMAP= REUSE | CREATE
specifies the action to take. AUTOMAP can have one of these values:

REUSE
uses the MAP fileref if the file exists. Otherwise, generates a JSON map and writes it to the MAP fileref.

CREATE
generates a JSON map and writes it to the MAP fileref.

Note: Using REPLACE or CREATE overwrites an existing file.

Alias REPLACE

Requirement If a JSON map file is specified but does not exist, the AUTOMAP option is required.

Interaction If a JSON map file is not specified, it is automatically created in memory. The AUTOMAP option is not required.

FILEREF=*fileref*
specifies a fileref that points to the location for the JSON input. The fileref cannot exceed eight bytes.

JSONCOMPRESS | JSONNOCOMPRESS
specifies whether to compress internal JSON information.

TIP Compression saves memory and disk space when caching information, at the expense of CPU time.

Default JSONNOCOMPRESS

MAP=*fileref* | '*map physical name*'
specifies a fileref that points to a physical location for the JSON map, or the quoted physical name of the JSON map file.

MEMLEAVE=*n* | *n*K | *n*M | *n*G | ALL
specifies the amount of memory to leave available when processing JSON. Once the memory limit is met, JSON information is cached to disk using an internally generated TEMP fileref.

n
amount of memory.

***n*K**
specifies the amount of memory in Kilobytes.

***n*M**
specifies the amount of memory in Megabytes.

*n*G
specifies the amount of memory in Gigabytes.

ALL
specifies that no memory is used to store JSON information. All JSON information is cached to disk.

Default 0.5G

NOALLDATA
Suppresses creation of the ALLDATA data set.

Note This option is available beginning with SAS Viya 3.4.

NO_REOPEN_MSG | NRM
Suppresses the message, `JSON data is only read once. To read the JSON again, reassign the JSON LIBNAME.`

Note The JSON file is read once, when the JSON engine LIBNAME is assigned. To read the JSON file again, you must deassign the LIBNAME libref and reassign it. For more information, see "Example 2: Use JSON from a URL Access Method Fileref" on page 183.

ORDINALCOUNT= ALL | NONE | *n*
specifies the maximum number of ordinal variables to generate for each data set. The maximum number of possible ordinal variables for a data set depends on the position of the data in the JSON, and can be less than the number that you specify. In that case, only the maximum number of possible ordinal variables is generated. ORDINALCOUNT can have one of these values:

ALL
creates all possible ordinal variables for the data set.

NONE
suppresses creation of ordinal variables for the data set.

n
is an integer that is greater than or equal to 0 (zero).

Default 2

RETAIN
retains values in the observation buffer between observations.

Details

The Basics

A JavaScript Object Notation (JSON) file is a file that contains a human-readable collection of data.

The JSON LIBNAME statement enables you to read a JSON file using the JSON engine.

The JSON engine uses a JSON map file to describe the data sets in the specified JSON file. The JSON engine can generate a JSON map file for you when you assign the LIBNAME statement, or you can supply your own JSON map file in the MAP= option.

The JSON automapper generates a data set for each object in the JSON. It also generates a data set named ALLDATA, which contains all JSON information in a single data set. After the JSON engine library is assigned, you can list the data sets in the library by using PROC DATASETS. You can get information about the data sets with PROC CONTENTS. You can view the data in each data set by using PROC PRINT.

If the data sets created by the automapper do not sufficiently describe the data, you can modify the generated map file. Before using the output data sets in SAS, you might want to merge the output data sets. You can also manipulate the data in the ALLDATA data set to create a new SAS data set.

JSON MAP Details

A JSON map is a file that describes the data sets in a JSON engine library. If specified, the JSON automapper can automatically generate a map of your JSON data.

A JSON map consists of a DATASETS array.

Each object in the DATASETS array describes a data set in the library.

Each data set object requires a DSNAME string and a TABLEPATH string and usually contains an optional VARIABLES array.

Each object in the VARIABLES array describes a variable in that data set.

Required items for each VARIABLES object are a NAME string, a TYPE string, and a PATH string.

For character TYPE variables, a LENGTH item is optional.

DSNAME

The DSNAME provides a name for the data set. The automapper generates a default name. When modifying your map file, you can provide any name that conforms to the SAS data set naming standard.

TABLEPATH

A TABLEPATH path tells the engine how to delimit data set observations, given this JSON:

```
{
  "data" : [
    { "a" : 1 , "b" : 2, "c" : "taxes" },
    { "a" : 2 , "b" : 4, "c" : "sport" },
    { "a" : 3 , "b" : 6, "c" : "vacation" }
   ]
  }
```

If you want an observation every time you encounter an object in the data array, set the TABLEPATH to "/root/data". The JSON mapper automatically generates this JSON map:

```
 {
```

```
"DATASETS": [
  {
    "DSNAME": "data",
    "TABLEPATH": "/root/data",
    "VARIABLES": [
      {
        "NAME": "ordinal_root",
        "TYPE": "ORDINAL",
        "PATH": "/root"
      },
      {
        "NAME": "ordinal_data",
        "TYPE": "ORDINAL",
        "PATH": "/root/data"
      },
      {
        "NAME": "a",
        "TYPE": "NUMERIC",
        "PATH": "/root/data/a"
      },
      {
        "NAME": "b",
        "TYPE": "NUMERIC",
        "PATH": "/root/data/b"
      },
      {
        "NAME": "c",
        "TYPE": "CHARACTER",
        "PATH": "/root/data/c",
        "CURRENT_LENGTH": 8
      }
    ]
  }
]
}
```

NAME

The NAME gives a name for the variable that conforms to the SAS naming standard.

TYPE

TYPE is ORDINAL, NUMERIC, or CHARACTER.

PATH

PATH is the path in the JSON map file for this variable.

LENGTH

The LENGTH setting is the optional length of this character variable. If no length is specified, the JSON LIBNAME engine sets this length to the maximum length of all values that it has seen for this variable.

Note When reusing maps with different JSON input, particularly JSON input with longer strings, the map LENGTH remains in effect and might result in truncation.

Tip When the JSON LIBNAME engine generates an automap, it generates a CURRENT_LENGTH: n value, set to the maximum length of the variable

in the current JSON data. This is for documentation only. When reading maps, the JSON LIBNAME engine ignores the CURRENT_LENGTH setting.

FORMAT

You can provide a FORMAT for a variable in a map. FORMAT is an array where the first element of the array is the format name, the second element is the width, and the third element is the decimal specification. The FORMAT variable is optional.

"FORMAT" : ["*FormatName*, "*width*", "*decimal-specification*"]

Note The JSON LIBNAME engine automapper does not currently generate FORMATs.

Example This map specifies a format BEST that contains a total of 12 digits with 3 decimal places.

```
{
   "NAME": "c",
   "TYPE": "NUMERIC",
   "PATH": "/root/nums/c",
   "FORMAT" : [ "BEST", 12, 3 ]
 }
```

INFORMAT

You can provide an INFORMAT for a variable in a map. INFORMAT is an array where the first element of the array is the informat name, the second element is the width, and the third element is the decimal specification. The INFORMAT variable is optional.

"INFORMAT" : ["*InformatName*, "*width*", "*decimal-specification*"]

Note The JSON LIBNAME engine automapper does not currently generate INFORMATs. This map specifies the informat IS8601DT that contains 19 digits and no decimal places.

```
{
    "NAME": "d",
    "TYPE": "NUMERIC",
    "PATH": "/root/nums/d",
    "INFORMAT" : [ "IS8601DT", 19, 0 ]
 }
```

Tip You can suppress "invalid data" messages printed in the log by adding a preceding "?" to the informat name.

```
"INFORMAT" : [ "?IS8601DT", 19, 0 ]
```

LABEL

You can provide a LABEL for a variable in a map. The LABEL variable is optional.

"LABEL" : ["*text*"]

Note The JSON LIBNAME engine automapper does not currently generate LABELs.

Example This map specifies a label for the variable d.

```
{
    "NAME": "d",
    "LABEL": "This is the d variable.",
    "TYPE": "NUMERIC",
    "PATH": "/root/nums/d"
 }
```

OPTIONS

You can turn RETAIN on or off for an individual variable. The OPTIONS variable is optional. For more information, see "RETAIN" on page 166.

"OPTIONS" : ["RETAIN" | "NORETAIN"]

Note Setting NORETAIN on ORDINAL variables has no effect and produces a NOTE during map validation. ORDINAL variables are always retained.

Tip Setting RETAIN on a variable in a map takes precedence over the RETAIN option in the LIBNAME statement. For example, with RETAIN set in the LIBNAME statement, and NORETAIN set on JSON variable Status, all variables are retained except for Status.

Example This example shows turning RETAIN on for the variable Status.

```
 {
    "NAME": "Status",
    "TYPE": "CHARACTER",
    "PATH": "/root/info/Status",
    "OPTIONS" : ["RETAIN"]
  }
```

Creating and Editing a JSON MAP

This example shows how to get a list of employee phone numbers, as well as type of phone, first name of the employee, and his age using a JSON file named example.json. This is the content of example.json:

```
[
  {
   "type": "Full",
   "info" : [
      { "name" : "Eric" , "age" : 21, "phone" : [
         { "type" : "cell", "number" : "540-555-2377" },
         { "type" : "home", "number" : "540-555-0120" }
         ]
      },
      { "name" : "John",  "age" : 22, "phone" : [
         { "type" : "cell", "number" : "919-555-6665" },
         { "type" : "home", "number" : "336-555-0140" }
         ]
      }
    ]
  },
  {
   "type": "Part",
   "info" : [
```

```
        { "name" : "Bjorn" , "age" : 27, "phone" : [
           { "type" : "cell",   "number" : "720-555-8377" },
           { "type" : "burner", "number" : "720-555-2877" },
           { "type" : "home",   "number" : "720-555-0194" }
           ]
        }
     ]

   }
]
```

The first step is to create a JSON map from the JSON file. This LIBNAME statement produces the JSON map file, user.map:

```
libname in json 'example.json' map='user.map' automap=create;
```

You can use PROC DATASETS to view the content of library IN. For these examples, we have changed the SAS output to list format to conserve space.

```
1   ods html close;
2   ods listing;
3   options nodate;

4   proc datasets lib=in; run; quit;
                                        Directory

                        Libref         IN
                        Engine         JSON
                        Access         READONLY
                        Physical Name  C:\Users\myname\example.json


                                                  Member
                                    #  Name       Type

                                    1  ALLDATA    DATA
                                    2  INFO       DATA
                                    3  INFO_PHONE DATA
                                    4  ROOT       DATA
```

Three data sets are produced by the JSON: INFO, INFO_PHONE, and ROOT.

