### Contents

*About This Book* ................................................................. v

*What’s New in SAS 9.4 Statements* ....................................... xi

**Chapter 1 • SAS Statements** .............................................. 1
  - Definition of Statements ............................................. 1
  - DATA Step Statements ................................................. 2
  - Global Statements ..................................................... 3

**Chapter 2 • Dictionary of SAS Statements** ................................. 5
  - SAS Statements Documented in Other SAS Publications ......... 7
  - DATA Step Statements by Category ................................ 7
  - Global Statements by Category ..................................... 11
  - Dictionary ...................................................................... 14

**Chapter 3 • Dictionary of SAS Statement Environment Variables** .... 449
  - Dictionary ...................................................................... 449

*Recommended Reading* .......................................................... 451

*Index* .................................................................................. 453
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 <option(s)> <argument-2 <option(s)> ...>**

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-option(s))>**

```
<value-range-set-1 <(picture-1-option(s))>
<value-range-set-2 <(picture-2-option(s))> ...>;
```
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:

```sas
infile file-specification obs = 100;
libname libref 'SAS-library';
```
About This Book
What’s New in SAS 9.4
Statements

Overview

New and enhanced features enable you to do the following:

• write a file to a SharePoint document library
• read data from user-specified text and to access ZIP files
• create and name a variable that contains the observation number that was just read from the data set
• control whether a KEY= search should begin at the top of the index for the data set that is being read
• 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
• open an AES (Advanced Encryption Standard) encrypted SAS data file
• 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.
• 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.

Writing Files into SharePoint

You can use the FILENAME statement, WebDAV access method to write a file to a SharePoint document library. To enable the ability to write a file to a SharePoint site, you must set the SHAREPOINT_COMP_MODE environment variable.

New SAS Statements

The following SAS statements are new:

FILENAME, DATAURL Access Method (p. 108)
   enables you to read data from user-specified text.
FILENAME, ZIP Access Method (p. 174)
   enables you to access ZIP files.

Enhanced SAS Statements

The following SAS statements have been enhanced:

FILENAME, EMAIL (SMTP) Access Method. (p. 111)
   • In the second maintenance release for SAS 9.4, 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 (p. 126)
   In the third maintenance release for SAS 9.4, 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 (p. 141)
• In the third maintenance release for SAS 9.4, 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 the second maintenance release for SAS 9.4, 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 (p. 148)
• 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 (p. 160)
• 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 (p. 166)
• 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 (p. 174)
In the third maintenance release for SAS 9.4, 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** (p. 274)
- The EXTENDOBS COUNTER= 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** (p. 298)
A new option, NOMSG, disables error and warning messages to the SAS log.

**MODIFY** (p. 309)
- A new option, CUROBS, creates and names a variable that contains the observation number that was just read from the data set.
- A new option, KEYRESET, controls whether a KEY= search should begin at the top of the index for the data set that is being read. When the value of the KEYRESET variable is 1, the index lookup begins at the top of the index. When the value of the KEYRESET variable is 0, the index lookup is not reset and the lookup continues where the prior lookup left off.

**SASFILE** (p. 385)
The new option ENCRYPTKEY= enables the SASFILE statement to open an AES (Advanced Encryption Standard) encrypted SAS data file.

**SET** (p. 394)
- A new option, CUROBS, creates and names a variable that contains the observation number that was just read from the data set.
- A new option, KEYRESET, controls whether a KEY= search should begin at the top of the index for the data set that is being read. When the value of the KEYRESET variable is 1, the index lookup begins at the top of the index. When the value of the KEYRESET variable is 0, the index lookup is not reset and the lookup continues where the prior lookup left off.

---

**Locked-Down State Restrictions**

The LOCKDOWN statement and LOCKDOWN system option are new in the first maintenance release for SAS 9.4. 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 the second maintenance release for SAS 9.4, 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.
Chapter 1
SAS Statements

Definition of Statements

A SAS 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. Each SAS statement ends with a semicolon.

This documentation covers two types of SAS statements:

• statements that are used in DATA step programming
• statements that are global in scope and can be used anywhere in a SAS program.

In addition to the statements documented in SAS Statements: Reference, statements are also documented in the following publications:

• Base SAS Procedures Guide
• 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 XML LIBNAME Engine: User's Guide
• SAS/ACCESS for Relational Databases: Reference
• SAS/CONNECT User's Guide
• SAS/SHARE User's Guide
DATA Step Statements

Executable and Declarative Statements

DATA step statements are executable or declarative statements that can appear in the DATA step. Executable statements result in some action during individual iterations of the DATA step; declarative statements supply information to SAS and take effect when the system compiles program statements.

The following tables show the SAS executable and declarative statements that you can use in the DATA step.

Table 1.1 Executable Statements in the DATA Step

<table>
<thead>
<tr>
<th>Executable Statements</th>
<th>IF, Subsetting</th>
<th>PUT, Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>IF, Subsetting</td>
<td>PUT, Column</td>
</tr>
<tr>
<td>Array Reference</td>
<td>IF-THEN/ELSE</td>
<td>PUT, Formatted</td>
</tr>
<tr>
<td>Assignment</td>
<td>INFILE</td>
<td>PUT, List</td>
</tr>
<tr>
<td>CALL</td>
<td>INPUT</td>
<td>PUT, Named</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>GO TO</td>
<td>PUT</td>
</tr>
<tr>
<td>DECLARE</td>
<td>INPUT, Column</td>
<td>PUT, ODS</td>
</tr>
<tr>
<td>DELETE</td>
<td>INPUT, Formatted</td>
<td>PUTLOG</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>INPUT, List</td>
<td>REDIRECT</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>INPUT, Named</td>
<td>REMOVE</td>
</tr>
<tr>
<td>DO</td>
<td>LEAVE</td>
<td>REPLACE</td>
</tr>
<tr>
<td>DO, Iterative</td>
<td>LINK</td>
<td>RESETLINE</td>
</tr>
<tr>
<td>DO UNTIL</td>
<td>LIST</td>
<td>RETURN</td>
</tr>
<tr>
<td>DO WHILE</td>
<td>LOSTCARD</td>
<td>SELECT</td>
</tr>
<tr>
<td>ERROR</td>
<td>MERGE</td>
<td>SET</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>MODIFY</td>
<td>STOP</td>
</tr>
<tr>
<td>FILE</td>
<td>Null</td>
<td>Sum</td>
</tr>
<tr>
<td>FILE, ODS</td>
<td>OUTPUT</td>
<td>UPDATE</td>
</tr>
</tbody>
</table>
Table 1.2  Declarative Statements in the DATA Step

<table>
<thead>
<tr>
<th>Declarative Statements</th>
<th>DATALINES4</th>
<th>Labels, Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRIB</td>
<td>DROP</td>
<td>LENGTH</td>
</tr>
<tr>
<td>BY</td>
<td>END</td>
<td>RENAME</td>
</tr>
<tr>
<td>CARDS</td>
<td>FORMAT</td>
<td>RETAIN</td>
</tr>
<tr>
<td>CARDS4</td>
<td>INFORMAT</td>
<td>WHERE</td>
</tr>
<tr>
<td>DATA</td>
<td>KEEP</td>
<td>WINDOW</td>
</tr>
<tr>
<td>DATALINES</td>
<td>LABEL</td>
<td></td>
</tr>
</tbody>
</table>

DATA step statements can be grouped into six functional categories. For a list of DATA step statements by category, see “DATA Step Statements by Category” on page 7.

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

Global statements can be divided into seven functional categories. For a list of global statements by category, see “Global Statements by Category” on page 11.

Other SAS software products have additional global statements that are used with those products. For information, see the SAS documentation for those products.
Chapter 2
Dictionary of SAS Statements

SAS Statements Documented in Other SAS Publications ..................... 7
DATA Step Statements by Category .................................................. 7
Global Statements by Category ....................................................... 11
Dictionary ....................................................................................... 14
  ABORT Statement ................................................................. 14
  ARRAY Statement ................................................................. 18
  Array Reference Statement ..................................................... 23
  Assignment Statement .......................................................... 26
  ATTRIB Statement ................................................................. 27
  BY Statement ....................................................................... 31
  CALL Statement ................................................................. 36
  CARDS Statement ............................................................... 37
  CARDS4 Statement ............................................................... 37
  CATNAME Statement ............................................................ 38
  CHECKPOINT EXECUTE_ALWAYS Statement ............................... 41
  Comment Statement ............................................................. 41
  CONTINUE Statement .......................................................... 43
  DATA Statement ................................................................. 44
  DATALINES Statement .......................................................... 53
  DATALINES4 Statement ........................................................ 54
  DELETE Statement ............................................................... 55
  DESCRIBE Statement ............................................................ 56
  DISPLAY Statement .............................................................. 57
  DM Statement ................................................................. 59
  DO Statement ..................................................................... 61
  DO Statement, Iterative ........................................................ 62
  DO UNTIL Statement ........................................................... 66
  DO WHILE Statement .......................................................... 67
  DROP Statement ................................................................. 68
  END Statement ................................................................. 70
  ENDSAS Statement .............................................................. 71
  ERROR Statement ............................................................... 71
  EXECUTE Statement ............................................................ 73
  FILE Statement ................................................................. 73
  FILENAME Statement ........................................................... 94
  FILENAME Statement, CATALOG Access Method ......................... 103
  FILENAME Statement, CLIPBOARD Access Method ..................... 106
  FILENAME Statement, DATAURL Access Method ....................... 108
  FILENAME Statement, EMAIL (SMTP) Access Method ................. 111
  FILENAME Statement, FTP Access Method ............................... 126
<table>
<thead>
<tr>
<th>Statement</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILENAME Statement, Hadoop Access Method</td>
<td>141</td>
</tr>
<tr>
<td>FILENAME Statement, SFTP Access Method</td>
<td>148</td>
</tr>
<tr>
<td>FILENAME Statement, SOCKET Access Method</td>
<td>155</td>
</tr>
<tr>
<td>FILENAME Statement, URL Access Method</td>
<td>160</td>
</tr>
<tr>
<td>FILENAME Statement, WebDAV Access Method</td>
<td>166</td>
</tr>
<tr>
<td>FILENAME Statement, ZIP Access Method</td>
<td>174</td>
</tr>
<tr>
<td>FOOTNOTE Statement</td>
<td>178</td>
</tr>
<tr>
<td>FORMAT Statement</td>
<td>182</td>
</tr>
<tr>
<td>GO TO Statement</td>
<td>185</td>
</tr>
<tr>
<td>IF Statement, Subsetting</td>
<td>187</td>
</tr>
<tr>
<td>IF-THEN/ELSE Statement</td>
<td>189</td>
</tr>
<tr>
<td>%INCLUDE Statement</td>
<td>191</td>
</tr>
<tr>
<td>INFILE Statement</td>
<td>198</td>
</tr>
<tr>
<td>INFORMAT Statement</td>
<td>228</td>
</tr>
<tr>
<td>INPUT Statement</td>
<td>232</td>
</tr>
<tr>
<td>INPUT Statement, Column</td>
<td>248</td>
</tr>
<tr>
<td>INPUT Statement, Formatted</td>
<td>251</td>
</tr>
<tr>
<td>INPUT Statement, List</td>
<td>255</td>
</tr>
<tr>
<td>INPUT Statement, Named</td>
<td>262</td>
</tr>
<tr>
<td>KEEP Statement</td>
<td>266</td>
</tr>
<tr>
<td>LABEL Statement</td>
<td>267</td>
</tr>
<tr>
<td>label: Statement</td>
<td>269</td>
</tr>
<tr>
<td>LEAVE Statement</td>
<td>270</td>
</tr>
<tr>
<td>LENGTH Statement</td>
<td>271</td>
</tr>
<tr>
<td>LIBNAME Statement</td>
<td>274</td>
</tr>
<tr>
<td>LIBNAME Statement for the JMP Engine</td>
<td>288</td>
</tr>
<tr>
<td>LIBNAME Statement for WebDAV Server Access</td>
<td>289</td>
</tr>
<tr>
<td>LINK Statement</td>
<td>293</td>
</tr>
<tr>
<td>LIST Statement</td>
<td>295</td>
</tr>
<tr>
<td>%LIST Statement</td>
<td>297</td>
</tr>
<tr>
<td>LOCK Statement</td>
<td>298</td>
</tr>
<tr>
<td>LOSTCARD Statement</td>
<td>301</td>
</tr>
<tr>
<td>MERGE Statement</td>
<td>304</td>
</tr>
<tr>
<td>MISSING Statement</td>
<td>308</td>
</tr>
<tr>
<td>MODIFY Statement</td>
<td>309</td>
</tr>
<tr>
<td>Null Statement</td>
<td>329</td>
</tr>
<tr>
<td>OPTIONS Statement</td>
<td>331</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>332</td>
</tr>
<tr>
<td>PAGE Statement</td>
<td>335</td>
</tr>
<tr>
<td>PUT Statement</td>
<td>335</td>
</tr>
<tr>
<td>PUT Statement, Column</td>
<td>354</td>
</tr>
<tr>
<td>PUT Statement, Formatted</td>
<td>356</td>
</tr>
<tr>
<td>PUT Statement, List</td>
<td>360</td>
</tr>
<tr>
<td>PUT Statement, Named</td>
<td>365</td>
</tr>
<tr>
<td>PUTLOG Statement</td>
<td>367</td>
</tr>
<tr>
<td>REDIRECT Statement</td>
<td>369</td>
</tr>
<tr>
<td>REMOVE Statement</td>
<td>370</td>
</tr>
<tr>
<td>RENAME Statement</td>
<td>372</td>
</tr>
<tr>
<td>REPLACE Statement</td>
<td>374</td>
</tr>
<tr>
<td>RESETLINE Statement</td>
<td>376</td>
</tr>
<tr>
<td>RETAIN Statement</td>
<td>378</td>
</tr>
<tr>
<td>RETURN Statement</td>
<td>382</td>
</tr>
<tr>
<td>RUN Statement</td>
<td>383</td>
</tr>
<tr>
<td>%RUN Statement</td>
<td>384</td>
</tr>
<tr>
<td>SASFILE Statement</td>
<td>385</td>
</tr>
<tr>
<td>SELECT Statement</td>
<td>391</td>
</tr>
</tbody>
</table>
SAS Statements Documented in Other SAS Publications

Some statements are documented with related subject matter in other SAS publications.

- Application Messaging with SAS
- SAS Companion for Windows
- SAS Companion for UNIX Environments
- SAS Companion for z/OS
- SAS DS2 Language Reference
- SAS FedSQL Language Reference
- SAS Language Interfaces to Metadata
- SAS Macro Language: Reference
- SAS Scalable Performance Data Engine: Reference
- SAS/ACCESS for Relational Databases: Reference
- SAS/CONNECT User's Guide
- SAS/SHARE User's Guide

DATA Step Statements by Category

In addition to being either executable or declarative, SAS DATA step statements can be grouped into five functional categories:
Table 2.1  Categories of DATA Step Statements

<table>
<thead>
<tr>
<th>Statements Category</th>
<th>Functionality</th>
</tr>
</thead>
</table>
| Action              | • create and modify variables  
                     | • select only certain observations to process in the DATA step  
                     | • look for errors in the input data  
                     | • work with observations as they are being created |
| Control             | • skip statements for certain observations  
                     | • change the order that statements are executed  
                     | • transfer control from one part of a program to another |
| File-handling       | • work with files used as input to the data set  
                     | • work with files to be written by the DATA step |
| Information         | • give SAS additional information about the program data vector  
                     | • give SAS additional information about the data set or data sets that are being created. |
| Window Display      | • display and customize windows. |

The following table lists and briefly describes the DATA step statements by category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>ABORT Statement (p. 14)</td>
<td>Stops executing the current DATA step, SAS job, or SAS session.</td>
</tr>
<tr>
<td></td>
<td>Assignment Statement (p. 26)</td>
<td>Evaluates an expression and stores the result in a variable.</td>
</tr>
<tr>
<td></td>
<td>CALL Statement (p. 36)</td>
<td>Invokes a SAS CALL routine.</td>
</tr>
<tr>
<td></td>
<td>DELETE Statement (p. 55)</td>
<td>Stops processing the current observation.</td>
</tr>
<tr>
<td></td>
<td>DESCRIBE Statement (p. 56)</td>
<td>Retrieves source code from a stored compiled DATA step program or a DATA step view.</td>
</tr>
<tr>
<td></td>
<td>ERROR Statement (p. 71)</td>
<td>Sets <em>ERROR</em> to 1. A message written to the SAS log is optional.</td>
</tr>
<tr>
<td></td>
<td>EXECUTE Statement (p. 73)</td>
<td>Executes a stored compiled DATA step program.</td>
</tr>
<tr>
<td></td>
<td>IF Statement, Subsetting (p. 187)</td>
<td>Continues processing only those observations that meet the condition of the specified expression.</td>
</tr>
<tr>
<td></td>
<td>LIST Statement (p. 295)</td>
<td>Writes to the SAS log the input data record for the observation that is being processed.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Category</strong></td>
<td><strong>Language Elements</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Language Elements</strong></td>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>LOSTCARD Statement (p. 301)</td>
<td>Resynchronizes the input data when SAS encounters a missing or invalid record in data that has multiple records per observation.</td>
<td></td>
</tr>
<tr>
<td>OUTPUT Statement (p. 332)</td>
<td>Writes the current observation to a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>PUTLOG Statement (p. 367)</td>
<td>Writes a message to the SAS log.</td>
<td></td>
</tr>
<tr>
<td>REDIRECT Statement (p. 369)</td>
<td>Points to different input or output SAS data sets when you execute a stored program.</td>
<td></td>
</tr>
<tr>
<td>REMOVE Statement (p. 370)</td>
<td>Deletes an observation from a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>REPLACE Statement (p. 374)</td>
<td>Replaces an observation in the same location.</td>
<td></td>
</tr>
<tr>
<td>STOP Statement (p. 409)</td>
<td>Stops execution of the current DATA step.</td>
<td></td>
</tr>
<tr>
<td>Sum Statement (p. 410)</td>
<td>Adds the result of an expression to an accumulator variable.</td>
<td></td>
</tr>
<tr>
<td>WHERE Statement (p. 427)</td>
<td>Selects observations from SAS data sets that meet a particular condition.</td>
<td></td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td><strong>Language Elements</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Language Elements</strong></td>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>CONTINUE Statement (p. 43)</td>
<td>Stops processing the current DO-loop iteration and resumes processing the next iteration.</td>
<td></td>
</tr>
<tr>
<td>DO Statement (p. 61)</td>
<td>Specifies a group of statements to be executed as a unit.</td>
<td></td>
</tr>
<tr>
<td>DO Statement, Iterative (p. 62)</td>
<td>Executes statements between the DO and END statements repetitively, based on the value of an index variable.</td>
<td></td>
</tr>
<tr>
<td>DO UNTIL Statement (p. 66)</td>
<td>Executes statements in a DO loop repetitively until a condition is true.</td>
<td></td>
</tr>
<tr>
<td>DO WHILE Statement (p. 67)</td>
<td>Executes statements in a DO-loop repetitively while a condition is true.</td>
<td></td>
</tr>
<tr>
<td>END Statement (p. 70)</td>
<td>Ends a DO group or SELECT group processing.</td>
<td></td>
</tr>
<tr>
<td>GO TO Statement (p. 185)</td>
<td>Directs program execution immediately to the statement label that is specified and, if followed by a RETURN statement, returns execution to the beginning of the DATA step.</td>
<td></td>
</tr>
<tr>
<td>IF-THEN/ELSE Statement (p. 189)</td>
<td>Executes a SAS statement for observations that meet specific conditions.</td>
<td></td>
</tr>
<tr>
<td>label: Statement (p. 269)</td>
<td>Identifies a statement that is referred to by another statement.</td>
<td></td>
</tr>
<tr>
<td>LEAVE Statement (p. 270)</td>
<td>Stops processing the current loop and resumes with the next statement in the sequence.</td>
<td></td>
</tr>
<tr>
<td>LINK Statement (p. 293)</td>
<td>Directs program execution immediately to the statement label that is specified and, if followed by a RETURN statement, returns execution to the statement that follows the LINK statement.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>RETURN Statement (p. 382)</td>
<td>Stops executing statements at the current point in the DATA step and returns to a predetermined point in the step.</td>
</tr>
<tr>
<td></td>
<td>SELECT Statement (p. 391)</td>
<td>Executes one of several statements or groups of statements.</td>
</tr>
<tr>
<td>File-Handling</td>
<td>BY Statement (p. 31)</td>
<td>Controls the operation of a SET, MERGE, MODIFY, or UPDATE statement in the DATA step and sets up special grouping variables.</td>
</tr>
<tr>
<td></td>
<td>CARDS Statement (p. 37)</td>
<td>Specifies that data lines follow.</td>
</tr>
<tr>
<td></td>
<td>CARDS4 Statement (p. 37)</td>
<td>Specifies that data lines that contain semicolons follow.</td>
</tr>
<tr>
<td></td>
<td>DATA Statement (p. 44)</td>
<td>Begins a DATA step and provides names for any output SAS data sets, views, or programs.</td>
</tr>
<tr>
<td></td>
<td>DATALINES Statement (p. 53)</td>
<td>Specifies that data lines follow.</td>
</tr>
<tr>
<td></td>
<td>DATALINES4 Statement (p. 54)</td>
<td>Indicates that data lines that contain semicolons follow.</td>
</tr>
<tr>
<td></td>
<td>FILE Statement (p. 73)</td>
<td>Specifies the current output file for PUT statements.</td>
</tr>
<tr>
<td></td>
<td>INFILE Statement (p. 198)</td>
<td>Specifies an external file to read with an INPUT statement.</td>
</tr>
<tr>
<td></td>
<td>INPUT Statement (p. 232)</td>
<td>Describes the arrangement of values in the input data record and assigns input values to the corresponding SAS variables.</td>
</tr>
<tr>
<td></td>
<td>INPUT Statement, Column (p. 248)</td>
<td>Reads input values from specified columns and assigns them to the corresponding SAS variables.</td>
</tr>
<tr>
<td></td>
<td>INPUT Statement, Formatted (p. 251)</td>
<td>Reads input values with specified informats and assigns them to the corresponding SAS variables.</td>
</tr>
<tr>
<td></td>
<td>INPUT Statement, List (p. 255)</td>
<td>Scans the input data record for input values and assigns them to the corresponding SAS variables.</td>
</tr>
<tr>
<td></td>
<td>INPUT Statement, Named (p. 262)</td>
<td>Reads data values that appear after a variable name that is followed by an equal sign and assigns them to corresponding SAS variables.</td>
</tr>
<tr>
<td></td>
<td>MERGE Statement (p. 304)</td>
<td>Joins observations from two or more SAS data sets into a single observation.</td>
</tr>
<tr>
<td></td>
<td>MODIFY Statement (p. 309)</td>
<td>Replaces, deletes, and appends observations in an existing SAS data set in place but does not create an additional copy.</td>
</tr>
<tr>
<td></td>
<td>PUT Statement (p. 335)</td>
<td>Writes lines to the SAS log, to the SAS output window, or to an external location that is specified in the most recent FILE statement.</td>
</tr>
<tr>
<td></td>
<td>PUT Statement, Column (p. 354)</td>
<td>Writes variable values in the specified columns in the output line.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PUT Statement, Formatted (p. 356)</td>
<td>Writes variable values with the specified format in the output line.</td>
<td></td>
</tr>
<tr>
<td>PUT Statement, List (p. 360)</td>
<td>Writes variable values and the specified character strings in the output line.</td>
<td></td>
</tr>
<tr>
<td>PUT Statement, Named (p. 365)</td>
<td>Writes variable values after the variable name and an equal sign.</td>
<td></td>
</tr>
<tr>
<td>SET Statement (p. 394)</td>
<td>Reads an observation from one or more SAS data sets.</td>
<td></td>
</tr>
<tr>
<td>UPDATE Statement (p. 422)</td>
<td>Updates a master file by applying transactions.</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>ARRAY Statement (p. 18)</td>
<td>Defines the elements of an array.</td>
</tr>
<tr>
<td></td>
<td>Array Reference Statement (p. 23)</td>
<td>Describes the elements in an array to be processed.</td>
</tr>
<tr>
<td></td>
<td>ATTRIB Statement (p. 27)</td>
<td>Associates a format, informat, label, and length with one or more variables.</td>
</tr>
<tr>
<td></td>
<td>DROP Statement (p. 68)</td>
<td>Excludes variables from output SAS data sets.</td>
</tr>
<tr>
<td></td>
<td>FORMAT Statement (p. 182)</td>
<td>Associates formats with variables.</td>
</tr>
<tr>
<td></td>
<td>INFORMAT Statement (p. 228)</td>
<td>Associates informats with variables.</td>
</tr>
<tr>
<td></td>
<td>KEEP Statement (p. 266)</td>
<td>Specifies the variables to include in output SAS data sets.</td>
</tr>
<tr>
<td></td>
<td>LABEL Statement (p. 267)</td>
<td>Assigns descriptive labels to variables.</td>
</tr>
<tr>
<td></td>
<td>LENGTH Statement (p. 271)</td>
<td>Specifies the number of bytes for storing variables.</td>
</tr>
<tr>
<td></td>
<td>RENAME Statement (p. 372)</td>
<td>Specifies new names for variables in output SAS data sets.</td>
</tr>
<tr>
<td></td>
<td>RETAIN Statement (p. 378)</td>
<td>Causes a variable that is created by an INPUT or assignment statement to retain its value from one iteration of the DATA step to the next.</td>
</tr>
<tr>
<td>Window Display</td>
<td>DISPLAY Statement (p. 57)</td>
<td>Displays a window that is created with the WINDOW statement.</td>
</tr>
<tr>
<td></td>
<td>WINDOW Statement (p. 434)</td>
<td>Creates customized windows for your applications.</td>
</tr>
</tbody>
</table>