You can use PROC PRINT to print the contents of the three data sets.

```
5         proc print data=in.root; run;

            ordinal_
     OBS      root      type

       1        1       Full
       2        2       Part
6         proc print data=in.info; run;

            ordinal_    ordinal_
     OBS      root        info      name     age

       1        1           1       Eric      21
       2        1           2       John      22
       3        2           3       Bjorn     27

7         proc print data=in.info_phone; run;

            ordinal_    ordinal_
     OBS      info        phone      type        number

       1        1           1       cell      540-555-2377
       2        1           2       home      540-555-0120
       3        2           3       cell      919-555-6665
       4        2           4       home      336-555-0140
       5        3           5       cell      720-555-8377
       6        3           6       burner    720-555-2877
       7        3           7       home      720-555-0194
```

Use a file editor to view the content of generated map file user.map.

```
{
  "DATASETS": [
    {
      "DSNAME": "root",
      "TABLEPATH": "/root",
      "VARIABLES": [
        {
          "NAME": "ordinal_root",
          "TYPE": "ORDINAL",
          "PATH": "/root"
        },
        {
          "NAME": "type",
          "TYPE": "CHARACTER",
          "PATH": "/root/type",
          "CURRENT_LENGTH": 4
        }
      ]
    },
    {
      "DSNAME": "info",
      "TABLEPATH": "/root/info",
      "VARIABLES": [
        {
          "NAME": "ordinal_root",
          "TYPE": "ORDINAL",
          "PATH": "/root"
        },
        {
          "NAME": "ordinal_info",
```

```
            "TYPE": "ORDINAL",
            "PATH": "/root/info"
          },
          {
            "NAME": "name",
            "TYPE": "CHARACTER",
            "PATH": "/root/info/name",
            "CURRENT_LENGTH": 5
          },
          {
            "NAME": "age",
            "TYPE": "NUMERIC",
            "PATH": "/root/info/age"
          }
        ]
      },
      {
        "DSNAME": "info_phone",
        "TABLEPATH": "/root/info/phone",
        "VARIABLES": [
          {
            "NAME": "ordinal_info",
            "TYPE": "ORDINAL",
            "PATH": "/root/info"
          },
          {
            "NAME": "ordinal_phone",
            "TYPE": "ORDINAL",
            "PATH": "/root/info/phone"
          },
          {
            "NAME": "type",
            "TYPE": "CHARACTER",
            "PATH": "/root/info/phone/type",
            "CURRENT_LENGTH": 6
          },
          {
            "NAME": "number",
            "TYPE": "CHARACTER",
            "PATH": "/root/info/phone/number",
            "CURRENT_LENGTH": 12
          }
        ]
      }
    ]
}
```

Using the file editor, copy the user.map file to create a new file: user.map.all.

Modify user.map.all as follows:

- Change the DSNAME of the first data set to ALL.
- Remove the ORDINAL variables from the ALL data set.
- Copy the variable descriptions of the variables that you want from the other data sets into the ALL data set. Ensure there is a comma after each variable object except the last one.

- Change the name of the type variable from the INFO_PHONE data set to PHONETYPE.
- Add a LABEL to the PHONETYPE variable.
- Set the TABLEPATH to /root/info/phone to get an observation after a phone object is encountered.
- Remove CURRENT_LENGTH from each variable.

Here is the modified map:

```
{
  "DATASETS": [
    {
      "DSNAME": "all",
      "TABLEPATH": "/root/info/phone",
      "VARIABLES": [
        {
          "NAME": "type",
          "TYPE": "CHARACTER",
          "PATH": "/root/type"
        },
        {
          "NAME": "name",
          "TYPE": "CHARACTER",
          "PATH": "/root/info/name"
        },
        {
          "NAME": "age",
          "TYPE": "NUMERIC",
          "PATH": "/root/info/age"
        },
        {
          "NAME": "phonetype",
          "TYPE": "CHARACTER",
          "LABEL": "This is the type of phone",
          "PATH": "/root/info/phone/type"
        },
        {
          "NAME": "number",
          "TYPE": "CHARACTER",
          "PATH": "/root/info/phone/number"
        }
      ]
    }
  ]
}
```

These results are displayed using PROC PRINT with the modified map.

```
8         filename allmap 'user.map.all';
9         libname in json 'example.json' map=allmap;
10         proc print data=in.all label; run;

                                  This is
                                  the type
     OBS    type    name     age   of phone       number

       1    Full    Eric      21    cell       540-555-2377
       2                       .    home       540-555-0120
       3            John      22    cell       919-555-6665
       4                       .    home       336-555-0140
       5    Part    Bjorn     27    cell       720-555-8377
       6                       .    burner     720-555-2877
       7                       .    home       720-555-0194
```

Notice that the observation buffer is cleared between Eric's cell phone and Eric's home phone. You can use the RETAIN option to keep the observation buffer from being cleared.

```
11         libname in json 'example.json' map=allmap RETAIN;
12         proc print data=in.all label; run;

                                  This is
                                  the type
     OBS    type    name     age   of phone       number

       1    Full    Eric      21    cell       540-555-2377
       2    Full    Eric      21    home       540-555-0120
       3    Full    John      22    cell       919-555-6665
       4    Full    John      22    home       336-555-0140
       5    Part    Bjorn     27    cell       720-555-8377
       6    Part    Bjorn     27    burner     720-555-2877
       7    Part    Bjorn     27    home       720-555-0194
```

Merging Data Sets

This example describes how to read JSON code into four SAS data sets and then use ordered values to merge the data sets into a single data set.

In this JSON code, there are two types (Work and Holding) and three statuses (complete, in the middle, and not started). There is also name and address information. The code exists in a file named example2.json.

```
[
  { "Type" : "Work",
    "info" : [{ "Status" : "complete",
                "name" : { "name" : "Eric", "Date" : "28sep15", "age":
21 },
                "add" : { "Address" : "55 Pelican Avenue", "zip" :
44442 }
              },
              { "Status" : "in the middle",
                "name" : { "name" : "John", "Date" : "02oct15",
"age": 22 },
                "add" : { "Address" : "268 Hydrangea" , "zip" :
40207 }
               }
              ]
   },
```

```
    { "Type" : "Holding",
      "info" : [{ "Status" : "not started",
                  "name" : { "name" : "Bjorn", "Date" : "01dec15",
"age": 27 },
                  "add" : { "Address" : "1217 Onslow", "zip" :"22801" }
                }
               ]
     }
]
```

Use this SAS code to assign the JSON LIBNAME engine to read the input JSON document. Use PROC DATASETS to show the name and member type of the four SAS data sets that were created.

```
filename in 'example2.json';
filename map 'example.map';
libname in json map=map automap=replace;

proc datasets library=in; run; quit;
```

```
1         options nocenter;
2         filename in 'example2.json';
3         filename map 'example.map';
4         libname in json map=map automap=replace;
NOTE: JSON data is only read once.  To read the JSON again, reassign the JSON
LIBNAME.
NOTE: Map file /r/example.com/mydir/work/json/example.map was replaced.
NOTE: Libref IN was successfully assigned as follows:
      Engine:        JSON
      Physical Name: /r/example.com/mydir/work/json/example2.json

5         proc datasets library=in;
                                          Directory

Libref         IN
Engine         JSON
Access         READONLY
Physical Name  /r/example.com/mydir/work/json/example.json


               Member
#  Name        Type

1  INFO        DATA
2  INFO_ADD    DATA
3  INFO_NAME   DATA
4  ROOT        DATA
6         run;

7         quit;
```

You can use PROC PRINT to print the contents of the data sets.

```
proc print data=in.root; run;

      ordinal_
Obs     root      Type

 1        1       Work
 2        2       Holding

proc print data=in.info_name; run;

      ordinal_    ordinal_
Obs     info        name       name      Date      age

 1        1           1       Eric      28sep15     21
 2        2           2       John      02oct15     22
 3        3           3       Bjorn     01dec15     27

proc print data=in.info; run;

      ordinal_    ordinal_
Obs     root        info      Status

 1        1           1       complete
 2        1           2       in the middle
 3        2           3       not started

proc print data=in.info_add;run;

      ordinal_    ordinal_
Obs     info         add          Address          zip

 1        1           1       55 Pelican Avenue   44442
 2        2           2       268 Hydrangea       40207
 3        3           3       1217 Onslow         22801
```

The four data sets contain ordinal variables, which are key variables that provide a relationship between two data sets.

The second data set, INFO_NAME, has an observation with John's name and it has an ORDINAL_INFO value of 2. In the fourth data set, INFO_ADD, the address associated with the ORDINAL_INFO value of 2 is 268 Hydrangea. This is the address associated with John.

To merge the four data sets based on ordinal values, merge the ROOT and INFO data sets by ORDINAL_ROOT.

```
data a;
   merge in.root in.info;
   by ORDINAL_root;
  run;

NOTE: There were 2 observations read from the data set IN.ROOT.
NOTE: There were 3 observations read from the data set IN.INFO.
NOTE: The data set WORK.A has 3 observations and 4 variables.

  proc print data=a; run;

      ordinal_              ordinal_
Obs     root      Type        info      Status

 1        1       Work          1       complete
 2        1       Work          2       in the middle
 3        2       Holding       3       not started
```

Next, to complete the creation of a single data set that contains all of the JSON information, merge Name and Address and drop the ORDINAL variables.

```
data b;
  merge a in.info_name in.info_add;
   by ORDINAL_info;
  drop ORDINAL_info ORDINAL_name ORDINAL_add ORDINAL_root;
run;

NOTE: There were 3 observations read from the data set WORK.A.
NOTE: There were 3 observations read from the data set IN.INFO_NAME.
NOTE: There were 3 observations read from the data set IN.INFO_ADD.
NOTE: The data set WORK.B has 3 observations and 7 variables.

proc print data=b; run;

Obs Type    Status        name  Date    age Address           zip

 1  Work    complete      Eric  28sep15  21 55 Pelican Avenue 44442
 2  Work    in the middle John  02oct15  22 268 Hydrangea     40207
 3  Holding not started   Bjorn 01dec15  27 1217 Onslow       22801
```

The ALLDATA Data Set

The ALLDATA data set gives you access to all of the JSON data in one data set. For example:

```
proc print data=in.ALLDATA(obs=24); run;
```

OBS	P	P1	P2	P3	P4	V	Value
1	1	stores				0	
2	2	stores	Name			1	Bob's Mart
3	2	stores	opened			1	06-01-2001
4	2	stores	sales			0	
5	3	stores	sales	Hot_Dogs		0	
6	4	stores	sales	Hot_Dogs	count	1	39
7	4	stores	sales	Hot_Dogs	price	1	1.09
8	3	stores	sales	Salami		0	
9	4	stores	sales	Salami	count	1	20
10	4	stores	sales	Salami	price	1	5.99
11	3	stores	sales	Canteloupes		0	
12	4	stores	sales	Canteloupes	count	1	26
13	4	stores	sales	Canteloupes	price	1	1.39
14	3	stores	sales	Mustard		0	
15	4	stores	sales	Mustard	count	1	6
16	4	stores	sales	Mustard	price	1	2.19
17	2	stores	Code			1	12BMx2
18	1	stores				0	
19	2	stores	Name			1	Grab 'n' Git
20	2	stores	opened			1	06-03-2012
21	2	stores	sales			0	
22	3	stores	sales	Hot_Dogs		0	
23	4	stores	sales	Hot_Dogs	count	1	18
24	4	stores	sales	Hot_Dogs	price	1	1.19

The ALLDATA data set example contains these variables:

- P is a NUMERIC data type that shows how many of the P1-P4 variables contain information for this observation
- P1–P4 are CHARACTER data types
- V is a NUMERIC data type showing whether a Value is available.
- Value is a CHARACTER data type and is the JSON value.