### Global Statements by Category

The following table lists and describes SAS global statements, organized by function into eight categories:
### Table 2.2 Global Statements by Category

<table>
<thead>
<tr>
<th>Statements Category</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>signal end of data lines or act as a placeholder.</td>
</tr>
<tr>
<td>Data Access</td>
<td>associate reference names with SAS libraries, SAS catalogs, external files and output devices, and access remote files.</td>
</tr>
<tr>
<td>Information</td>
<td>give SAS additional information about the program data vector.</td>
</tr>
<tr>
<td>Log Control</td>
<td>alter the appearance of the SAS log.</td>
</tr>
<tr>
<td>Operating Environment</td>
<td>access the operating environment directly.</td>
</tr>
<tr>
<td>Output Control</td>
<td>add titles and footnotes to your SAS output; deliver output in a variety of formats.</td>
</tr>
<tr>
<td>Program Control</td>
<td>govern how SAS processes your SAS program.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Null Statement (p. 329)</td>
<td>Signals the end of data lines or acts as a placeholder.</td>
</tr>
<tr>
<td>Data Access</td>
<td>CATNAME Statement (p. 38)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Statement (p. 94)</td>
<td>Associates a SAS fileref with an external file or an output device, disassociates a fileref and external file, or lists attributes of external files.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Statement, CATALOG Access Method (p. 103)</td>
<td>Enables you to reference a SAS catalog as an external file.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Statement, CLIPBOARD Access Method (p. 106)</td>
<td>Enables you to read text data from and write text data to the clipboard on the host computer.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Statement, DATAURL Access Method (p. 108)</td>
<td>Enables you to read data from user-specified text.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Statement, EMAIL (SMTP) Access Method (p. 111)</td>
<td>Enables you to send electronic mail programmatically from SAS using the SMTP (Simple Mail Transfer Protocol) email interface.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FILENAME Statement, FTP Access Method (p. 126)</td>
<td>Enables you to access remote files by using the FTP protocol.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, Hadoop Access Method (p. 141)</td>
<td>Enables you to access files on a Hadoop Distributed File System (HDFS) whose location is specified in a configuration file.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, SFTP Access Method (p. 148)</td>
<td>Enables you to access remote files by using the SFTP protocol.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, SOCKET Access Method (p. 155)</td>
<td>Enables you to read from or write to a TCP/IP socket.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, URL Access Method (p. 160)</td>
<td>Enables you to access remote files by using the URL access method.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, WebDAV Access Method (p. 166)</td>
<td>Enables you to access remote files by using the WebDAV protocol.</td>
<td></td>
</tr>
<tr>
<td>FILENAME Statement, ZIP Access Method (p. 174)</td>
<td>Enables you to access ZIP files.</td>
<td></td>
</tr>
<tr>
<td>LIBNAME Statement (p. 274)</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>LIBNAME Statement for the JMP Engine (p. 288)</td>
<td>Associates a libref with a JMP data table and enables you to read and write JMP data tables.</td>
<td></td>
</tr>
<tr>
<td>LIBNAME Statement for WebDAV Server Access (p. 289)</td>
<td>Associates a libref with a SAS library and enables access to a WebDAV (Web-based Distributed Authoring And Versioning) server.</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>MISSING Statement (p. 308)</td>
<td>Assigns characters in your input data to represent special missing values for numeric data.</td>
</tr>
<tr>
<td>Log Control</td>
<td>Comment Statement (p. 41)</td>
<td>Specifies the purpose of the statement or program.</td>
</tr>
<tr>
<td></td>
<td>PAGE Statement (p. 335)</td>
<td>Skips to a new page in the SAS log.</td>
</tr>
<tr>
<td></td>
<td>RESETLINE Statement (p. 376)</td>
<td>Restarts the program line numbers in the SAS log to 1.</td>
</tr>
<tr>
<td></td>
<td>SKIP Statement (p. 408)</td>
<td>Creates a blank line in the SAS log.</td>
</tr>
<tr>
<td>Operating Environment</td>
<td>X Statement (p. 446)</td>
<td>Issues an operating-environment command from within a SAS session.</td>
</tr>
<tr>
<td>Output Control</td>
<td>FOOTNOTE Statement (p. 178)</td>
<td>Writes up to 10 lines of text at the bottom of the procedure or DATA step output.</td>
</tr>
<tr>
<td></td>
<td>TITLE Statement (p. 412)</td>
<td>Specifies title lines for SAS output.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Program Control</td>
<td>CHECKPOINT</td>
<td>Indicates to execute the DATA step or PROC step that immediately follows without considering the checkpoint-restart data.</td>
</tr>
<tr>
<td></td>
<td>EXECUTE_ALWAYS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Statement (p. 41)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DM Statement (p. 59)</td>
<td>Submits SAS Program Editor, Log, Procedure Output or text editor commands as SAS statements.</td>
</tr>
<tr>
<td></td>
<td>ENDSAS Statement (p. 71)</td>
<td>Terminates a SAS job or session after the current DATA or PROC step executes.</td>
</tr>
<tr>
<td></td>
<td>%INCLUDE Statement (p. 191)</td>
<td>Brings a SAS programming statement, data lines, or both, into a current SAS program.</td>
</tr>
<tr>
<td></td>
<td>%LIST Statement (p. 297)</td>
<td>Displays lines that are entered in the current session.</td>
</tr>
<tr>
<td></td>
<td>LOCK Statement (p. 298)</td>
<td>Acquires, lists, or releases an exclusive lock on an existing SAS file.</td>
</tr>
<tr>
<td></td>
<td>OPTIONS Statement (p. 331)</td>
<td>Specifies or changes the value of one or more SAS system options.</td>
</tr>
<tr>
<td></td>
<td>RUN Statement (p. 383)</td>
<td>Executes the previously entered SAS statements.</td>
</tr>
<tr>
<td></td>
<td>%RUN Statement (p. 384)</td>
<td>Ends source statements following a %INCLUDE * statement.</td>
</tr>
<tr>
<td></td>
<td>SASFILE Statement (p. 385)</td>
<td>Opens a SAS data set and allocates enough buffers to hold the entire file in memory.</td>
</tr>
<tr>
<td></td>
<td>SYSECHO Statement (p. 411)</td>
<td>Sends a global statement complete event and passes a text string back to the IOM client.</td>
</tr>
</tbody>
</table>

## Dictionary

### ABORT Statement

Stops executing the current DATA step, SAS job, or SAS session.

- **Valid in:** DATA step
- **Category:** Action
- **Type:** Executable
- **See:** ABORT Statement in the SAS Companion for Windows, SAS Companion for z/OS, and SAS Companion for UNIX Environments

### Syntax

```
ABORT <ABEND | CANCEL <FILE> | RETURN > <n> <NOLIST>;
```
Without Arguments
If you specify no argument, the ABORT statement produces these results under the following methods of operation:

batch mode and noninteractive mode
- stops processing the current DATA step and writes an error message to the SAS log. Data sets can contain an incomplete number of observations or no observations, depending on when SAS encountered the ABORT statement.
- sets the OBS= system option to 0.
- continues limited processing of the remainder of the SAS job, including executing macro statements, executing system options statements, and syntax checking of program statements.
- creates output data sets for subsequent DATA and PROC steps with no observations.

windowing environment
- stops processing the current DATA step
- creates a data set that contains the observations that are processed before the ABORT statement is encountered
- prints a message to the log that an ABORT statement terminated the DATA step
- continues processing any DATA or PROC steps that follow the ABORT statement

interactive line mode
- stops processing the current DATA step. Any further DATA steps or procedures execute normally.

Arguments

ABEND
causes abnormal termination of the current SAS job or session. Results depend on the method of operation:
- batch mode and noninteractive mode.
  - stops processing immediately.
  - sends an error message to the SAS log that states that execution was terminated by the ABEND option of the ABORT statement.
  - does not execute any subsequent statements or check syntax.
  - returns control to the operating environment; further action is based on how your operating environment and your site treat jobs that end abnormally.
- windowing environment and interactive line mode.
  - causes your windowing environment and interactive line mode to stop processing immediately and return you to your operating environment.

CANCEL <FILE>
causes the execution of the submitted statements to be canceled. Results depend on the method of operation:
- batch mode and noninteractive mode.
  - the entire SAS program and SAS system are terminated.
  - an error message is written to the SAS log.
windowing environment and interactive line mode.
  - clears only the current submitted program.
  - other subsequent submitted programs are not affected.
  - an error message is written to the SAS log.
workspace server and stored process server.
  - clears only the currently submitted program.
  - other subsequent submit calls are not affected.
  - an error message is written to the SAS log.
SAS IntrNet application server.
  - creates a separate execution for each request and submits the request code. A CANCEL argument in the request code clears the current submitted code but does not terminate the execution or the SAS session.

FILE
  when coded as an option to the CANCEL argument in an autoexec file or in a %INCLUDE file, causes only the contents of the autoexec file or %INCLUDE file to be cleared by the ABORT statement. Other submitted source statements will be executed after the autoexec or %INCLUDE file.

Restriction
The CANCEL argument cannot be submitted using SAS/SHARE, SAS/CONNECT, or SAS/AF.

Note
When the ABORT CANCEL FILE option is executed within a %INCLUDE file, all open macros are closed and execution resumes at the next source line of code.

RETURN
causes the immediate normal termination of the current SAS job or session. Results depend on the method of operation:
  - batch mode and noninteractive mode
    - stops processing immediately
    - sends an error message to the SAS log stating that execution was terminated by the RETURN option in the ABORT statement
    - does not execute any subsequent statements or check syntax
    - returns control to your operating environment with a condition code indicating an error
  - windowing environment
    - causes your windowing environment and interactive line mode to stop processing immediately and return you to your operating environment.

\( n \)
is an integer value that enables you to specify a condition code:
  - when used with the CANCEL argument, the value is placed in the SYSINFO automatic macro variable
  - when not used with the CANCEL argument, the error code that is returned by SAS is ERROR. The value of ERROR depends on the operating system. The condition code \( n \) is returned to the operating system as the final SAS system exit code.
NOLIST

suppresses the output of all variables to the SAS log.

Requirement  NOLIST must be the last option in the ABORT statement.

Details

The ABORT statement causes SAS to stop processing the current DATA step. What happens next depends on

• the method that you use to submit your SAS statements
• the arguments that you use with ABORT
• your operating environment.

The ABORT statement usually appears in a clause of an IF-THEN statement or a SELECT statement that is designed to stop processing when an error condition occurs.

Note: The return code generated by the ABORT statement is ignored by SAS if the system option ERRORABEND is in effect.

Note: When you execute an ABORT statement in a DATA step, SAS does not use data sets that were created in the step to replace existing data sets with the same name.

Operating Environment Information

The only difference between the ABEND and RETURN options is that with ABEND further action is based on how your operating environment and site treat jobs that end abnormally. RETURN simply returns a condition code that indicates an error.

Comparisons

• When you use the SAS windowing environment or interactive line mode, the ABORT statement and the STOP statement both stop processing. The ABORT statement sets the value of the automatic variable _ERROR_ to 1, and the STOP statement does not.
• In batch or noninteractive mode, the ABORT and STOP statements also have different effects. Both stop processing, but only ABORT sets the value of the automatic variable _ERROR_ to 1. Use the STOP statement, therefore, when you want to stop only the current DATA step and continue processing with the next step.

Example: Stopping Execution of SAS

This example uses the ABORT statement as part of an IF-THEN statement to stop execution of SAS when it encounters a data value that would otherwise cause a division-by-zero condition.

```
if volume=0 then abort 255;
  density=mass/volume;
```

The 255 value causes SAS to return the condition code 255 to the operating environment when the ABORT statement executes.

See Also

Statements:

• “STOP Statement” on page 409
ARRAY Statement

Defines the elements of an array.

Valid in: DATA step
Category: Information
Type: Declarative

Syntax

ARRAY array-name { subscript } <$> <length>
array-elements> <(initial-value-list)> ;

Arguments

array-name
specifies the name of the array.

Restriction

Array-name must be a SAS name that is not the name of a SAS variable in the same DATA step.

CAUTION

Using the name of a SAS function as an array name can cause unpredictable results. If you inadvertently use a function name as the name of the array, SAS treats parenthetical references that involve the name as array references, not function references, for the duration of the DATA step. A warning message is written to the SAS log.

{subscript}
describes the number and arrangement of elements in the array by using an asterisk, a number, or a range of numbers. Subscript has one of these forms:

{dimension-size(s)}
specifies the number of elements in each dimension of the array. Dimension-size is a numeric representation of either the number of elements in a one-dimensional array or the number of elements in each dimension of a multidimensional array.

Tip
You can enclose the subscript in braces ({}), brackets ([ ] ) or parentheses ( ( )).

Examples

This ARRAY statement defines a one dimensional array that is named SIMPLE. The SIMPLE array groups together three variables that are named RED, GREEN, and YELLOW:

array simple{3} red green yellow;

An array with more than one dimension is known as a multidimensional array. You can have any number of dimensions in a multidimensional array. For example, a two-dimensional array provides row and column arrangement of array elements. SAS places variables into a two-dimensional array by filling all rows in order, beginning at the upper left corner of the array (known as row-major order). This statement defines a two-dimensional array with five rows and three columns:
array x{5,3} score1-score15;

\{<lower >:upper<, ...<lower >: upper>\}
are the bounds of each dimension of an array, where lower is the lower bound of that dimension and upper is the upper bound.

**Range**
In most explicit arrays, the subscript in each dimension of the array ranges from 1 to \(n\), where \(n\) is the number of elements in that dimension.

**Tips**
For most arrays, 1 is a convenient lower bound. Thus, you do not need to specify the lower and upper bounds. However, specifying both bounds is useful when the array dimensions have a convenient beginning point other than 1.

To reduce the computational time that is needed for subscript evaluation, specify a lower bound of 0.

**Examples**
In the following example, the value of each dimension is by default the upper bound of that dimension.

```sas
array x{5,3} score1-score15;
```

As an alternative, the following ARRAY statement is a longhand version of the previous example:

```sas
array x{1:5,1:3} score1-score15;
```

\{*\}
specifies that SAS is to determine the subscript by counting the variables in the array. When you specify the asterisk, also include array-elements.

**Restriction**
You cannot use the asterisk with _TEMPORARY_ arrays or when you define a multidimensional array.

\$ specifies that the elements in the array are character elements.

**Tip**
The dollar sign is not necessary if the elements have been previously defined as character elements.

**length**
specifies the length of elements in the array that have not been previously assigned a length.

**array-elements**
specifies the names of the elements that make up the array. Array-elements must be either all numeric or all character, and they can be listed in any order. The elements can be

**variables**
lists variable names.

**Range**
The names must be either variables that you define in the ARRAY statement or variables that SAS creates by concatenating the array name and a number. For example, when the subscript is a number (not the asterisk), you do not need to name each variable in the array. Instead, SAS creates variable names by concatenating the array name and the numbers 1, 2, 3, … \(n\).
Restriction  If you use _ALL_, all the previously defined variables must be of the same type.

Tips  These SAS variable lists enable you to reference variables that have been previously defined in the same DATA step:

_NUMERIC_ specifies all numeric variables.

_CHARACTER_ specifies all character variables.

_ALL_ specifies all variables.

Example  “Example 1: Defining Arrays” on page 21

_TEMPORARY_ creates a list of temporary data elements.

Range  Temporary data elements can be numeric or character.

Tips  Temporary data elements behave like DATA step variables with these exceptions:

They do not have names. Refer to temporary data elements by the array name and dimension.

They do not appear in the output data set.

You cannot use the special subscript asterisk (*) to refer to all the elements.

Temporary data element values are always automatically retained, rather than being reset to missing at the beginning of the next iteration of the DATA step.

Arrays of temporary elements are useful when the only purpose for creating an array is to perform a calculation. To preserve the result of the calculation, assign it to a variable. You can improve performance time by using temporary data elements.

(initial-value-list) gives initial values for the corresponding elements in the array. The values for elements can be numbers or character strings. You must enclose all character strings in quotation marks. To specify one or more initial values directly, use the following format:

(initial-value(s))

To specify an iteration factor and nested sublists for the initial values, use the following format:

<constant-iter-value*> <(constant value | constant-sublist<)>
Elements and values are matched by position. If there are more array elements than initial values, the remaining array elements receive missing values and SAS issues a warning.

You can separate the values in the initial value list with either a comma or a blank space.

You can also use a shorthand notation for specifying a range of sequential integers. The increment is always +1.

If you have not previously specified the attributes of the array elements (such as length or type), the attributes of any initial values that you specify are automatically assigned to the corresponding array element. Initial values are retained until a new value is assigned to the array element.

When any (or all) elements are assigned initial values, all elements behave as if they were named in a RETAIN statement.

Example
The following examples show how to use the iteration factor and nested sublists. All of these ARRAY statements contain the same initial value list:

ARRAY x{10} x1-x10 (10*5);
ARRAY x{10} x1-x10 (5*(5 5)));
ARRAY x{10} x1-x10 (5 5 3*(5 5) 5 5);
ARRAY x{10} x1-x10 (2*(5 5) 5 5 2*(5 5));
ARRAY x{10} x1-x10 (2*(5 2*(5 5)));

Details
The ARRAY statement defines a set of elements that you plan to process as a group. You refer to elements of the array by the array name and subscript. Because you usually want to process more than one element in an array, arrays are often referenced within DO groups.

Comparisons
• Arrays in the SAS language are different from arrays in many other languages. A SAS array is simply a convenient way of temporarily identifying a group of variables. It is not a data structure, and array-name is not a variable.
• An ARRAY statement defines an array. An array reference uses an array element in a program statement.

Examples

Example 1: Defining Arrays
• array rain {5} janr febr marr aprr mayr;
• array days{7} d1-d7;
Example 2: Assigning Initial Numeric Values
- array test{4} t1 t2 t3 t4 (90 80 70 70);
- array test{4} t1-t4 (90 80 2*70);
- array test{4} _TEMPORARY_ (90 80 70 70);

Example 3: Defining Initial Character Values
- array test2{*} $ a1 a2 a3 ('a','b','c');

Example 4: Defining More Advanced Arrays
- array new{2:5} green jacobs denato fetzer;
- array x{5,3} score1-score15;
- array test{3:4,3:7} test1-test10;
- array temp{0:999} _TEMPORARY_;
- array x{10} (2*1:5);

Example 5: Creating a Range of Variable Names That Have Leading Zeros
The following example shows that you can create a range of variable names that have leading zeros. Each variable name has a length of three characters, and the names sort correctly (A01, A02, … A10). Without leading zeros, the variable names would sort in the following order: A1, A10, A2, … A9.

data test (drop=i);
  array a{10} A01-A10;
  do i=1 to 10;
    a{i}=i;
  end;
run;
proc print noobs data=test;
run;

Output 2.1  Array Names That Have Leading Zeros

The SAS System

<table>
<thead>
<tr>
<th>A01</th>
<th>A02</th>
<th>A03</th>
<th>A04</th>
<th>A05</th>
<th>A06</th>
<th>A07</th>
<th>A08</th>
<th>A09</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

See Also
- “Array Processing” in SAS Language Reference: Concepts
Statements:

- “Array Reference Statement” on page 23

---

**Array Reference Statement**

Describes the elements in an array to be processed.

| Valid in: | DATA step |
| Category: | Information |
| Type: | Declarative |

### Syntax

`array-name {subscript};`

### Arguments

- **array-name**
  - is the name of an array that was previously defined with an ARRAY statement in the same DATA step.

- **{subscript}**
  - specifies the subscript. Any of these forms can be used:
    - `{variable-1, …variable-n}`
      - specifies a variable, or variable list that is usually used with DO-loop processing. For each execution of the DO loop, the current value of this variable becomes the subscript of the array element being processed.

### Tips

- You can enclose a subscript in braces `{ }`, brackets `[ ]`, or parentheses `()`.

### Example

“Example 1: Using Iterative DO-Loop Processing” on page 25

- `{*}`
  - forces SAS to treat the elements in the array as a variable list.

### Restriction

- When you define an array that contains temporary array elements, you cannot reference the array elements with an asterisk.

### Tips

- The asterisk can be used with the INPUT and PUT statements, and with some SAS functions.

- This syntax is provided for convenience and is an exception to usual array processing.

### Example

“Example 4: Using the Asterisk References as a Variable List” on page 25

- `expression-1, …expression-n`
  - specifies a SAS expression.

### Range

- The expression must evaluate to a subscript value when the statement that contains the array reference executes. The expression can also be
an integer with a value between the lower and upper bounds of the array, inclusive.

Example

“Example 3: Specifying the Subscript” on page 25

Details

• To refer to an array in a program statement, use an array reference. The ARRAY statement that defines the array must appear in the DATA step before any references to that array. An array definition is only in effect for the duration of the DATA step. If you want to use the same array in several DATA steps, redefine the array in each step.

CAUTION:

Using the name of a SAS function as an array name can cause unpredictable results. If you inadvertently use a function name as the name of the array, SAS treats parenthetical references that involve the name as array references, not function references, for the duration of the DATA step. A warning message is written to the SAS log.

• You can use an array reference anywhere that you can write a SAS expression, including SAS functions and these SAS statements:
  - assignment statement
  - sum statement
  - DO UNTIL(expression)
  - DO WHILE(expression)
  - IF
  - INPUT
  - PUT
  - SELECT
  - WINDOW.

• The DIM function is often used with the iterative DO statement to return the number of elements in a dimension of an array, when the lower bound of the dimension is 1. If you use DIM, you can change the number of array elements without changing the upper bound of the DO statement. For example, because DIM(NEW) returns a value of 4, the following statements process all the elements in the array:

```sas
array new{*} score1-score4;
  do i=1 to dim(new);
    new{i}=new{i}+10;
  end;
```

Comparisons

An ARRAY statement defines an array, whereas an array reference defines the members of the array to process.
Examples

Example 1: Using Iterative DO-Loop Processing
In this example, the statements process each element of the array, using the value of variable I as the subscript on the array references for each iteration of the DO loop. If an array element has a value of 99, the IF-THEN statement changes that value to 100.

```sas
array days{7} d1-d7;
do i=1 to 7;
   if days{i}=99 then days{i}=100;
end;
```

Example 2: Referencing Many Arrays in One Statement
You can refer to more than one array in a single SAS statement. In this example, you create two arrays, DAYS and HOURS. The statements inside the DO loop substitute the current value of variable I to reference each array element in both arrays.

```sas
array days{7} d1-d7;
array hours{7} h1-h7;
do i=1 to 7;
   if days{i}=99 then days{i}=100;
   hours{i}=days{i}*24;
end;
```

Example 3: Specifying the Subscript
In this example, the INPUT statement reads in variables A1, A2, and the third element (A3) of the array named ARR1:

```sas
array arr1{*} a1-a3;
x=1;
input a1 a2 arr1{x+2};
```

Example 4: Using the Asterisk References as a Variable List
- array cost{10} cost1-cost10;
  totcost=sum(of cost {*});
- array days{7} d1-d7;
  input days {*};
- array hours{7} h1-h7;
  put hours {*};

See Also
- “Array Processing” in SAS Language Reference: Concepts

Functions:
- “DIM Function” in SAS Functions and CALL Routines: Reference

Statements:
- “ARRAY Statement” on page 18
- “DO Statement, Iterative” on page 62
Assignment Statement
Evaluates an expression and stores the result in a variable.

Valid in: DATA step
Category: Action
Type: Executable

Syntax

\[
\text{variable} = \text{expression};
\]

Arguments

\text{variable}

names a new or existing variable.

Range \text{variable} can be a variable name, array reference, or SUBSTR function.

Tip Variables that are created by the Assignment statement are not automatically retained.

\text{expression}

is any SAS expression.