In observation 1, P=1 indicates that P1 contains information and the P2–P4 variables are blank. Also, in observation 1, V=0 indicates that Value is blank. In observation 6, P=4 indicates that the P1–P4 variables contain information and V=1 indicates that Value contains a value.

Observations where V=0 indicate the beginning of a new JSON object. For example, in observations 1 and 18, where P=1 and V=1 indicate that a new Stores object begins. Observation 5 indicates that a new stores/sales/Hot_Dogs object begins. The V=0 observations are helpful for keeping track of the JSON objects.

The JSON LIBNAME engine also creates macro symbols describing lengths of the ALLDATA data set.

Here are the macro variables and values for the JSON example. The JSON LIBNAME is **IN**.

```
IN_JADPNUM=4
IN_JADVLEN=20
IN_JADP1LEN=6
IN_JADP2LEN=6
IN_JADP3LEN=11
IN_JADP4LEN=5
```

Using the ALLDATA Data Set

Create a data set where each observation has these variables:

- item
- item price and the count
- store name and code

Note that when you look at the ALLDATA data set, you see the Store Name, and then observations that contain the item name, the count and price. You create two data sets from the ALLDATA data set. One data set named StoreInfo contains the store Name and Code. The other data set named ItemInfo contains the Item, Count, and Price. Each time you find a Price, an observation is written to the ItemInfo data set. Each time you find a Code, an observation is written to the StoreInfo data set. In order to merge the two data sets, each data set contains a key that indicates which store the observation data came from. Here is the code for the example:

```
data
       storeinfo (keep = Key StoreName Code)
       iteminfo  (keep = Key Item Price Count)
             ;
         /*---------------------------------------------------------------------*/
         /* bring in ALLDATA                                                    */
         /*---------------------------------------------------------------------*/
```

```
        set in.ALLDATA;

        /*-------------------------------------------------------------------*/
        /* Since StoreName and Code are found in the Value variable of ALLDATA   */
        /* the length of StoreName and COde should be &IN_JADVLEN  Similarly,   */
        /* Item comes from P3, so its length is &IN_JADP3LEN.                    */
        /*-------------------------------------------------------------------*/
        length StoreName $ &IN_JADVLEN  Code $ &IN_JADVLEN Item $ &IN_JADP3LEN ;

        /*-------------------------------------------------------------------*/
        /* Both StoreName and Count are picked up in observations before the     */
        /* observations which tell us to write and output observation.  So we    */
        /* need to retain StoreName and Count                                    */
        /*-------------------------------------------------------------------*/
        retain StoreName Count;

        /*-------------------------------------------------------------------*/
        /* We want a Key which we'll increment each time we see a store object   */
        /* (v=0 p=1 and P1="stores"                                              */
        /*-------------------------------------------------------------------*/
        retain Key 0;

        /*-------------------------------------------------------------------------*/
        /*  increment our key each time we see a new store object                  */
        /*                                                                         */
        /*      OBS    P      P1        P2      P3              P4      V    Value */
        /*        1    1    stores                                    0          */
        /*-------------------------------------------------------------------------*/
        if v=0 and p=1 and p1="stores" then key+1;

        /*-------------------------------------------------------------------------------
*/
        /*  get StoreName, as in observation 2
*/
        /*
*/
        /*      OBS    P      P1        P2      P3              P4      V    Value
*/
        /*        2    2    stores    Name                              1    Bob's Mart
*/
        /*-------------------------------------------------------------------------------
*/
        if P=2 and p2 = "Name" then StoreName = value;

        /*-------------------------------------------------------------------------*/
        /* get Code and write a storeinfo observation                              */
        /*     OBS    P      P1        P2      P3              P4      V    Value  */
        /*      17    2    stores    Code                              1    12BMx2 */
        /*-------------------------------------------------------------------------*/
        if p=2 and p2 = "Code" then do;
           Code = Value;
           output storeinfo;
        end;

        /
*-------------------------------------------------------------------------------*/
```

```
         /* get Count,  example observations
below                                        */
         /
*                                                                                  */
         /*     OBS    P      P1        P2      P3              P4      V
Value        */
         /*       6    4    stores    sales     Hot_Dogs        count    1
39           */
         /*       9    4    stores    sales     Salami          count    1
20           */
         /*      12    4    stores    sales     Canteloupes     count    1
26           */
         /
*-----------------------------------------------------------------------------*/
         if P=4 and p4 = "count" then Count = input(value, 5.);

        /
*-----------------------------------------------------------------------------*/
        /*  get Price, and from that same observation, p3 is the Item.  And each
time     */
        /*  we see price, we want to output an
observation.                               */
        /
*                                                                                  */
        /*      OBS    P      P1        P2      P3              P4      V
Value        */
        /*        7    4    stores    sales     Hot_Dogs        price    1
1.09         */
        /*       10    4    stores    sales     Salami          price    1
5.99         */
        /*       13    4    stores    sales     Canteloupes     price    1
1.39         */
        /*       16    4    stores    sales     Mustard         price    1
2.19         */
        /
*-----------------------------------------------------------------------------*/
         if P=4 and p4 = "price" then do;
           Item = p3;
           Price = input(value, 5.);
           output iteminfo;
         end;


        run;
```

Here are the two data sets:

```
        proc print data=storeinfo; run;


OBS    StoreName                Code     Key

 1     Bob's Mart              12BMx2     1
 2     Grab 'n' Git            10GNx9     2
 3     Larry's Quick Shoppe    17LQx2     3

        proc print data=iteminfo; run;


OBS    Item           Count    Key    Price

  1    Hot_Dogs          39      1     1.09
  2    Salami            20      1     5.99
  3    Canteloupes       26      1     1.39
  4    Mustard            6      1     2.19
  5    Hot_Dogs          18      2     1.19
  6    Salami             3      2     7.99
  7    Mustard            6      2     2.19
  8    Beer              20      2     8.99
  9    Hot_Dogs          39      3     1.09
 10    Salami            20      3     5.99
 11    Mustard            6      3     2.19
 12    Beer               7      3     8.99
 13    Wine              15      3    12.99
```

This code merges the two data sets:

```
data a;
  merge storeinfo iteminfo;
  by key;
run;
```

This code is the merged data sets:

```
       proc print data=a;
        var Storename  Code Item Count Price;
       run;


OBS    StoreName                Code     Item           Count    Price

  1    Bob's Mart              12BMx2    Hot_Dogs          39     1.09
  2    Bob's Mart              12BMx2    Salami            20     5.99
  3    Bob's Mart              12BMx2    Canteloupes       26     1.39
  4    Bob's Mart              12BMx2    Mustard            6     2.19
  5    Grab 'n' Git            10GNx9    Hot_Dogs          18     1.19
  6    Grab 'n' Git            10GNx9    Salami             3     7.99
  7    Grab 'n' Git            10GNx9    Mustard            6     2.19
  8    Grab 'n' Git            10GNx9    Beer              20     8.99
  9    Larry's Quick Shoppe    17LQx2    Hot_Dogs          39     1.09
 10    Larry's Quick Shoppe    17LQx2    Salami            20     5.99
 11    Larry's Quick Shoppe    17LQx2    Mustard            6     2.19
 12    Larry's Quick Shoppe    17LQx2    Beer               7     8.99
 13    Larry's Quick Shoppe    17LQx2    Wine              15    12.99
```

Using the JSON Pretty Print DATA Step Function

The JSON Pretty Print DATA step function creates a readable copy of input JSON. Here is the syntax:

```
data _null_;
  rc = jsonpp("input file","output file");
run;
```

input file
The input JSON. This can be a physical name or a fileref.

output file
The output JSON. This can be a physical name or a fileref. If the output file is 'LOG', then the output is written to the SAS log.

This example reads the disk file test.json and creates a disk file test.json.pp

```
data _null_;
  rc = jsonpp('test.json', 'test.json.pp');
run;

/* read json from a url fileref and pretty print to disk fileref */

filename in url "http://example.com/~username/snip.json";
filename out 'pp.txt';
data _null_;
 rc = jsonpp('in','out');
run;
```

Examples

Example 1: Use the LIBNAME Statement to Create or Reuse a JSON Map

This example reads JSON data from a file called my.json and creates a JSON map in the file my.map. The engine creates a map if it does not exist. Otherwise, it uses the existing map.

```
filename in 'my.json';
filename map 'my.map';
libname in json map=map automap=reuse;
```

This example shows how to access a JSON document using the physical location of the document:

```
filename map 'my.map';
libname in json 'my.json' map=map automap=reuse;
```

Example 2: Use JSON from a URL Access Method Fileref

This example shows how to access JSON from a URL fileref.

```
filename in url "http://example.com/~username/snip.json" debug;
filename map 'snap.map';
libname in json map=map automap=replace;
proc contents data=in._all_; run;
```

When you want to call a RESTful API a second time, reassign the LIBNAME libref.

```
libname in json ... ;
```

Example 3: Re-Create an Existing JSON Map File in a Different Encoding

This example shows the settings necessary to use an existing map file that was created in the local SAS session encoding and create a map file in a different encoding. In this example, the new encoding is UTF-8.

This map is created with the session encoding:

```
filename m1 "mapexample.map";
libname in json map=m1 automap=create;
```

This map is created with an encoding value of UTF-8:

```
filename m2 "mapexample.map.utf8"  encoding=utf8;
libname in json map=m2 automap=create;
```

Example 4: Read JSON Created in a Different Encoding

This example shows the settings necessary to read JSON created with a SAS session encoding that is different from your SAS session encoding. The engine reads the file myjson as UTF-8, regardless of the session encoding. To read JSON created with a UTF encoding other than UTF-8, such as UTF-16LE, you must specify the appropriate UTF encoding.

```
filename in "myjson.utf8" encoding=utf8;
libname in json;
```

LIBNAME Statement: WebDAV Server Access

Associates a libref with a SAS library and enables access to a WebDAV (Web-based Distributed Authoring And Versioning) server.

Valid in: Anywhere

Category: Data Access

Restriction: Access to WebDAV servers is not supported on OpenVMS or z/OS.

Syntax

LIBNAME *libref* <*engine*> '*SAS-library*' < *options* > WEBDAV USER="*user-ID*" PASSWORD="*user-password*" *WEBDAV options*;

LIBNAME *libref* CLEAR | _ALL_ CLEAR ;

LIBNAME *libref* LIST | _ALL_ LIST ;

Arguments

libref
specifies a shortcut name for the aggregate storage location where your SAS files are stored.

Range 1 to 8 bytes

Tip The association between a libref and a SAS library lasts only for the duration of the SAS session or until you change it or discontinue it with another LIBNAME statement.

engine
specifies the name of a valid SAS engine.

Restriction REMOTE engines are not supported with the WebDAV options.

See For a list of valid engines, see the SAS documentation for your operating environment.

'SAS-library'
specifies the URL location (path) on a WebDAV server. The URL specifies either HTTP or HTTPS communication protocols.

Restriction Only one data library is supported when using the WebDAV extension to the LIBNAME statement.

Requirement When using the HTTPS communication protocol, you must use the Transport Layer Security (TLS) protocol that provides secure network communications. For more information, see *Encryption in SAS*.

CLEAR
disassociates one or more currently assigned librefs. When a libref using a WebDAV server is cleared, the cached files stored locally are deleted also.

Tip Specify *libref* to disassociate a single libref. Specify _ALL_ to disassociate all currently assigned librefs.

LIST
writes the attributes of one or more SAS libraries to the SAS log.

Tip Specify *libref* to list the attributes of a single SAS library. Specify _ALL_ to list the attributes of all SAS libraries that have librefs in your current session.

ALL
specifies that the CLEAR or LIST argument applies to all currently assigned librefs.

LIBNAME Options

For valid LIBNAME statement options, see "LIBNAME Statement" on page 139.

WebDAV Specific Options

AUTHDOMAIN="*auth-domain*"
specifies the name of an authentication domain metadata object in order to connect to the WebDAV server. The authentication domain references credentials (user ID and password) without your having to explicitly specify the credentials. The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

An 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 WebDAV server and is resolved by the BASE engine calling the SAS Metadata Server and returning the authentication credentials.

Requirement The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

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

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

LOCALCACHE="*directory name*"
specifies a directory where a temporary subdirectory is created to hold local copies of the server files. Each libref has its own unique subdirectory. If a directory is not specified, then the subdirectories are created in the SAS WORK directory. SAS deletes the temporary files when the SAS program completes.

Default SAS WORK directory

LOCKDURATION=*n*
specifies the number of minutes that the files written through the WebDAV libref are locked. SAS unlocks the files when the SAS program successfully completes. If the SAS program fails, then the locks expire after the time allotted.

Default 30

PASSWORD="*user-password*"
specifies a password for the user to access the WebDAV server. The password is case sensitive and it must be enclosed in single or double quotation marks.

Alias PWD=, PW=, PASS=

Tip You can specify the PROMPT option instead of the PASSWORD= option.

PROMPT
specifies to prompt for the user login password, if necessary.

Interaction If PROMPT is specified without USER=, then the user is prompted for an ID, as well as a password.

Tip If you specify the PROMPT option, you do not need to specify the PASSWORD= option.

PROXY=*url*
specifies the Uniform Resource Locator (URL) for the proxy server in one of these forms:

- "http://*hostname*"
- "http://*hostname:port*"

USER="*user-ID*"
specifies the user name for access to the WebDAV server. The user ID is case sensitive and it must be enclosed in single or double quotation marks.

Alias UID

Tip If PROMPT is specified, but USER= is not, then the user is prompted for an ID as well as a password.

WEBDAV
specifies that the libref access a WebDAV server.

Details

Data Set Options That Function Differently with a WebDAV Server

This table lists the data set options that have different functionality when using a WebDAV server. All other data set options function as described in the *SAS Data Set Options: Reference*.

Table 2.3 *Data Set Option Functionality with a WebDAV Server*

Data Set Option	WebDAV Storage Functionality
CNTLLEV=	*LIB* locks all data sets in the library before writing the data into the local cache. All members are unlocked after the DATA step has completed and the data set has been written back to the WebDAV server. *MEM* locks the member before writing the data into the local cache. Member is unlocked after the DATA step has completed and the data has been written back to the WebDAV server. *REC* is not supported. WebDAV allows updates to the entire data set only.

Data Set Option	WebDAV Storage Functionality
FILECLOSE	The VxTAPE engine is not supported. Therefore, this option is ignored.
GENMAX=	This functionality is not supported because the maximum number of revisions to keep cannot be specified in the WebDAV server.
GENNUM=	This functionality is not supported in WebDAV.
IDXNAME=	Users can specify an index to use if one exists.
INDEX=	Indexes can be created in the local cache and saved on the WebDAV server.
TOBSNO=	Remote engines are not supported. Therefore, this option is ignored.

WebDAV File Processing

When accessing a WebDAV server, the file is pulled from the WebDAV server to your local disk storage for processing. When you complete the updating, the file is pushed back to the WebDAV server for storage. The file is removed from the local disk storage when it is pushed back.

Multiple Librefs to a WebDAV Library

When you assign a libref to a file on a WebDAV server, the path (URL location), user ID, and password are associated with that libref. After the first libref has been assigned, the user ID and password is validated on subsequent attempts to assign another libref to the same library.

Note: Lock errors that you typically would not see might occur if either a different user ID or the password, or both, are used in the subsequent attempt to assign a libref to the same library.

Locked Files on a WebDAV Server

In local libraries, SAS locks a file when you open it to prevent other users from altering the file while it is being read. WebDAV locks require Write access to a library, and there is no concept of a read lock. In addition, WebDAV servers can go down, come back up, or go offline at any time. Consequently, SAS honors a lock request on a file on a WebDAV server only if the file is already locked by another user.

Example: Associating a Libref with a WebDAV Directory

This example associates the libref **davdata** with the WebDAV directory **/users/mydir/datadir** on the WebDAV server **www.webserver.com**:

```
libname davdata v9 "https://www.webserver.com/users/mydir/datadir"
    webdav user="mydir" pw="12345";
```

See Also

Statements:

- "FILENAME Statement: WebDAV Access Method" on page 114
- "LIBNAME Statement" on page 139

%LIST Statement

Displays lines that are entered in the current session.

Valid in:	Anywhere
Category:	Program Control
Requirement:	The MACRO macro system option is required. This option is enabled by default. The %LIST statement is not valid when the NOMACRO macro system option is specified.

Syntax

%LIST<*n* <:*m* | – *m*> >;

Without Arguments

In interactive line mode processing, if you use the %LIST statement without arguments, it displays all previously entered program lines.

Arguments

n
displays line *n*.

n–m
displays lines *n* through *m*.

Alias *n:m*

Details

Where and When to Use

The %LIST statement can be used anywhere in a SAS job except between a DATALINES or DATALINES4 statement and the matching semicolon (;) or semicolons (`;;;;`). This statement is useful mainly in interactive line mode sessions to display SAS program code on the monitor. It is also useful to determine lines to include when you use the %INCLUDE statement.

Interactions

CAUTION
In all modes of execution, the SPOOL system option controls whether SAS statements are saved. When the SPOOL system option is in effect in interactive line mode, all SAS statements and data lines are saved automatically when they are submitted. You can display them by using the %LIST statement. When NOSPOOL is in effect, %LIST cannot display previous lines.

CAUTION
The MACRO macro system option is required. The MACRO system option is enabled by default and is required for the %LIST statement to work. If you specify the NOMACRO system option and then specify the %LIST statement, you get an error in the SAS log.

Example: Displaying Lines That Are Entered in the Current Session

This %LIST statement displays lines 10 through 20:

```
%list 10-20;
```

See Also

Statements:

- "%INCLUDE Statement" on page 132

System Options:

- "SPOOL System Option" in *SAS System Options: Reference*

LOCK Statement

Acquires, lists, or releases an exclusive lock on an existing SAS file.

Valid in: Anywhere

Category: Program Control

Restrictions: The SAS file must exist before you can request a lock.

You cannot lock a SAS file that another SAS session is currently accessing (either from an exclusive lock or because the file is open).

The LOCK statement syntax is the same whether you issue the statement in a single-user environment or in a client/server environment. However, some LOCK statement functionality applies only to a client/server environment.

Syntax

LOCK *libref*<.*member-name*<.*member-type* | .*entry-name*.*entry-type*>>
<LIST | QUERY | SHOW | CLEAR | NOMSG>;

Arguments

libref
is a name that is associated with a SAS library. The libref (library reference) must be a valid SAS name. If the libref is Sasuser or Work, you must specify it.

Range 1 to 8 bytes

Tip Typically, in a single-user environment, you would not issue the LOCK statement to exclusively lock a library. To lock a library that is accessed via a multiuser SAS/SHARE server, see the LOCK statement in *SAS/SHARE User's Guide*.

member-name
is a valid SAS name that specifies a member of the SAS library that is associated with the libref.

member-type
is the type of SAS file to be locked. For example, valid values are DATA, VIEW, CATALOG, MDDB, and so on. The default is DATA.

entry-name
is the name of the catalog entry to be locked.

Tip In a single-user environment, if you issue the LOCK statement to lock an individual catalog entry, the entire catalog is locked. Typically, you would not issue the LOCK statement to exclusively lock a catalog entry. To lock a

catalog entry in a library that is accessed via a multiuser SAS/SHARE server, see the LOCK statement in *SAS/SHARE User's Guide*.

entry-type
is the type of catalog entry to be locked.

Tip In a single-user environment, if you issue the LOCK statement to lock an individual catalog entry, the entire catalog is locked. Typically, you would not issue the LOCK statement to exclusively lock a catalog entry. To lock a catalog entry in a library that is accessed via a multiuser SAS/SHARE server, see the LOCK statement in *SAS/SHARE User's Guide*.

LIST | QUERY | SHOW
writes to the SAS log whether you have an exclusive lock on the specified SAS file.

Tip This option provides more information in a client/server environment. To use this option in a client/server environment, see the LOCK statement in *SAS/SHARE User's Guide*.

CLEAR
releases a lock on the specified SAS file that was acquired using the LOCK statement in your SAS session.

NOMSG
specifies that warnings and error messages are not written to the SAS log. NOMSG does not suppress notes that tell you that a lock is successful or that a lock is cleared.

Interactions To suppress warnings and error messages, you must specify NOMSG for each execution of the LOCK statement.

The SAS macro variable SYSLCKRC return code is not affected if NOMSG is specified.