Tip \text{expression} can contain the variable that is used on the left side of the equal sign. When a variable appears on both sides of a statement, the original value on the right side is used to evaluate the expression, and the result is stored in the variable on the left side of the equal sign. For more information, see “Expressions” in SAS Language Reference: Concepts.

Details
Assignment statements evaluate the expression on the right side of the equal sign and store the result in the variable that is specified on the left side of the equal sign.

Example: Various Expressions in Assignment Statements
These assignment statements use different types of expressions:

- \text{name}='Amanda Jones';
- \text{WholeName}='Ms. '||\text{name};
- \text{a} = \text{a} + \text{b};

See Also

Statements:
- “Sum Statement” on page 410
ATTRIB Statement

Associates a format, informat, label, and length with one or more variables.

Valid in: DATA step
Category: Information
Type: Declarative
See: ATTRIB Statement under Windows, UNIX, and z/OS

Syntax

ATTRIB variable-list(s) attribute-list(s) ;

Arguments

variable-list(s)

names the variables that you want to associate with the attributes.

Tip List the variables in any form that SAS allows.

attribute-list(s)

specifies one or more attributes to assign to variable-list. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=format

associates a format with variables in variable-list.

Tip The format can be either a standard SAS format or a format that is defined with the FORMAT procedure.

INFORMAT=informat

associates an informat with variables in variable-list.

Tip The informat can be either a standard SAS informat or an informat that is defined with the FORMAT procedure.

LABEL='label'

associates a label with variables in variable-list.

LENGTH=<$>length

specifies the length of variables in variable-list.

Range For character variables, the range is 1 to 32,767 bytes for all operating environments.

Restriction You cannot change the length of a variable using LENGTH= from PROC DATASETS.

Requirement Put a dollar sign ($) in front of the length of character variables.
Operating environment
For numeric variables, the minimum length that you can specify with the LENGTH= specification is 2 bytes in some operating environments and 3 bytes in others.

Tip
Use the ATTRIB statement before the SET statement to change the length of variables in an output data set when you use an existing data set as input.

**TRANSCODE=**YES | NO
specifies whether character variables can be transcoded. Use TRANSCODE=NO to suppress transcoding.

Default
YES

Restrictions
The TRANSCODE=NO attribute is not supported by some SAS Workspace Server clients. In SAS 9.2, if the attribute is not supported, variable values with TRANSCODE=NO are replaced (masked) with asterisks (*). Prior to SAS 9.2, variables with TRANSCODE=NO were transcoded.

Prior releases of SAS cannot access a SAS 9.1 data set that contains a variable with a TRANSCODE=NO attribute.

Transcode suppression is not supported by the V6TAPE engine.

Interactions
You can use the “**VTRANSCODE Function**” in *SAS National Language Support (NLS): Reference Guide* and “**VTRANSCODEX Function**” in *SAS National Language Support (NLS): Reference Guide* to return a value that indicates whether transcoding is on or off for a character variable.

If the TRANSCODE= attribute is set to NO for any character variable in a data set, then PROC CONTENTS prints a transcode column that contains the TRANSCODE= value for each variable in the data set. If all variables in the data set are set to the default TRANSCODE= value (YES), then no transcode column prints.

See

Details

**The Basics**
Using the ATTRIB statement in the DATA step permanently associates attributes with variables by changing the descriptor information of the SAS data set that contains the variables.

You can use ATTRIB in a PROC step, but the rules are different.

**How SAS Treats Variables When You Assign Informats with the INFORMAT= Option in the ATTRIB Statement**
Informats that are associated with variables by using the INFORMAT= option in the ATTRIB statement behave like informats that are used with modified list input. SAS reads the variables by using the scanning feature of list input, but applies the informat. In modified list input, SAS does the following:
- does not use the value of w in an informat to specify column positions or input field widths in an external file
- uses the value of w in an informat to specify the length of previously undefined character variables
- ignores the value of w in numeric informats
- uses the value of d in an informat in the same way it usually does for numeric informats
- treats blanks that are embedded as input data as delimiters unless you change their status with the DLM= or DLMSTR= option specification in an INFILE statement.

If you have coded the INPUT statement to use another style of input, such as formatted input or column input, that style of input is not used when you use the INFORMAT= option in the ATTRIB statement.

**How SAS Treats Transcoded Variables When You Use the SET and MERGE Statements**

When you use the SET or MERGE statement to create a data set from several data sets, SAS makes the TRANSCODE= attribute of the variable in the output data set equal to the TRANSCODE= value of the variable in the first data set. See “Example 2: Using the SET Statement with Transcoded Variables” on page 30 and “Example 3: Using the MERGE Statement with Transcoded Variables” on page 30.

*Note:* The TRANSCODE= attribute is set when the variable is first seen on an input data set or in an ATTRIB TRANSCODE= statement. If a SET or MERGE statement comes before an ATTRIB TRANSCODE= statement and the TRANSCODE= attribute contradicts the SET statement, a warning will occur.

**Comparisons**

You can use either an ATTRIB statement or an individual attribute statement such as FORMAT, INFORMAT, LABEL, and LENGTH to change an attribute that is associated with a variable.

**Examples**

**Example 1: Examples of ATTRIB Statements with Varying Numbers of Variables and Attributes**

Here are examples of ATTRIB statements that contain different numbers of variables and attributes:

- single variable and single attribute:
  ```
  attrib cost length=4;
  ```

- single variable with multiple attributes:
  ```
  attrib saleday informat=mmddyy.
  format=worddate.;
  ```

- multiple variables with the same multiple attributes:
  ```
  attrib x y length=$4 label='TEST VARIABLE';
  ```

- multiple variables with different multiple attributes:
  ```
  attrib x length=$4 label='TEST VARIABLE'
  y length=$2 label='RESPONSE';
  ```
• variable list with single attribute:
  
```sas
attrib month1-month12
  label='MONTHLY SALES';
```

**Example 2: Using the SET Statement with Transcoded Variables**

In this example, which uses the SET statement, the variable Z's TRANSCODE= attribute in data set A is NO because B is the first data set and Z's TRANSCODE= attribute in data set B is NO.

```sas
data b;
  length z $4;
  z = 'ice';
  attrib z transcode = no;
data c;
  length z $4;
  z = 'snow';
  attrib z transcode = yes;
data a;
  set b;
  set c;
  /* Check transcode setting for variable Z */
  rc1 = vtranscode(z);
  put rc1=;
run;
```

**Example 3: Using the MERGE Statement with Transcoded Variables**

In this example, which uses the MERGE statement, the variable Z's TRANSCODE= attribute in data set A is YES because C is the first data set and Z's TRANSCODE= attribute in data set C is YES.

```sas
data b;
  length z $4;
  z = 'ice';
  attrib z transcode = no;
data c;
  length z $4;
  z = 'snow';
  attrib z transcode = yes;
data a;
  merge c b;
  /* Check transcode setting for variable Z */
  rc1 = vtranscode(z);
  put rc1=;
run;
```

**See Also**

• “How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?” in *SAS Language Reference: Concepts*

**Functions:**

• “VTRANSCODE Function” in *SAS National Language Support (NLS): Reference Guide*
• “VTRANSCODEX Function” in *SAS National Language Support (NLS): Reference Guide*

**Statements:**

• “FORMAT Statement” on page 182
• “INFORMAT Statement” on page 228
• “LABEL Statement” on page 267
• “LENGTH Statement” on page 271

**BY Statement**

Controls the operation of a SET, MERGE, MODIFY, or UPDATE statement in the DATA step and sets up special grouping variables.

Valid in: DATA step or PROC step

Category: File-Handling

Type: Declarative

**Syntax**

```
BY <DESCENDING> variable-1
<...<DESCENDING> variable-n> <NOTSORTED> <GROUPFORMAT>;
```

**Arguments**

**DESCENDING**

specifies that the data sets are sorted in descending order by the variable that is specified. DESCENDING means largest to smallest numerically, or reverse alphabetical for character variables.

**Restriction**

You cannot use the DESCENDING option with data sets that are indexed because indexes are always stored in ascending order.

**Example**

“Example 2: Specifying Sort Order” on page 34

**GROUPFORMAT**

uses the formatted values, instead of the internal values, of the BY variables to determine where BY groups begin and end, and therefore how FIRST. variable and LAST. variable are assigned. Although the GROUPFORMAT option can appear anywhere in the BY statement, the option applies to all variables in the BY statement.

**Restrictions**

You must sort the observations in a data set based on the value of the BY variables before using the GROUPFORMAT option in the BY statement.

You can use the GROUPFORMAT option in a BY statement only in a DATA step.
**Interaction**

If you also use the NOTSORTED option, you can group the observations in a data set by the formatted value of the BY variables without requiring that the data set be sorted or indexed.

| Note | BY-group processing in the DATA step using the GROUPFORMAT option is the same as BY-group processing with formatted values in SAS procedures. |
| Tips | Using the GROUPFORMAT option is useful when you define your own formats to display data that is grouped. Using the GROUPFORMAT option in the DATA step ensures that BY groups that you use to create a data set match the BY groups in PROC steps that report grouped, formatted data. |
| See | “BY-Group Processing in the DATA Step” in SAS Language Reference: Concepts |
| Example | “Example 4: Grouping Observations By Using Formatted Values” on page 35 |

**variable**

names each variable by which the data set is sorted or indexed. These variables are referred to as BY variables for the current DATA or PROC step.

| Requirement | If you designate a name literal as the BY variable in BY-group processing and you want to refer to the corresponding FIRST. or LAST. temporary variables, you must include the FIRST. or LAST. portion of the two-level variable name within single quotation marks. For example: |
| Tip | The data set can be sorted or indexed by more than one variable. |
| Examples | “Example 1: Specifying One or More BY Variables” on page 34 |
| | “Example 2: Specifying Sort Order” on page 34 |
| | “Example 3: BY-Group Processing with Nonsorted Data” on page 34 |
| | “Example 4: Grouping Observations By Using Formatted Values” on page 35 |

**NOTSORTED**

specifies that observations with the same BY value are grouped together but are not necessarily sorted in alphabetical or numeric order.

| Restriction | You cannot use the NOTSORTED option with the MERGE and UPDATE statements. |
| Tips | The NOTSORTED option can appear anywhere in the BY statement. |
Using the NOTSORTED option is useful if you have data that falls into other logical groupings such as chronological order or categories.

Example

“Example 3: BY-Group Processing with Nonsorted Data” on page 34

Details

**How SAS Identifies the Beginning and End of a BY Group**

SAS identifies the beginning and end of a BY group by creating two temporary variables for each BY variable: FIRST.variable and LAST.variable. The value of these variables is either 0 or 1. SAS sets the value of FIRST.variable to 1 when it reads the first observation in a BY group, and sets the value of LAST.variable to 1 when it reads the last observation in a BY group. These temporary variables are available for DATA step programming but are not added to the output data set.

For a complete explanation of how SAS processes grouped data and of how to prepare your data, see “BY-Group Processing in the DATA Step” in *SAS Language Reference: Concepts*.

**In a DATA Step**

The BY statement applies only to the SET, MERGE, MODIFY, or UPDATE statement that precedes it in the DATA step, and only one BY statement can accompany each of these statements in a DATA step.

The data sets that are listed in the SET, MERGE, or UPDATE statements must be sorted by the values of the variables that are listed in the BY statement or have an appropriate index. As a default, SAS expects the data sets to be arranged in ascending numeric order or in alphabetical order. The observations can be arranged by one of the following methods:

- sort the data set
- create an index for the variables
- input the observations in order.

*Note:* MODIFY does not require sorted data, but sorting can improve performance.

*Note:* The BY statement honors the linguistic collation of data that is sorted by using the SORT procedure with the SORTSEQ=LINGUISTIC option.

For more information, see “How to Prepare Your Data Sets” in *SAS Language Reference: Concepts*.

**In a PROC Step**

You can specify the BY statement with some SAS procedures to modify their action. For more information, see the individual procedure in the *Base SAS Procedures Guide* for a discussion of how the BY statement affects processing for SAS procedures.

**With SAS Views**

If you create a DATA step view by reading from a DBMS and the SET, MERGE, UPDATE, or MODIFY statement is followed by a BY statement, the BY statement might cause the DBMS to sort the data in order to return the data in sorted order. Sorting the data could increase execution time.
Processing BY Groups
SAS assigns the following values to FIRST.\textit{variable} and LAST.\textit{variable}:

- FIRST.\textit{variable} has a value of 1 under the following conditions:
  - when the current observation is the first observation that is read from the data set.
  - when you do not use the GROUPFORMAT option and the internal value of the variable in the current observation differs from the internal value in the previous observation.