Tip NOMSG is useful if you want a LOCK statement resubmitted in a code loop until a lock is available, but you do want error messages displayed in the SAS log each time that an exclusive lock is not available.

Details

General Information

The LOCK statement enables you to acquire, list, or release an exclusive lock on an existing SAS file. With an exclusive lock, no other operation in the current SAS session can read, write, or lock the file until the lock is released. In addition, with an exclusive lock, when the file is open, the lock ensures that another SAS session cannot access the file.

The primary use of the LOCK statement is to retain exclusive control of a SAS file across SAS statement boundaries. There are times when it is desirable to perform

several operations on a file, one right after the other, without losing exclusive control of the file. However, when a DATA or PROC step finishes executing and control flows from it to the next operation, the file is closed and becomes available for processing by another SAS session that has access to the same data storage location. To ensure that an exclusive lock is guaranteed across multiple SAS sessions, you must use SAS/SHARE.

To release an exclusive lock, use the CLEAR option. In addition, an exclusive lock on a data set is released when you use the DATASETS procedure DELETE statement to delete the data set.

Return Codes for the LOCK Statement

The SAS macro variable SYSLCKRC contains the return code from the LOCK statement. These actions result in a nonzero value in SYSLCKRC:

- You try to lock a file but cannot obtain the lock (for example, the file was in use or is locked by another SAS session).
- You use a LOCK statement with the LIST option to list a lock.
- You use a LOCK statement with the CLEAR option to release a lock that you do not have.

For more information about the SYSLCKRC SAS macro variable, see *SAS Macro Language: Reference*.

Comparisons

- With SAS/SHARE software, you can also use the LOCK statement. Some LOCK statement functionality applies only to a client/server environment.
- The CNTLLEV= data set option specifies the level at which shared Update access to a SAS data set is denied.

Example: Locking a SAS File

This SAS program illustrates the process of locking a SAS data set. Including the LOCK statement provides protection for the multistep program by acquiring exclusive access to the file.

```
libname mydata 'SAS-library';
lock mydata.census;  1
data mydata.census;  2
   modify mydata.census;
   /* statements to remove obsolete observations */
run;  3
proc sort force data=mydata.census;  4
   by CrimeRate;
run;
proc datasets library=mydata;  5
   modify census;
```

```
   index create CrimeRate;
quit;
lock mydata.census clear;  6
```

1 Acquires exclusive access to the SAS data set MyData.Census.

2 Opens MyData.Census to remove observations. During the update, no other operation in the current SAS session and no other SAS session can access the file.

3 At the end of the DATA step, the file is closed. No other operation in the current SAS session can access the file. However, until the file is reopened, it can be accessed by another SAS session.

4 Opens MyData.Census to sort the file. During the sort, no other operation in the current SAS session and no other SAS session can access the file. At the end of the procedure, the file is closed, which means that no other operation in the current SAS session can access the file, but the file can be accessed by another SAS session.

5 Opens MyData.Census to rebuild the file's index. No other operation in the current SAS session and no other SAS session can access the file. At the end of the procedure, the file is closed.

6 Releases the exclusive lock on MyData.Census.

See Also

- For information about locking a data object in a library that is accessed via a multiuser SAS/SHARE server, see "LOCK Statement" in *SAS/SHARE User's Guide*.

Data Set Options:

- "CNTLLEV= Data Set Option" in *SAS Data Set Options: Reference*

MISSING Statement

Assigns characters in your input data to represent special missing values for numeric data.

Valid in: Anywhere

Category: Information

Syntax

MISSING *character(s)*;

Arguments

character
is the value in your input data that represents a special missing value.

Range Special missing values can be any of the 26 letters of the alphabet (uppercase or lowercase) or the underscore (_).

Tip You can specify more than one character.

Details

The MISSING statement usually appears within a DATA step, but it is global in scope.

Comparisons

The MISSING= system option enables you to specify a character to be printed when numeric variables contain ordinary missing values (.). If your data contains characters that represent special missing values, such as `a` or `z`, do not use the MISSING= option to define them; simply define these values in a MISSING statement.

Example: Identifying Certain Types of Missing Data

With survey data, you might want to identify certain types of missing data. For example, in the data, an `A` can mean that the respondent is not at home at the time of the survey; an `R` can mean that the respondent refused to answer. Use the MISSING statement to identify to SAS that the values `A` and `R` in the input data lines are to be considered special missing values rather than invalid numeric data values:

```
data survey;
   missing a r;
   input id answer;
   datalines;
001 2
002 R
003 1
004 A
005 2
;
```

The resulting data set SURVEY contains exactly the values that are coded in the input data.

See Also

Statements:

- "UPDATE Statement" in *SAS DATA Step Statements: Reference*

System Options:

- "MISSING= System Option" in *SAS System Options: Reference*

Null Statement

Signals the end of data lines or acts as a placeholder.

Valid in: Anywhere

Category: Action

Type: Executable

Syntax

;

or

;;;;

Without Arguments

The Null statement signals the end of the data lines that occur in your program.

Details

The primary use of the Null statement is to signal the end of data lines that follow a DATALINES or CARDS statement. In this case, the Null statement functions as a step boundary. When your data lines contain semicolons, use the DATALINES4 or CARDS4 statement and a Null statement of four semicolons.

Although the Null statement performs no action, it is an executable statement. Therefore, a label can precede the Null statement, or you can use it in conditional processing.

Examples

Example 1: Marking the End of Data Lines

The Null statement in this program marks the end of data lines and functions as a step boundary.

```
data test;
   input score1 score2 score3;
   datalines;
55 135 177
44 132 169
;
```

Example 2: Marking the End of Data Lines That Contain Semicolons

The input data records in this example contain semicolons. Use the Null statement following the DATALINES4 statement to signal the end of the data lines.

```
data test2;
   input code1 $ code2 $ code3 $;
   datalines4;
55;39;1  135;32;4  177;27;3
78;29;1  149;22;4  179;37;3
;;;;
```

Example 3: Using a Statement Label with Null

The Null statement is useful while you are developing a program. For example, use it after a statement label to test your program before you code the statements that follow the label.

```
data _null_;
   set dsn;
   file print header=header;
   put 'report text';
     ...more statements...
   return;
   header:;
run;
```

See Also

Statements:

- "DATALINES Statement" in *SAS DATA Step Statements: Reference*
- "DATALINES4 Statement" in *SAS DATA Step Statements: Reference*
- "GO TO Statement" in *SAS DATA Step Statements: Reference*
- "LABEL Statement" in *SAS DATA Step Statements: Reference*

OPTIONS Statement

Specifies or changes the value of one or more SAS system options.

Valid in: Anywhere

Category: Program Control

See: OPTIONS Statement under z/OS

Syntax

OPTIONS *options*;

Arguments

option
specifies one or more SAS system options to be changed.

See *SAS System Options: Reference*

System Options Syntax in SAS Help Center

Details

The change that is made by the OPTIONS statement remains in effect for the rest of the job, session, SAS process, or until you issue another OPTIONS statement to change the options again. You can specify SAS system options through the OPTIONS statement, through the OPTIONS window, at SAS invocation, and at the initiation of a SAS process.

If you attempt to set an option that is restricted by your site administrator, SAS issues a note that the option is restricted and cannot be changed. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Note: If you want a particular group of options to be in effect for all your SAS jobs or sessions, store an OPTIONS statement in an autoexec file or list the system options in a configuration file or custom_option_set.

Note: For a system option with a null value, the GETOPTION function returns a value of ' ' (single quotation marks with a blank space between them), for example, `EMAILID=' '`. This GETOPTION value can then be used in the OPTIONS statement.

An OPTIONS statement can appear at any place in a SAS program, except within data lines.

Operating Environment Information: The system options that are available depend on your operating environment. Also, the syntax that is used to specify a system option in the OPTIONS statement might be different from the syntax that is used at SAS invocation. For more information, see the SAS documentation for your operating environment.

Comparisons

The OPTIONS statement requires you to enter the complete statement including system option name and value, if necessary. The SAS OPTIONS window displays the options' names and settings in columns. To change a setting, type over the value that is displayed and press Enter or Return.

Example: Changing the Value of a System Option

This example suppresses the date that is normally written to SAS LISTING output and sets a line size of 72:

```
options nodate linesize=72;
```

See Also

"Definition of System Options" in *SAS System Options: Reference*

PAGE Statement

Skips to a new page in the SAS log.

Valid in: Anywhere

Category: Log Control

Syntax

PAGE;

Without Arguments

The PAGE statement skips to a new page in the SAS log.

Details

You can use the PAGE statement when you run SAS in a windowing environment, batch, or noninteractive mode. The PAGE statement itself does not appear in the log. When you run SAS in interactive line mode, PAGE might print blank lines to the display monitor (or to the alternate log file).

See Also

Statements:

- "LIST Statement" in *SAS DATA Step Statements: Reference*

System Options:

- "LINESIZE= System Option" in *SAS System Options: Reference*
- "PAGESIZE= System Option" in *SAS System Options: Reference*

RESETLINE Statement

Restarts the program line numbers in the SAS log to 1.

Valid in: Anywhere

Category: Log Control

Type: Executable

Syntax

RESETLINE;

Without Arguments

Use the RESETLINE statement to reset the program line numbers in the SAS log to 1.

Details

Program statements are identified by line numbers in the SAS log. The line numbers start with 1 and continue with the sequence of line numbering until the end of the SAS session or batch program.

You use the RESETLINE statement in your program to restart the program line numbering at 1.

Note: If you use the SPOOL system option, you can use only the %INCLUDE statement to resubmit lines of code that were submitted after the most recent RESETLINE statement.

Example: Resetting Line Numbers in the SAS Log

This example resets the program line numbers between DATA steps.

```
data a;
   a=1;
run;
resetline;
data b;
   b=2;
run;
```

The program writes these lines to the SAS log:

```
1    data a;
2       a=1;
3    run;

NOTE: The data set WORK.A has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           4.79 seconds
      cpu time            0.28 seconds


4
5    resetline;
1
2    data b;
3       b=2;
4    run;

NOTE: The data set WORK.B has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

RUN Statement

Executes the previously entered SAS statements.

Valid in: Anywhere

Category: Program Control

Syntax

RUN <CANCEL>;

Without Arguments

Without arguments, the RUN statement executes the previously entered SAS statements.

Arguments

CANCEL
terminates the current step without executing it. SAS prints a message that indicates that the step was not executed.

CAUTION
The CANCEL option does not prevent execution of a DATA step that contains a DATALINES or DATALINES4 statement.

CAUTION
The CANCEL option has no effect when you use the KILL option with PROC DATASETS.

Details

Although the RUN statement is not required between steps in a SAS program, using it creates a step boundary and can make the SAS log easier to read.

Examples

Example 1: Executing SAS Statements

This RUN statement marks a step boundary and executes this PROC PRINT step:

```
proc print data=report;
   title 'Status Report';
run;
```

Example 2: Using the CANCEL Option

This example shows the usefulness of the CANCEL option in a line prompt mode session. The fourth statement in the DATA step contains an invalid value for PI (4.13 instead of 3.14). RUN with CANCEL ends the DATA step and prevents it from executing.

```
data circle;
```

```
   infile file-specification;
   input radius;
   c=2*4.13*radius;
run cancel;
```

This message is written to the SAS log:

```
WARNING: DATA step not executed at user's request.
```

%RUN Statement

Ends source statements following a %INCLUDE * statement.

Valid in: Anywhere

Category: Program Control

Syntax

%RUN;

Without Arguments

The %RUN statement causes SAS to stop reading input from the keyboard (including subsequent SAS statements on the same line as %RUN) and resume reading from the previous input source.

Details

Using the %INCLUDE statement with an asterisk specifies that you enter source lines from the keyboard.

Note: The asterisk (*) cannot be used to specify keyboard entry if you use the Enhanced Editor in the Microsoft Windows operating environment.

Comparisons

The RUN statement executes previously entered DATA or PROC steps. The %RUN statement ends the prompting for source statements and returns program control to the original source program, when you use the %INCLUDE statement to allow data to be entered from the keyboard.

The type of prompt that you use depends on how you run the SAS session. The include operation is most useful in interactive line and noninteractive modes, but it

can also be used in windowing and batch mode. When you are running SAS in batch mode, include the %RUN statement in the external file that is referenced by the SASTERM fileref.

Example: Entering Source Lines from the Keyboard

To request keyboard-entry source in a %INCLUDE statement, follow the statement with an asterisk:

```
%include *;
```

Note: The asterisk (*) cannot be used to specify keyboard entry if you use the Enhanced Editor in the Microsoft Windows operating environment.

When it executes this statement, SAS prompts you to enter source lines from the keyboard. When you finish entering code from the keyboard, enter this statement to return processing to the program that contains the %INCLUDE statement.

```
%run;
```

See Also

Statements:

- "%INCLUDE Statement" on page 132
- "RUN Statement" on page 201

SASFILE Statement

Opens a SAS data set and allocates enough buffers to hold the entire file in memory.

Valid in:	Anywhere
Category:	Program Control
Restriction:	A SAS data set opened by the SASFILE statement can be used for subsequent input (read) or update processing but not for output or utility processing.
See:	"SASFILE Statement: z/OS" in *SAS Companion for z/OS* in *SAS Companion for z/OS*

Syntax

SASFILE <*libref*.> *member-name*<.*member-type*> <(*password-options*)>
OPEN | LOAD | CLOSE;

Arguments

libref

a name that is associated with a SAS library. The libref (library reference) must be a valid SAS name. The default libref is either User (if assigned) or Work (if User is not assigned).

Range 1 to 8 bytes

Restriction The libref cannot represent a concatenation of SAS libraries that contain a library in sequential format.

member-name

a valid SAS name that is a SAS data file. The data file is a member of the SAS library associated with the libref.

Restriction The SAS data set must have been created with the V7, V8, or V9 Base SAS engine.

member-type

the type of SAS file to be opened. A valid value is DATA, which is the default.

password-options

specifies one or more of these password options:

ENCRYPTKEY=*key-value*

enables the SASFILE statement to open an AES-encrypted SAS data file. If a SAS data file is encrypted with the AES (Advanced Encryption Standard) algorithm, a key value is assigned to the file and must be specified in order to access the file. The key value can be up to 64 bytes long.

Interaction If you do not specify the ENCRYPTKEY= option for an AES-encrypted SAS data file, a dialog box prompts you to specify the key value.

See "AES Encryption" in *SAS Programmer's Guide: Essentials*

READ=*password*

enables the SASFILE statement to open a read-protected file. The *password* must be a valid SAS name.

WRITE=*password*

enables the SASFILE statement to use the WRITE password to open a file that is both read-protected and write-protected. The *password* must be a valid SAS name.

ALTER=*password*

enables the SASFILE statement to use the ALTER password to open a file that is both read-protected and alter-protected. The *password* must be a valid SAS name.

PW=*password*

enables the SASFILE statement to use the password to open a file that is assigned for all levels of protection. The *password* must be a valid SAS name.

Tip When SASFILE is executed, SAS checks whether the file is read-protected. Therefore, if the file is read-protected, you must include the READ= password in the SASFILE statement. If the file is either write-protected or alter-protected, you can use a WRITE=, ALTER=, or PW= password. However, the file is opened only in input (read) mode. For subsequent processing, you must specify the necessary password or passwords. See "Example 2: Specifying Passwords with the SASFILE Statement" on page 210.

OPEN
opens the file, allocates the buffers, but defers reading the data into memory until a procedure, statement, or application is executed.

LOAD
opens the file, allocates the buffers, and reads the data into memory.

Note: If the total number of allowed buffers is less than the number of buffers required for the file based on the number of data set pages and index file pages, SAS issues a warning about how many pages are read into memory.

CLOSE
frees the buffers and closes the file.

Details

General Information

The SASFILE statement opens a SAS data set and allocates enough buffers to hold the entire file in memory. After the file is read, data is held in memory, available to subsequent DATA and PROC steps or applications, until either a second SASFILE statement closes the file and frees the buffers or the program ends. Ending the program automatically closes the file and frees the buffers.

Using the SASFILE statement can improve performance by reducing these tasks:

- multiple open or close operations (including allocation and freeing of memory for buffers) to process a SAS data set to one open or close operation
- I/O processing by holding the data in memory

If your SAS program consists of steps that read a SAS data set multiple times and you have an adequate amount of memory so that the entire file can be held in real memory, the program should benefit from using the SASFILE statement. Also, SASFILE is especially useful as part of a program that starts a SAS server such as a SAS/SHARE server. However, it is recommended that you set up a test in your environment to measure performance with and without the SASFILE statement.

Processing a SAS Data Set Opened with SASFILE

When the SASFILE statement executes, SAS opens the specified file. When subsequent DATA and PROC steps execute, SAS does not open the file for each

request; the file remains open until a second SASFILE statement closes it or the program or session ends.

When a SAS data set is opened by the SASFILE statement, the file is opened for input processing and can be used for subsequent input or update processing. However, the file cannot be used for subsequent utility or output processing, because utility and output processing requires exclusive access to the file (member-level locking). For example, you cannot replace the file or rename its variables.

This table provides a list of several SAS procedures and statements and specifies whether they are allowed if the file is opened by the SASFILE statement.

Table 2.4 *Processing Requests for a File Opened by SASFILE*

Processing Request	Open Mode	Allowed
APPEND procedure	update	Yes
DATA step that creates or replaces the file	output	No
DATASETS procedure to rename or add a variable, add or change a label, or add or remove integrity constraints or indexes	utility	No
DATASETS procedure with AGE, CHANGE, or DELETE statements	does not open the file but requires exclusive access	No
FSEDIT procedure	update	Yes
PRINT procedure	input	Yes
SORT procedure that replaces the original data set with a sorted one	output	No
SQL procedure to modify, add, or delete observations	update	Yes
SQL procedure with the CREATE TABLE or CREATE VIEW statement	output	No
SQL procedure to create or remove integrity constraints or indexes	utility	No

Buffer Allocation

A buffer is a reserved area of memory that holds a segment of data while it is processed. The number of allocated buffers determines how much data can be held in memory at one time.

The number of buffers is not a permanent attribute of a SAS file. That is, it is valid only for the current SAS session or job. When a SAS file is opened, a default number of buffers for processing the file is set. The default depends on the operating environment but is typically a small number. To specify a different number of buffers, use the BUFNO= data set option or system option.

When the SASFILE statement is executed, SAS automatically allocates the number of buffers based on the number of data set pages and index file pages (if an index file exists). For example:

- If the number of data set pages is five and there is no index file, SAS allocates five buffers.
- If the number of data set pages is 500 and the number of index file pages is 200, SAS allocates 700 buffers.

If a file that is held in memory increases in size during processing, the number of allocated buffers increases to accommodate the file. If SASFILE is executed for a SAS data set, the BUFNO= option is ignored.

I/O Processing

An I/O (input/output) request reads a segment of data from a storage device (such as a disk) and transfers the data to memory, or conversely transfers the data from memory and writes it to the storage device. When a SAS data set is opened by the SASFILE statement, data is read once and held in memory, which should reduce the number of I/O requests.

CAUTION

I/O processing can be reduced only if there is sufficient real memory. If the SAS data set is very large, you might not have sufficient real memory to hold the entire file. If insufficient memory exists, your operating environment can simulate more memory than actually exists, which is virtual memory. If virtual memory occurs, data access I/O requests are replaced with swapping I/O requests, which could result in no performance improvement. In addition, both SAS and your operating environment have a maximum amount of memory that can be allocated, which could be exceeded by the needs of your program. If your program needs exceed the memory that is available, the number of allocated buffers might be decreased to the default allocation in order to free memory.

TIP To determine how much memory a SAS data set requires, execute the CONTENTS procedure for the file to list its page size, the number of data set pages, the index file size, and the number of index file pages.

Using the SASFILE Statement in a SAS/SHARE Environment

Here are considerations for using the SASFILE statement with SAS/SHARE software:

- You must execute the SASFILE statement before you execute the PROC SERVER statement.
- If the client (the computer on which you use a SAS session to access a SAS/SHARE server) executes the SASFILE statement, it is rejected.
- After the SASFILE statement is executed, all users who subsequently open the file access the data held in memory instead of data that is stored on the disk.
- After the SASFILE statement is executed, you cannot close the file and free the buffers until the SAS/SHARE server is terminated.
- You can use the ALLOCATE SASFILE command for the PROC SERVER statement as an alternative that brings part of the file into memory (controlled by the BUFNO= option).
- If the SASFILE statement is executed and you execute ALLOCATE SASFILE by specifying a value for BUFNO= that is a larger number of buffers than allocated by SASFILE, performance is not improved.

Comparisons

- Use the BUFNO= system option or data set option to specify a specific number of buffers.
- With SAS/SHARE software, you can use the ALLOCATE SASFILE command for the PROC SERVER statement to bring part of the file into memory (controlled by the BUFNO= option).

Examples

Example 1: Using SASFILE in a Program with Multiple Steps

This SAS program illustrates the process of opening a SAS data set, transferring its data to memory, and reading that data held in memory for multiple tasks. The program consists of steps that read the file multiple times.