  If you use the GROUPFORMAT option, FIRST.\textit{variable} has a value of 1 when the formatted value of the variable in the current observation differs from the formatted value in the previous observation.

- FIRST.\textit{variable} has a value of 1 for any preceding variable in the BY statement.

  In all other cases, FIRST.\textit{variable} has a value of 0.

- LAST.\textit{variable} has a value of 1 under the following conditions:
  - when the current observation is the last observation that is read from the data set.
  - when you use the GROUPFORMAT option and the internal value of the variable in the current observation differs from the internal value in the next observation.

  If you use the GROUPFORMAT option, LAST.\textit{variable} has a value of 1 when the formatted value of the variable in the current observation differs from the formatted value in the next observation.

- LAST.\textit{variable} has a value of 1 for any preceding variable in the BY statement.

  In all other cases, LAST.\textit{variable} has a value of 0.

Examples

Example 1: Specifying One or More BY Variables

- Observations are in ascending order of the variable DEPT:

  \texttt{by dept;}

- Observations are in alphabetical (ascending) order by CITY and, within each value of CITY, in ascending order by ZIPCODE:

  \texttt{by city zipcode;}

Example 2: Specifying Sort Order

- Observations are in ascending order of SALESREP and, within each SALESREP value, in descending order of the values of JANSALES:

  \texttt{by salesrep descending jansales;}

- Observations are in descending order of BEDROOMS, and, within each value of BEDROOMS, in descending order of PRICE:

  \texttt{by descending bedrooms descending price;}

Example 3: BY-Group Processing with Nonsorted Data

Observations are ordered by the name of the month in which the expenses were accrued:

\texttt{by month notsorted;}

Example 4: Grouping Observations By Using Formatted Values

The following example illustrates the use of the GROUPFORMAT option.

```sas
proc format;
   value range
      low -55 = 'Under 55'
      55-60  = '55 to 60'
      60-65  = '60 to 65'
      65-70  = '65 to 70'
      other  = 'Over 70';
run;
proc sort data=sashelp.class out=sorted_class;
   by height;
run;
data _null_
   format height range.;
   set sorted_class;
   by height groupformat;
   if first.height then
      put 'Shortest in ' height ' measures ' height:best12.;
run;
```

SAS writes the following output to the log:

```
Shortest in Under 55 measures 51.3
Shortest in 55 to 60 measures 56.3
Shortest in 60 to 65 measures 62.5
Shortest in 65 to 70 measures 65.3
Shortest in Over 70 measures 72
```

Example 5: Combining Multiple Observations and Grouping Them Based on One BY Value

The following example shows how to use FIRST.variable and LAST.variable with BY-group processing.

```sas
data Inventory;
   length RecordID 8 Invoice $ 30 ItemLine $ 50;
   infile datalines;
   input RecordID Invoice ItemLine &;
   drop RecordID;
   datalines;
A74  A5296  Highlighters
A75  A5296  Lot # 7603
A76  A5296  Yellow Blue Green
A77  A5296  24 per box
A78  A5297  Paper Clips
A79  A5297  Lot # 7423
A80  A5297  Small Medium Large
A81  A5298  Gluestick
A82  A5298  Lot # 4422
A83  A5298  New item
A84  A5299  Rubber bands
A85  A5299  Lot # 7892
A86  A5299  Wide width, Narrow width
A87  A5299  1000 per box
;
data combined;
```
array Line{4} $ 60 ;
retain Linel-Line4;
keep Invoice Linel-Line4;
set Inventory;
by Invoice;

if first.Invoice then do;
call missing(of Linel-Line4);
records = 0;
end;

records + 1;
Line[records]=ItemLine;

if last.Invoice then output;
run;
proc print data=combined;
title 'Office Supply Inventory';
run;

Output 2.2  Output from Combining Multiple Observations

<table>
<thead>
<tr>
<th>Obs</th>
<th>Line1</th>
<th>Line2</th>
<th>Line3</th>
<th>Line4</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Highlighters</td>
<td>Lot # 7603</td>
<td>Yellow Blue Green</td>
<td>24 per box</td>
<td>A5296</td>
</tr>
<tr>
<td>2</td>
<td>Paper Clips</td>
<td>Lot # 7423</td>
<td>Small Medium Large</td>
<td></td>
<td>A5297</td>
</tr>
<tr>
<td>3</td>
<td>Gluestick</td>
<td>Lot # 4422</td>
<td>New item</td>
<td></td>
<td>A5298</td>
</tr>
<tr>
<td>4</td>
<td>Rubber bands</td>
<td>Lot # 7892</td>
<td>Wide width, Narrow width</td>
<td>1000 per box</td>
<td>A5299</td>
</tr>
</tbody>
</table>

See Also

Statements:
- “MERGE Statement” on page 304
- “MODIFY Statement” on page 309
- “SET Statement” on page 394
- “UPDATE Statement” on page 422

CALL Statement
Invokes a SAS CALL routine.

Valid in: DATA step
Category: Action
Type: Executable
**Syntax**

```sas
CALL routine(parameter-1 , ...parameter-n);
```

**Arguments**

- **routine** specifies the name of the SAS CALL routine that you want to invoke.

See For information about available routines, see *SAS Functions and CALL Routines: Reference*

- **(parameter)** is a piece of information to be passed to or returned from the routine.

  **Requirement** Enclose this information, which depends on the specific routine, in parentheses.

  **Tip** You can specify additional parameters, separated by commas.

**Details**

SAS CALL routines can assign variable values and perform other system functions.

**See Also**

*SAS Functions and CALL Routines: Reference*

---

**CARDS Statement**

Specifies that data lines follow.

- **Valid in:** DATA step
- **Category:** File-Handling
- **Type:** Declarative
- **Alias:** DATALINES, LINES
- **See:** “DATALINES Statement” on page 53

CARDS statement in the *SAS Companion for UNIX Environments*

---

**CARDS4 Statement**

Specifies that data lines that contain semicolons follow.

- **Valid in:** DATA step
- **Category:** File-Handling
- **Type:** Declarative
- **Alias:** DATALINES4, LINES4
- **See:** “DATALINES4 Statement” on page 54
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.

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, therefore, will allow 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 will be 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 will be 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:

```sas
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:

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

The SAS log displays this message:

```
Log 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

```sas
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.

```sas
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 94
- “FILENAME Statement, CATALOG Access Method” on page 103
- “LIBNAME Statement” on page 274 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
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.

Restrictions
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; */

CONTINUE Statement

Stops processing the current DO-loop iteration and resumes processing the next iteration.

**Valid in:** DATA step

**Category:** Control

**Type:** Executable

**Restriction:** Can be used only in a DO loop

**Syntax**

CONTINUE;

**Without Arguments**

The CONTINUE statement has no arguments. It stops processing statements within the current DO-loop iteration based on a condition. Processing resumes with the next iteration of the DO loop.

**Comparisons**

- The CONTINUE statement stops the processing of the current iteration of a loop and resumes with the next iteration; the LEAVE statement causes processing of the current loop to end.
- You can use the CONTINUE statement only in a DO loop; you can use the LEAVE statement in a DO loop or a SELECT group.
Example: Preventing Other Statements from Executing

This DATA step creates a report of benefits for new full-time employees. If an employee's status is PT (part-time), the CONTINUE statement prevents the second INPUT statement and the OUTPUT statement from executing.

```sas
data new_emp;
  drop i;
  do i=1 to 5;
    input name $ idno status $;
    /* return to top of loop * /
    /* when condition is true */
    if status='PT' then continue;
    input benefits $10.;
    output;
  end;
  datalines;
  Jones 9011 PT
  Thomas 876 PT
  Richards 1002 PT
  Eye/Dental
  Kelly 85111 PT
  Smith 433 PT
  HMO
;```

See Also

Statements:

- “DO Statement, Iterative” on page 62
- “LEAVE Statement” on page 270

**DATA Statement**

Begins a DATA step and provides names for any output SAS data sets, views, or programs.

- **Valid in:** DATA step
- **Category:** File-Handling
- **Type:** Declarative

**Syntax**

Form 1:  
```
DATA <data-set-name-1 <(data-set-options-1)> >
<...data-set-name-n <(data-set-options-n)> >
</DEBUG> <NESTING> <STACK = stack-size> > <NOLIST>;```

Form 2:  
```
DATA _NULL_ </DEBUG> <NESTING> <STACK = stack-size> > <NOLIST>;;```

Form 3:  
```
DATA view-name <data-set-name-1 <(data-set-options-1)> >
<...data-set-name-n <(data-set-options-n)> > /
VIEW=view-name <(password-option <SOURCE=source-option>)>
</NESTING> <NOLIST>;;```
**Form 4:**
```
DATA data-set-name / PGM=program-name (<password-option> <SOURCE=source-option>)
<NESTING> <NOLIST>;
```

**Form 5:**
```
DATA VIEW=view-name <(password-option)> <NOLIST>;
DESCRIBE;
```

**Form 6:**
```
DATA PGM=program-name <(password-option)> <NOLIST>;
<DESCRIBE>;
<REDIRECT INPUT | OUTPUT old-name-1 = new-name-1 <… old-name-n = new-name-n> ;>
<EXECUTE>;
```

**Without Arguments**

If you omit the arguments, the DATA step automatically names each successive data set that you create as DATA<n>, where <n> is the smallest integer that makes the name unique.

**Arguments**

**data-set-name**

names the SAS data file or DATA step view that the DATA step creates. To create a DATA step view, you must specify at least one **data-set-name** and that **data-set-name** must match **view-name**.

**Restriction**

**data-set-name** must conform to the rules for SAS names, and additional restrictions might be imposed by your operating environment.

**Tips**

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

You can execute a DATA step without creating a SAS data set. See “Example 5: Creating a Custom Report” on page 51. For more information, see “When Not Creating a Data Set (Form 2)” on page 49.

**See**

For more information about the types of SAS data set names and when to use each type, see “Names in the SAS Language” in *SAS Language Reference: Concepts*.

**(data-set-options)**

specifies optional arguments that the DATA step applies when it writes observations to the output data set.

**See**

*SAS Data Set Options: Reference* for a definition and list of data set options.

**Example**

“Example 1: Creating Multiple Data Files and Using Data Set Options” on page 50

**/ DEBUG**

enables you to debug your program interactively by helping identify logic errors, and sometimes data errors.

**/ NESTING**

specifies that a note will be printed to the SAS log for the beginning and end of each DO-END and SELECT-END nesting level. This option enables you to debug mismatched DO-END and SELECT-END statements and is particularly useful in large programs where the nesting level is not obvious.
/ STACK=stack-size
specifies the maximum number of nested LINK statements.

_NULL_
specifies that SAS does not create a data set when it executes the DATA step.

VIEW=view-name
names a view that the DATA step uses to store the input DATA step view.

Restrictions view-name must match one of the data set names.

SAS creates only one view in a DATA step.

Tips If you specify additional data sets in the DATA statement, SAS creates these data sets when the view is processed in a subsequent DATA or PROC step. Views have the capability of generating other data sets when the view is executed.

SAS macro variables resolve when the view is created. Use the SYMGET function to delay macro variable resolution until the view is processed.

Examples
“Example 2: Creating Input DATA Step Views” on page 50

“Example 3: Creating a View and a Data File” on page 50

password-option
assigns a password to a stored compiled DATA step program or a DATA step view.

Note: To DESCRIBE a password-protected DATA step program, you must specify its password. If the program has more than one password, you must specify the most restrictive password. ALTER is the most restrictive, and READ is the least restrictive. For more information, see “DESCRIBE Statement” on page 56.

The following password options are available:

ALTER=alter-password
assigns an ALTER password to a SAS data file. The password enables you to protect or replace a stored compiled DATA step program or a DATA step view.

要求
If you use an ALTER password in creating a stored compiled DATA step program or a DATA step view, an ALTER password is required to replace the program or view.

If you use an ALTER password in creating a stored compiled DATA step program or a DATA step view, an ALTER password is required to execute a DESCRIBE statement.

READ=read-password
assigns a READ password to a SAS data file. The password enables you to read or execute a stored compiled DATA step program or a DATA step view.

要求
If you use a READ password in creating a stored compiled DATA step program or a DATA step view, a READ password is required to execute the program or view.
If you use a READ password in creating a stored compiled DATA step program or a DATA step view, a READ password is required to execute DESCRIBE and EXECUTE statements. If you use an invalid password, SAS will execute the DESCRIBE statement.

**Tip**
If you use a READ password in creating a stored compiled DATA step program or a DATA step view, no password is required to replace the program or view.