```
libname mydata 'SAS-library';
sasfile mydata.census.data open;   1
data test1;
   set mydata.census;   2
run;
data test2;
   set mydata.census;   3
run;
proc summary data=mydata.census print;   4
run;
data mydata.census;   5
```

```
   modify mydata.census;
   .
   . (statements to modify data)
   .
run;
sasfile mydata.census close;  6
```

1 Opens SAS data set MyData.Census and allocates the number of buffers based on the number of data set pages and index file pages.

2 Reads all pages of MyData.Census and transfers all data from disk to memory.

3 Reads MyData.Census a second time, but this time from memory without additional I/O requests.

4 Reads MyData.Census a third time, again from memory without additional I/O requests.

5 Reads MyData.Census a fourth time, again from memory without additional I/O requests. If the MODIFY statement successfully changes data in memory, the changed data is transferred from memory to disk at the end of the DATA step.

6 Closes MyData.Census and frees allocated buffers.

Example 2: Specifying Passwords with the SASFILE Statement

This SAS program illustrates using the SASFILE statement and specifying passwords for a SAS data set that is both read-protected and alter-protected.

```
libname mydata 'SAS-library';
sasfile mydata.census (read=gizmo) open;  1
proc print data=mydata.census (read=gizmo);  2
run;
data mydata.census;
   modify mydata.census (alter=luke);  3
   .
   . (statements to modify data)
   .
run;
```

1 The SASFILE statement specifies the READ password, which is sufficient to open the file.

2 In the PRINT procedure, the READ password must be specified again.

3 The ALTER password is used in the MODIFY statement, because the data set is being updated.

Note: It is acceptable to use the higher-level ALTER password instead of the READ password in the preceding example.

See Also

- For information about using the SASFILE statement in a SAS/SHARE environment, see "SERVER Procedure" in *SAS/SHARE User's Guide*.

Data Set Options:

- "BUFNO= Data Set Option" in *SAS Data Set Options: Reference*

System Options:

- "BUFNO= System Option" in *SAS System Options: Reference*

SKIP Statement

Creates a blank line in the SAS log.

Valid in: Anywhere

Category: Log Control

Syntax

SKIP <*n*>;

Without Arguments

Using SKIP without arguments causes SAS to create one blank line in the log.

Arguments

n

specifies the number of blank lines that you want to create in the log.

Tip If the number specified is greater than the number of lines that remain on the page, SAS goes to the top of the next page.

Details

The SKIP statement itself does not appear in the log. You can use this statement in all methods of operation.

See Also

Statements:

- "PAGE Statement" on page 199

System Options:

- "LINESIZE= System Option" in *SAS System Options: Reference*
- "PAGESIZE= System Option" in *SAS System Options: Reference*

SYSECHO Statement

Sends a global statement complete event and passes a text string back to the IOM client.

Valid in: Anywhere

Category: Program Control

Restriction: Has an effect only in objectserver mode

Syntax

SYSECHO <"*text*"> ;

Without Arguments

Using SYSECHO without arguments sends a global statement complete event to the IOM client.

Arguments

"text"
specifies a text string that is passed back to the IOM client.

Range 1–64 characters

Requirement The text string must be enclosed in double quotation marks.

Details

The SYSECHO statement enables IOM clients to manually track the progress of a segment of a submitted SAS program.

When the SYSECHO statement is executed, a global statement complete event is generated and, if specified, the text string is passed back to the IOM client.

TITLE Statement

Specifies title lines for SAS output.

Valid in: Anywhere

Category: Output Control

Restriction: The TITLE statement does not support Unicode.

See: TITLE Statement under Windows, UNIX, or z/OS

Syntax

TITLE <*n*> <*ods-format-options*> <'*text*' | "*text*">;

Without Arguments

Using TITLE without arguments cancels all existing titles.

Arguments

n

specifies the relative line that contains the title line.

Range 1–10

Tips The title line with the highest number appears on the bottom line. If you omit *n*, SAS assumes a value of 1. Therefore, you can specify TITLE or TITLE1 for the first title line.

You can create titles that contain blank lines between the lines of text. For example, if you specify text with a TITLE statement and a TITLE3 statement, there is a blank line between the two lines of text.

ods-format-options

specifies formatting options for the ODS HTML, RTF, and PRINTER destinations.

BOLD

specifies that the title text is bold font weight.

ODS destination HTML, RTF, PRINTER

COLOR=*color*

specifies the title text color.

Alias C

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

BCOLOR=*color*

specifies the background color of the title block.

ODS destination HTML, RTF, PRINTER

FONT=*font-face*
specifies the font to use. If you supply multiple fonts, then the destination device uses the first one that is installed on your system.

Alias F

ODS destination HTML, RTF, PRINTER

HEIGHT=*dimension* | *size*
specifies the size of the font for titles.

dimension
is a nonnegative number.

Units of Measure for Dimension

cm	Centimeters
em	Standard typesetting measurement unit for width
ex	Standard typesetting measurement unit for height
in	Inches
mm	Millimeters
pt	A printer's point

Restriction If you specify *dimension*, then specify a unit of measure. Without a unit of measure, the number becomes a relative size.

size
The value of *size* is relative to all other font sizes in the HTML document.

Range 1 to 7

Alias H

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

ITALIC
specifies that the title text is in italics.

ODS destination HTML, RTF, PRINTER

JUSTIFY= CENTER | LEFT | RIGHT
specifies justification.

CENTER
specifies center justification.

Alias C

LEFT
specifies left justification.

Alias L

RIGHT
specifies right justification.

Alias R

Alias J

ODS destination HTML, RTF, PRINTER

Example "Example 3: Customizing Titles and Footnotes By Using the Output Delivery System" on page 218

LINK='*url*'
specifies a hyperlink.

ODS destination HTML, RTF, PRINTER

Tip The visual properties for LINK= always come from the current style.

UNDERLIN= 0 | 1 | 2 | 3
specifies whether the subsequent text is underlined. Value 0 indicates no underlining. Values 1, 2, and 3 indicate underlining.

Alias U

ODS destination HTML, RTF, PRINTER

Tip ODS generates the same type of underline for values 1, 2, and 3. However, SAS/GRAPH uses values 1, 2, and 3 to generate increasingly thicker underlines.

Note The defaults for how ODS renders the TITLE statement come from style elements relating to system titles in the current style. The TITLE statement syntax with *ods-format-options* is a way to override the settings provided by the current style. The current style varies according to the ODS destination. For more information about how to determine the current style, see "Understanding Styles, Style Elements, and Style Attributes" in *SAS Output Delivery System: Procedures Guide*. Also see "TEMPLATE Procedure: Creating a Style Template" in *SAS Output Delivery System: Procedures Guide*.

Tips You can specify these options by letter, word, or words by preceding each letter or word of *text* by the option.

For example, this code makes the title "Red, White, and Blue" appear in different colors.

```
title color=red "Red," color=white "White, and" color=blue "Blue";
```

'text' | "text"
specifies text that is enclosed in single or double quotation marks.

You can customize titles by inserting BY variable values (#BYVAL*n*), BY variable names (#BYVAR*n*), or BY lines (#BYLINE) in titles that are specified in PROC steps. Embed the items in the specified title text string at the position where you want the substitution text to appear.

#BYVAL*n*
#BYVAL(*variable-name*)
substitutes the current value of the specified BY variable for #BYVAL in the text string and displays the value in the title.

Follow these rules when you use #BYVAL in the TITLE statement of a PROC step:

- Specify the variable that is used by #BYVAL in the BY statement.
- Insert #BYVAL in the specified title text string at the position where you want the substitution text to appear.
- Follow #BYVAL with a delimiting character, either a space or other non-alphanumeric character (for example, a quotation mark) that ends the text string.
- If you want the #BYVAL substitution to be followed immediately by other text, with no delimiter, use a trailing dot (as with macro variables).

Specify the variable with one of these values:

n
specifies which variable in the BY statement #BYVAL should use. The value of *n* indicates the position of the variable in the BY statement.

Example **`#BYVAL2`** specifies the second variable in the BY statement.

variable-name
names the BY variable.

Tip *Variable-name* is not case sensitive.

Example **`#BYVAL(YEAR)`** specifies the BY variable, YEAR.

#BYVAR*n*
#BYVAR(*variable-name*)
substitutes the name of the BY variable or label that is associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string and displays the name or label in the title.

Follow these rules when you use #BYVAR in the TITLE statement of a PROC step:

- Specify the variable that is used by #BYVAR in the BY statement.
- Insert #BYVAR in the specified title text string at the position where you want the substitution text to appear.

- Follow #BYVAR with a delimiting character, either a space or other non-alphanumeric character (for example, a quotation mark) that ends the text string.
- If you want the #BYVAR substitution to be followed immediately by other text, with no delimiter, use a trailing dot (as with macro variables).

Specify the variable with one of these values:

n
specifies which variable in the BY statement #BYVAR should use. The value of *n* indicates the position of the variable in the BY statement.

Example **`#BYVAR2`** specifies the second variable in the BY statement.

variable-name
names the BY variable.

Tip *variable-name* is not case-sensitive.

Example **`#BYVAR(SITES)`** specifies the BY variable SITES.

#BYLINE
substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string and displays the BY line in the title.

Tip #BYLINE produces output that contains a BY line at the top of the page unless you suppress the BY line by using NOBYLINE in an OPTIONS statement.

See For more information about NOBYLINE, see the "BYLINE System Option" in *SAS System Options: Reference*.

Tips For compatibility with previous releases, SAS accepts some text without quotation marks. When writing new programs or updating existing programs, always enclose text in quotation marks.

You can use macro variables and macros to change the information in TITLE statements. If the title is enclosed in double quotation marks (""), the text indicated is substituted into the title. If the title is enclosed in single quotation marks (''), the text is not substituted.

You can use macro variables and macros to change the information in TITLE statements. The SAS macro facility resolves the macro variable.

See For more information about including quotation marks as part of the title, see "Expressions" in *SAS Language Reference: Concepts* .

Details

A TITLE statement takes effect when the step or RUN group with which it is associated executes. After you specify a title for a line, it is used for all subsequent output until you cancel the title or define another title for that line. A TITLE

statement for a given line cancels the previous TITLE statement for that line and for all lines with larger *n* numbers.

Operating Environment Information: The maximum title length that is allowed depends on your operating environment and the value of the LINESIZE= system option. For more information, see the SAS documentation for your operating environment.

Comparisons

You can also create titles with the TITLES window.

Examples

Example 1: Using the TITLE Statement

These examples show how you can use the TITLE statement.

- This statement suppresses a title on line *n* and all lines after it:

```
titlen;
```

- These code lines are examples of TITLE statements:
  - `title 'First Draft';`
  - `title2 "Year's End Report";`
  - `title2 'Year''s End Report';`

Example 2: Customizing Titles By Using BY Variable Values

You can customize titles by inserting BY variable values in the titles that you specify in PROC steps. These examples show how to use #BYVAL*n*, #BYVAR*n*, and #BYLINE:

- `title 'Quarterly Sales for #byval(site)';`
- `title 'Annual Costs for #byvar2';`
- `title 'Data Group #byline';`

Example 3: Customizing Titles and Footnotes By Using the Output Delivery System

You can customize titles and footnotes with ODS. This example shows you how to change the color, justification, and size of the text for the title and footnote:

```
ods html path='c:\temp' file='test.html';

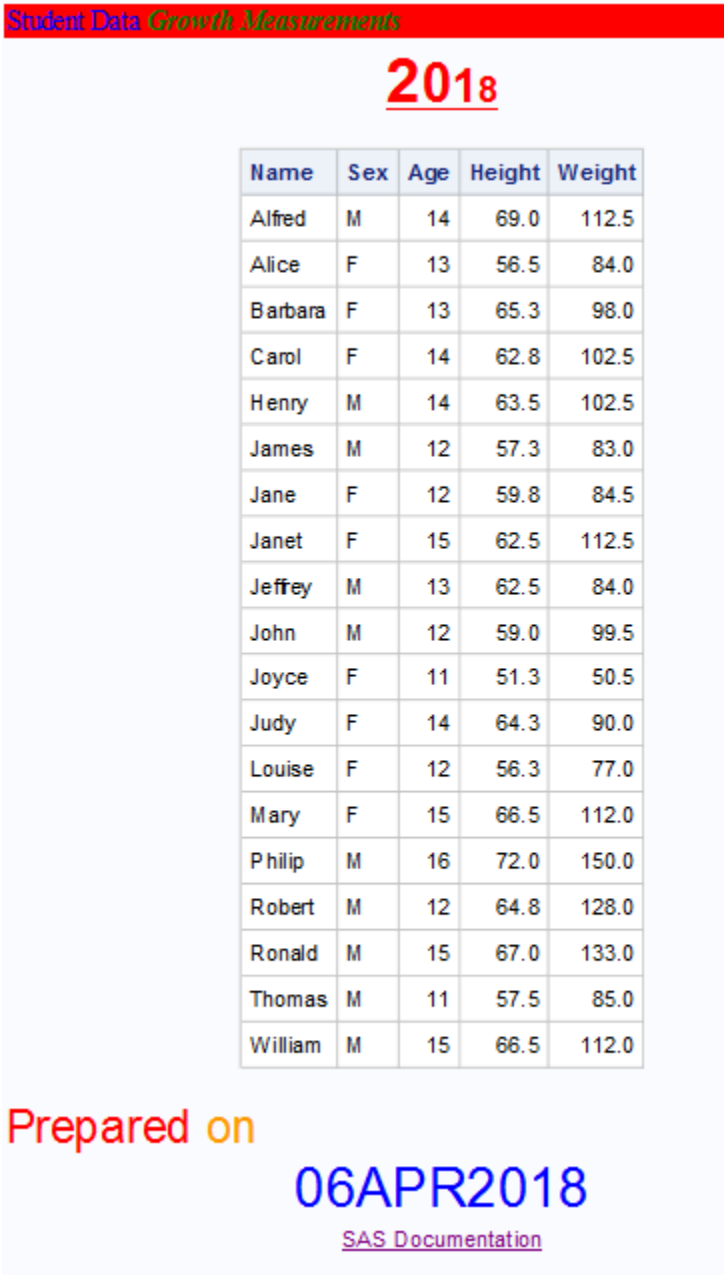
title j=left
font= 'Times New Roman' color=blue bcolor=red "Student Data "
c=green bold italic "Growth Measurements";
title2 j=center color=red underlin=1
```

```
height=28pt "2"
height=24pt "0"
height=20pt "1"
height=16pt "8";
footnote j=left height=20pt
color=red "Prepared "
c='#FF9900' "on";
footnote2 j=center color=blue
height=24pt "&sysdate9";
footnote3 link='http://support.sas.com/documentation' "SAS
Documentation";

proc print data=sashelp.class noobs;
run;

ods html close;
```

Output 2.1 *Output with Customized Titles and Footnotes*


Student Data Growth Measurements

2018

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0

Prepared on

06APR2018

SAS Documentation

See Also

- "Overview" in *SAS Output Delivery System: Procedures Guide*

Statements:

- "FOOTNOTE Statement" on page 128

System Options:

- "LINESIZE= System Option" in *SAS System Options: Reference*

X Statement

Issues an operating-system command from within a SAS session.

Valid in: Anywhere

Restriction: In SAS Viya, the SAS Compute Server runs SAS sessions with the XCMD system option disabled (set to NOXCMD by default). When NOXCMD is set, the X statement is not valid. Your system administrator must configure the server to enable the XCMD system option. For more information, see Configure Servers to Allow XCMDs

Requirement: The XCMD system option must be enabled to submit X commands in your SAS program. For more information about the XCMD system option, see the documentation for your operating system:

- "XCMD System Option: Windows" in *SAS Companion for Windows*
- "XCMD System Option: UNIX" in *SAS Companion for UNIX Environments*
- "XCMD System Option: z/OS" in *SAS Companion for z/OS*

Syntax

X <'*operating-environment-command*'>;

Without Arguments

Using X without arguments places you in your operating environment, where you can issue commands that are specific to your environment.

Arguments

'operating-environment-command'
specifies an operating-environment command that is enclosed in quotation marks.

Details

In all operating environments, you can use the X statement when you run SAS in windowing or interactive line mode. In some operating environments, you can use the X statement when you run SAS in batch or noninteractive mode.

Operating Environment Information: The X statement is dependent on your operating environment. See the SAS documentation for your operating environment to determine whether it is a valid statement on your system. Keep in mind that the way you return from operating-environment mode to the SAS session

is dependent on your operating environment and the commands that you use with the X statement are specific to your operating environment.

You can use the X statement with SAS macros to write a SAS program that can run in multiple operating environments. For more information, see *SAS Macro Language: Reference*.

XCMD functionality can be processed in SAS code. You can use SAS functions and call routines that have XCMD functionality. For users who have not enabled the X command in their environment, this table shows a sample of alternative calls that can be used:

Table 2.5 *XCMD Functionality That Exists in SAS Functions and Call Routines*

Functionality	XCMD/OS Function	SAS Language Element
Access network endpoint	wget/curl	PROC HTTP
Copy a file	cp	FCOPY
Create a directory	mkdir	DCREATE
Delete a file or directory	rm	FDELETE
Rename a file	mv	RENAME
Manipulate ZIP file	zip/unzip	FILENAME Statement: ZIP
Searches files and directories for specified criteria	find	FILENAME DOPEN DREAD DCLOSE PRXMATCH
Searches data sets for lines that match a regular expression.	grep	PRXMATCH
Specify file metadata information.	ls	FINFO
Specify the directories and files within the current directory.	dir	FILENAME DOPEN DREAD DCLOSE

Comparisons

In a windowing session, the X command works exactly like the X statement except that you issue the command from a command line. You submit the X statement from the Program Editor window.

The X statement is similar to the SYSTEM function, the X command, and the CALL SYSTEM routine. In most cases, the X statement, X command or %SYSEXEC macro statement are preferable because they require less overhead. However, the SYSTEM function can be executed conditionally. The X statement is a global statement and executes as a DATA step is being compiled.

See Also

CALL Routines:

- "CALL SYSTEM Routine" in *SAS Functions and CALL Routines: Reference*

Functions:

- "SYSTEM Function" in *SAS Functions and CALL Routines: Reference*

3

Dictionary of SAS Global Statement Environment Variables

Dictionary

SAS_FTP_AUTHTLS Environment Variable

Enables Transport Layer Security (TLS) authentication.

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

Used by: SAS FILENAME statement, FTP Access Method

Syntax

SAS_FTP_AUTHTLS 0 | 1 | 2 | 3

Required Arguments

0

Uses the FILENAME statement, FTP Access Method option to determine TLS security. This is the default setting.

1

Enforces TLS authentication. If security authorization fails, an error is returned.

2

Enforces TLS authentication if the AUTHTLS, PROT=, or PBSZ= option is specified in the FILENAME statement, FTP Access Method.

Note If you do not specify the AUTHTLS, PROT=, or PBSZ= option in the FILENAME statement, FTP Access Method, TLS authentication is attempted. If the FTP server does not accept TLS authentication, then basic FTP authentication is used.

3

Enforces TLS authentication and a note is written to the SAS log. If security authorization fails, an error is returned.

Details

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that are designed to provide communication security over the internet. TLS and SSL are protocols that provide network data privacy, data integrity, and authentication.

The SAS_FTP_AUTHTLS environment variable can be used instead of adding the AUTHTLS, PROT=, or PBSZ= option to your FILENAME FTP access method statement to enforce TLS authentication. If the SAS_FTP_AUTHTLS environment variable is used, the default values of PROT=P and PBSZ=0 are used in conjunction with AUTHTLS.

How you define the SAS environment variables depends on your operating environment. For most operating environments, you can define the environment variables either locally (for use only in your SAS session) or globally. For example, you can define the SAS environment variables with the SET system option in a SAS configuration file, at SAS invocation, with the OPTIONS statement, or in the SAS System Options window. In addition, you can use your operating system to define the environment variables.

This table includes examples of defining the SAS_FTP_AUTHTLS environment variable.

Table 3.1 *Defining the SAS_FTP_AUTHTLS Environment Variable*

Method	Example
SAS configuration file	`-set SAS_FTP_AUTHTLS 1`

Method	Example
SAS invocation	`-set SAS_FTP_AUTHTLS 1`
OPTIONS statement	`options set=SAS_FTP_AUTHTLS 1;`

For more information about TLS, see *Encryption in SAS*

See Also

Statements:

- "FILENAME Statement: FTP Access Method" on page 67

TD_1MB_ROW Environment Variable

Specifies whether response row sizes up to 1MB are supported for data on Teradata.

Category: Data Access

Default: ON

Restriction: This option applies to SAS 9.4M8 only. It is not relevant for SAS Viya 3.5.

Details

Use the TD_1MB_ROW environment variable to specify whether you want to support response rows up to 1MB for the Teradata interface. By default, SAS/ACCESS supports 1MB response rows. If you do not want to support response rows that are this large, set TD_1MB_ROW to OFF. When TD_1MB_ROW is OFF, response rows up to 64K are supported.

In your SAS session, you can set this environment variable by using the OPTIONS statement:

```
options set=TD_1MB_ROW OFF;
```

You can also set this environment variable on invocation of the SAS session, in the SAS autoexec file:

```
sas -nodms -set TD_1MB_ROW OFF
```

For more information, see "Processing Large Response Rows" in *SAS/ACCESS for Relational Databases: Reference*.