**PW=password**  
assigns a READ and ALTER password, both having the same value.

**SOURCE=source-option**  
specifies one of the following source options:

**SAVE**  
saves the source code that created a stored compiled DATA step program or a DATA step view.

**ENCRYPT**  
encrypts and saves the source code that created a stored compiled DATA step program or a DATA step view.

**Tip**  
If you encrypt source code, use the ALTER password option as well. SAS issues a warning message if you do not use ALTER.

**NOSAVE**  
does not save the source code.

**CAUTION:**  
If you use the NOSAVE option for a DATA step view, the view cannot be migrated or copied from one version of SAS to another version.

**Default**  
**SAVE**

**PGM=program-name**  
names the stored compiled program that SAS creates or executes in the DATA step. To create a stored compiled program, specify a slash (/) before the PGM= option. To execute a stored compiled program, specify the PGM= option without a slash (/).

**Tip**  
SAS macro variables resolve when the stored program is created. Use the SYMGET function to delay macro variable resolution until the view is processed.

**Example**  
“Example 4: Storing and Executing a Compiled Program” on page 51

**NOLIST**  
suppresses the output of all variables to the SAS log when the value of _ERROR_ is 1.

**Restriction**  
NOLIST must be the last option in the DATA statement.
Details

Using the DATA Statement
The DATA step begins with the DATA statement. You use the DATA statement to create the following types of output: SAS data sets, data views, and stored programs. You can specify more than one output in a DATA statement. However, only one of the outputs can be a data view. You create a view by specifying the VIEW= option on page 46 and a stored program by specifying the PGM= option on page 47.

Using Both a READ and an ALTER Password
If you use both a READ and an ALTER password in creating a stored compiled DATA step program or a DATA step view, the following items apply:

- A READ or ALTER password is required to execute the stored compiled DATA step program or DATA step view.
- A READ or ALTER password is required if the stored compiled DATA step program or DATA step view contains both DESCRIBE and EXECUTE statements.
- If you use an ALTER password with the DESCRIBE and EXECUTE statements, the following items apply:
  - SAS executes both the DESCRIBE and the EXECUTE statements.
  - If you execute a stored compiled DATA step program or DATA step view with an invalid ALTER password:
    The DESCRIBE statement does not execute.
    In batch mode, the EXECUTE statement has no effect.
    In interactive mode, SAS prompts you for a READ password. If the READ password is valid, SAS processes the EXECUTE statement. If it is invalid, SAS does not process the EXECUTE statement.
- If you use a READ password with the DESCRIBE and EXECUTE statements, the following items apply:
  - In interactive mode, SAS prompts you for the ALTER password:
    If you enter a valid ALTER password, SAS executes both the DESCRIBE and the EXECUTE statements.
    If you enter an invalid ALTER password, SAS processes the EXECUTE statement but not the DESCRIBE statement.
  - In batch mode, SAS processes the EXECUTE statement but not the DESCRIBE statement.
  - In both interactive and batch modes, if you specify an invalid READ password SAS does not process the EXECUTE statement.
- An ALTER password is required if the stored compiled DATA step program or DATA step view contains a DESCRIBE statement.
- An ALTER password is required to replace the stored compiled DATA step program or DATA step view.

Creating an Output Data Set (Form 1)
Use the DATA statement to create one or more output data sets. You can use data set options to customize the output data set. The following DATA step creates two output
data sets, EXAMPLE1 and EXAMPLE2. It uses the data set option DROP to prevent the variable IDNUMBER from being written to the EXAMPLE2 data set.

```sas
data example1 example2 (drop=IDnumber);
  set sample;

  . . .more SAS statements . .
run;
```

**When Not Creating a Data Set (Form 2)**

Usually, the DATA statement specifies at least one data set name that SAS uses to create an output data set. However, when the purpose of a DATA step is to write a report or to write data to an external file, you might not want to create an output data set. Using the keyword _NULL_ as the data set name causes SAS to execute the DATA step without writing observations to a data set. This example writes to the SAS log the value of Name for each observation. SAS does not create an output data set.

```sas
data _NULL_;
  set sample;
  put Name ID;
run;
```

**Creating a DATA Step View (Form 3)**

You can create DATA step views and execute them at a later time. The following DATA step example creates a DATA step view. It uses the SOURCE=ENCRYPT option to both save and encrypt the source code.

```sas
data phone_list / view=phone_list (source=encrypt);
  set customer_list;

  . . .more SAS statements . .
run;
```

For more information, see “DATA Step Views” in *SAS Language Reference: Concepts*.

**Creating a Stored Compiled DATA Step Program (Form 4)**

The ability to compile and store DATA step programs enables you to execute the stored programs later. Stored compiled DATA step programs can reduce processing costs by eliminating the need to compile DATA step programs repeatedly. The following DATA step example compiles and stores a DATA step program. It uses the ALTER password option, which allows the user to replace an existing stored program, and to protect the stored compiled program from being replaced.

```sas
data testfile / pgm=stored.test_program (alter=sales);
  set sales_data;

  . . .more SAS statements . .
run;
```

For more information, see “Stored Compiled DATA Step Programs” in *SAS Language Reference: Concepts*.

**Describing a DATA Step View (Form 5)**

The following example uses the DESCRIBE statement in a DATA step view to write a copy of the source code to the SAS log.

```sas
data view=inventory;
  describe;
run;
```

For more information, see the “DESCRIBE Statement” on page 56.
Executing a Stored Compiled DATA Step Program (Form 6)
The following example executes a stored compiled DATA step program. It uses the DESCRIBE statement to write a copy of the source code to the SAS log.

```sas
libname stored 'SAS library';
data pgm=stored.employee_list;
  describe;
  execute;
run;
libname stored 'SAS library';
data pgm=stored.test_program;
  describe;
  execute;
  . . .more SAS statements . .
run;
```

For information, see the “DESCRIBE Statement” on page 56 and the “EXECUTE Statement” on page 73.

Examples

Example 1: Creating Multiple Data Files and Using Data Set Options
This DATA statement creates more than one data set, and it changes the contents of the output data sets:

```sas
data error (keep=subject date weight)
  fitness(label='Exercise Study'
    rename=(weight=pounds));
```

The ERROR data set contains three variables. SAS assigns a label to the FITNESS data set and renames the variable `weight` to `pounds`.

Example 2: Creating Input DATA Step Views
This DATA step creates an input DATA step view instead of a SAS data file:

```sas
libname ourlib 'SAS-library';
data ourlib.test / view=ourlib.test;
  set ourlib.fittest;
  tot=sum(of score1-score10);
run;
```

Example 3: Creating a View and a Data File
This DATA step creates an input DATA step view named THEIRLIB.TEST and an additional temporary SAS data set named SCORETOT:

```sas
libname ourlib 'SAS-library-1';
libname theirlib 'SAS-library-2';
data theirlib.test scoretot
  / view=theirlib.test;
  set ourlib.fittest;
  tot=sum(of score1-score10);
run;
```

SAS does not create the data file SCORETOT until a subsequent DATA or PROC step processes the view THEIRLIB.TEST.
**Example 4: Storing and Executing a Compiled Program**

The first DATA step produces a stored compiled program named STORED.SALESFIG:

```sas
libname in 'SAS-library-1';
libname stored 'SAS-library-2';
data salesdata / pgm=stored.salesfig;
  set in.sales;
  qtr1tot=jan+feb+mar;
run;
```

SAS creates the data set SALESDATA when it executes the stored compiled program STORED.SALESFIG.

```sas
data pgm=stored.salesfig;
run;
```

**Example 5: Creating a Custom Report**

The second DATA step in this program produces a custom report and uses the _NULL_ keyword to execute the DATA step without creating a SAS data set:

```sas
data sales;
  input dept : $10. jan feb mar;
datalines;
  shoes 4344 3555 2666
  housewares 3777 4888 7999
  appliances 53111 7122 41333
;
data _null_;  
  set sales;
  qtr1tot=jan+feb+mar;
  put 'Total Quarterly Sales: ' qtr1tot dollar12.;
run;
```

**Example 6: Using a Password with a Stored Compiled DATA Step Program**

The first DATA step creates a stored compiled DATA step program called STORED.ITEMS. This program includes the ALTER password, which limits access to the program.

```sas
data sample;
  input Name $ TotalItems $;
datalines;
  Lin 328
  Susan 433
  Ken 156
  Pat 340
;
proc print data=sample;
run;
libname stored 'SAS-library';
data employees / pgm=stored.items (alter=klondike);
  set sample;
  if TotalItems > 200 then output;
run;
```
This DATA step executes the stored compiled DATA step program STORED.ITEMS. It uses the DESCRIBE statement to print the source code to the SAS log. Because the program was created with the ALTER password, you must use the password if you use the DESCRIBE statement. If you do not enter the password, SAS will prompt you for it.

```sas
data pgm=stored.items (alter=klondike);
    describe;
    execute;
    run;
```

**Example 7: Displaying Nesting Levels**

The following program has two nesting levels. SAS will generate four log messages, one begin and end message for each nesting level.

```sas
data _null_ /nesting;
    do i = 1 to 10;
        do j = 1 to 5;
            put i= j=;
        end;
    end;
run;
```

**Log 2.2  Nesting Level Debug (partial SAS log)**

```
6   data _null_ /nesting;
7     do i = 1 to 10;
       -
719   NOTE 719-185: *** DO begin level 1 ***.
8       do j = 1 to 5;
       -
719   NOTE 719-185: *** DO begin level 2 ***.
9           put i= j=;
10         end;
11     end;
11     ---
720   NOTE 720-185: *** DO end level 2 ***.
12    run;
```

**See Also**

- “DATA Step Programming in Hadoop” in *SAS In-Database Products: User’s Guide*
- “Definition of Data Set Options” in *SAS Data Set Options: Reference*

**Statements:**

- “DESCRIBE Statement” on page 56
- “EXECUTE Statement” on page 73
- “LINK Statement” on page 293
DATALINES Statement

Specifies that data lines follow.

Valid in: DATA step
Category: File-Handling
Type: Declarative
Alias: CARDS, LINES
See: Data lines cannot contain semicolons. Use the “DATALINES4 Statement” on page 54 when your data contain semicolons.

Syntax

DATALINES;

Without Arguments

Use the DATALINES statement with an INPUT statement to read data that you enter directly in the program, rather than data stored in an external file.

Details

Using the DATALINES Statement

The DATALINES statement is the last statement in the DATA step and immediately precedes the first data line. Use a null statement (a single semicolon) to indicate the end of the input data.

You can use only one DATALINES statement in a DATA step. Use separate DATA steps to enter multiple sets of data.

Reading Long Data Lines

SAS handles data line length with the CARDIMAGE system option. If you use CARDIMAGE, SAS processes data lines exactly like 80-byte punched card images padded with blanks. If you use NOCARDIMAGE, SAS processes data lines longer than 80 columns in their entirety.

Using Input Options with In-stream Data

The DATALINES statement does not provide input options for reading data. However, you can access some options by using the DATALINES statement in conjunction with an INFILE statement. Specify DATALINES in the INFILE statement to indicate the source of the data and then use the options that you need. For more information, see “Example 2: Reading In-stream Data with Options” on page 54.

Comparisons

• Use the DATALINES statement whenever data do not contain semicolons. If your data contain semicolons, use the DATALINES4 statement.

• The following SAS statements also read data or point to a location where data are stored:
The INFILE statement points to raw data lines stored in another file. The INPUT statement reads those data lines.

The %INCLUDE statement brings SAS program statements or data lines stored in SAS files or external files into the current program.

The SET, MERGE, MODIFY, and UPDATE statements read observations from existing SAS data sets.

Examples

Example 1: Using the DATALINES Statement
In this example, SAS reads a data line and assigns values to two character variables, NAME and DEPT, for each observation in the DATA step:

```sas
data person;
  input name $ dept $;
datalines;
John Sales
Mary Acctng
;
```

Example 2: Reading In-stream Data with Options
This example takes advantage of options available with the INFILE statement to read in-stream data lines. With the DELIMITER= option, you can use list input to read data values that are delimited by commas instead of blanks.

```sas
data person;
  infile datalines delimiter=',';
  input name $ dept $;
datalines;
John,Sales
Mary,Acctng
;
```

See Also

Statements:
- “DATALINES4 Statement” on page 54
- “INFILE Statement” on page 198

System Options:
- “CARDIMAGE System Option” in SAS System Options: Reference

---

DATALINES4 Statement

Indicates that data lines that contain semicolons follow.

Valid in: DATA step

Category: File-Handling

Type: Declarative
Syntax

DATALINES4;

Without Arguments

Use the DATALINES4 statement together with an INPUT statement to read data that contain semicolons that you enter directly in the program.

Details

The DATALINES4 statement is the last statement in the DATA step and immediately precedes the first data line. Follow the data lines with four consecutive semicolons that are located in columns 1 through 4.

Comparisons

Use the DATALINES4 statement when data contain semicolons. If your data do not contain semicolons, use the DATALINES statement.

Example: Reading Data Lines That Contain Semicolons

In this example, SAS reads data lines that contain internal semicolons until it encounters a line of four semicolons. Execution continues with the rest of the program.

    data biblio;
      input number citation $50.;
      datalines4;
      KIRK, 1988
      2 LIN ET AL., 1995; BRADY, 1993
      3 BERG, 1990; ROA, 1994; WILLIAMS, 1992
    ;;;;

See Also

Statements:

• “DATALINES Statement” on page 53

DELETE Statement

Stops processing the current observation.

Valid in: DATA step
Category: Action
Type: Executable

Syntax

DELETE;
**Without Arguments**
When DELETE executes, the current observation is not written to a data set, and SAS returns immediately to the beginning of the DATA step for the next iteration.

**Details**
The DELETE statement is often used in a THEN clause of an IF-THEN statement or as part of a conditionally executed DO group.

**Comparisons**
- Use the DELETE statement when it is easier to specify a condition that excludes observations from the data set or when there is no need to continue processing the DATA step statements for the current observation.
- Use the subsetting IF statement when it is easier to specify a condition for including observations.
- Do not confuse the DROP statement with the DELETE statement. The DROP statement excludes variables from an output data set; the DELETE statement excludes observations.

**Examples**

**Example 1: Using the DELETE Statement as Part of an IF-THEN Statement**
When the value of LEAFWT is missing, the current observation is deleted:

```sas
if leafwt=. then delete;
```

**Example 2: Using the DELETE Statement to Subset Raw Data**

```sas
data topsales;
  infile file-specification;
  input region office product yrsales;
  if yrsales<100000 then delete;
run;
```

**See Also**

**Statements:**
- “DO Statement” on page 61
- “DROP Statement” on page 68
- “IF Statement, Subsetting” on page 187
- “IF-THEN/ELSE Statement” on page 189

---

**DESCRIBE Statement**
Retrieves source code from a stored compiled DATA step program or a DATA step view.

**Valid in:** DATA step

**Category:** Action
**Type:** Executable  
**Restriction:** Use DESCRIBE only with stored compiled DATA step programs and DATA step views.  
**Requirement:** You must specify the PGM= or the VIEW= option in the DATA statement.

### Syntax

```
DESCRIBE;
```

#### Without Arguments

Use the DESCRIBE statement to retrieve program source code from a stored compiled DATA step program or a DATA step view. SAS writes the source statements to the SAS log.

### Details

Use the DESCRIBE statement without the EXECUTE statement to retrieve source code from a stored compiled DATA step program or a DATA step view. Use the DESCRIBE statement with the EXECUTE statement to retrieve source code and execute a stored compiled DATA step program. For information about how to use these statements with the DATA statement, see “DATA Statement” on page 44.

**Note:** To DESCRIBE a password-protected view or DATA step program, you must specify its password. If the view or program was created with more than one password, you must specify the most restrictive password. As with other SAS files, the ALTER password is the most restrictive, and the READ password is the least restrictive. For more information, see “Executing a Stored Compiled DATA Step Program” in *SAS Language Reference: Concepts* and “Using Passwords with Views” in *SAS Language Reference: Concepts*.

### See Also

**Statements:**
- “DATA Statement” on page 44
- “EXECUTE Statement” on page 73

---

**DISPLAY Statement**

Displays a window that is created with the WINDOW statement.

**Valid in:** DATA step  
**Category:** Window Display  
**Type:** Executable

### Syntax

```
DISPLAY window<.group> <NOINPUT> <BLANK> <BELL> <DELETE>;
```
Arguments

`window<.group>`

names the window and group of fields to be displayed. This field is preceded by a period (.)

Tip  If the window has more than one group of fields, give the complete `window.group` specification. If a window contains a single unnamed group, use only `window`.

**NOINPUT**

specifies that you cannot input values into fields that are displayed in the window.

Default  If you omit NOINPUT, you can input values into unprotected fields that are displayed in the window.

Restriction  If you use NOINPUT in all DISPLAY statements in a DATA step, you must include a STOP statement to stop processing the DATA step.

Tip  The NOINPUT option is useful when you want to allow values to be entered into a window at some times but not others. For example, you can display a window once for entering values and a second time for verifying them.

**BLANK**

clears the window.

Tip  Use the BLANK option when you want to display different groups of fields in a window and you do not want text from the previous group to appear in the current display.

**BELL**

produces an audible alarm, beep, or bell sound when the window is displayed if your personal computer is equipped with a speaker device that provides sound.

**DELETE**

deletes the display of the window after processing passes from the DISPLAY statement on which the option appears.

Details

You must create a window in the same DATA step that you use to display it. Once you display a window, the window remains visible until you display another window over it or until the end of the DATA step. When you display a window that contains fields where you enter values, either enter a value or press Enter at each unprotected field to cause SAS to proceed to the next display. You cannot skip any fields.

While a window is being displayed, use commands and function keys to view other windows, to change the size of the current window, and so on.

A DATA step that contains a DISPLAY statement continues execution until the last observation that is read by a SET, MERGE, UPDATE, MODIFY, or INPUT statement has been processed or until a STOP or ABORT statement is executed. You can also issue the END command on the command line of the window to stop the execution of the DATA step.

You must create a window before you can display it. See the “WINDOW Statement” on page 434 for a description of how to create windows. A window that is displayed with the DISPLAY statement does not become part of the SAS log or output file.
Example

This DATA step creates and displays a window named start. The start window fills the entire screen. Both lines of text are centered.

```sas
data _null_;
    window start
        #5 @28 'WELCOME TO THE SAS SYSTEM'
        #12 @30 'PRESS ENTER TO CONTINUE';
    display start;
    stop;
run;
```

Although the start window in this example does not require you to input any values, you must press Enter to cause the execution to proceed to the STOP statement. If you omit the STOP statement, the DATA step executes endlessly unless you enter END on the command line of the window.

Note: Because this DATA step does not read any observations, SAS cannot detect an end-of-file to cause DATA step execution to cease. If you add the NOINPUT option to the DISPLAY statement, the window displays quickly and is removed.

See Also

Statements:

- “WINDOW Statement” on page 434

---

DM Statement

Submits SAS Program Editor, Log, Procedure Output or text editor commands as SAS statements.

**Valid in:** Anywhere

**Category:** Program Control

**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.

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

```sas
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.

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

DO Statement
Specifies a group of statements to be executed as a unit.

- **Valid in:** DATA step
- **Category:** Control
- **Type:** Executable

**Syntax**
```
DO;
...more SAS statements...
END;
```

**Without Arguments**
Use the DO statement for simple DO group processing.

**Details**
The DO statement is the simplest form of DO group processing. The statements between the DO and END statements are called a **DO group**. You can nest DO statements within DO groups.

**Note:** The memory capabilities of your system can limit the number of nested DO statements that you can use. For details, see the SAS documentation about how many levels of nested DO statements your system's memory can support.

A simple DO statement is often used within IF-THEN/ELSE statements to designate a group of statements to be executed depending on whether the IF condition is true or false.

**Comparisons**
There are three other forms of the DO statement:
• The iterative DO statement executes statements between DO and END statements repetitively based on the value of an index variable. The iterative DO statement can contain a WHILE or UNTIL clause.

• The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop.

• The DO WHILE statement executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop.

Example: Using the DO Statement

In this simple DO group, the statements between DO and END are performed only when YEARS is greater than 5. If YEARS is less than or equal to 5, statements in the DO group do not execute, and the program continues with the assignment statement that follows the ELSE statement.

```sas
if years>5 then
do;
    months=years*12;
    put years= months=;
end;
else yrsleft=5-years;
```

See Also

Statements:

• “DO Statement, Iterative” on page 62
• “DO UNTIL Statement” on page 66
• “DO WHILE Statement” on page 67

DO Statement, Iterative

Executes statements between the DO and END statements repetitively, based on the value of an index variable.

Valid in: DATA step
Category: Control
Type: Executable

Syntax

```
DO index-variable=specification-1 <, ... specification-n>
   ;
   ...more SAS statements...
END;
```

Arguments

`index-variable`

names a variable whose value governs execution of the DO group.
Tip  Unless you specify to drop it, the index variable is included in the data set that is being created.

CAUTION  Avoid changing the index variable within the DO group. If you modify the index variable within the iterative DO group, you might cause infinite looping.

**specification**

denotes an expression or a series of expressions in this form

\[ \text{start} \ <\text{TO stop}> \ <\text{BY increment}> \ <\text{WHILE(expression)}> \ | \ <\text{UNTIL(expression)}> \]

The DO group is executed first with \textit{index-variable} equal to \textit{start}. The value of \textit{start} is evaluated before the first execution of the loop.

\textit{start}

specifies the initial value of the index variable.

When it is used without \textit{TO stop} or \textit{BY increment}, the value of \textit{start} can be a series of items expressed in this form:

\[ \text{item-1} \ <, \ ... \item-n> ; \]

The items can be either all numeric or all character constants, or they can be variables. Enclose character constants in quotation marks. The DO group is executed once for each value in the list. If a WHILE condition is added, it applies only to the item that it immediately follows.

**Requirement**  When it is used with \textit{TO stop} or \textit{BY increment}, \textit{start} must be a number or an expression that yields a number.

**Example**  “Example 1: Using Various Forms of the Iterative DO Statement” on page 64

\textit{TO stop}

specifies the ending value of the index variable.

When both \textit{start} and \textit{stop} are present, execution continues (based on the value of \textit{increment}) until the value of \textit{index-variable} passes the value of \textit{stop}. When only \textit{start} and \textit{increment} are present, execution continues (based on the value of \textit{increment}) until a statement directs execution out of the loop, or until a WHILE or UNTIL expression that is specified in the DO statement is satisfied. If neither \textit{stop} nor \textit{increment} is specified, the group executes according to the value of \textit{start}. The value of \textit{stop} is evaluated before the first execution of the loop.

**Requirement**  \textit{Stop} must be a number or an expression that yields a number.

**Tip**  Any changes to \textit{stop} made within the DO group do not affect the number of iterations. To stop iteration of a loop before it finishes processing, change the value of \textit{index-variable} so that it passes the value of \textit{stop}, or use a LEAVE statement to go to a statement outside the loop.

**Example**  “Example 1: Using Various Forms of the Iterative DO Statement” on page 64

\textit{BY increment}

specifies a positive or negative number (or an expression that yields a number) to control the incrementing of \textit{index-variable}. 
The value of \textit{increment} is evaluated before the execution of the loop. Any changes to the increment that are made within the DO group do not affect the number of iterations. If no increment is specified, the index variable is increased by 1. When \textit{increment} is positive, \textit{start} must be the lower bound and \textit{stop}, if present, must be the upper bound for the loop. If \textit{increment} is negative, \textit{start} must be the upper bound and \textit{stop}, if present, must be the lower bound for the loop.

Example

```
Example 1: Using Various Forms of the Iterative DO Statement” on page 64
```

\textbf{WHILE(}expression\textbf{)} | \textbf{UNTIL(}expression\textbf{)}

Evaluates, either before or after execution of the DO group, any SAS expression that you specify. Enclose the expression in parentheses.

A WHILE expression is evaluated before each execution of the loop, so that the statements inside the group are executed repetitively while the expression is true.

An UNTIL expression is evaluated after each execution of the loop, so that the statements inside the group are executed repetitively until the expression is true.

\textbf{Restriction}

A WHILE or UNTIL specification affects only the last item in the clause in which it is located.

\textbf{See}

“DO WHILE Statement” on page 67 and “DO UNTIL Statement” on page 66

\textbf{Example}

```
“Example 1: Using Various Forms of the Iterative DO Statement” on page 64
```

\textbf{Requirement}

The iterative DO statement requires at least one \textit{specification} argument.

\textbf{Tips}

The order of the optional TO and BY clauses can be reversed.

When you use more than one \textit{specification}, each one is evaluated before its execution.

\textbf{Comparisons}

There are three other forms of the DO statement:

- The DO statement, the simplest form of DO-group processing, designates a group of statements to be executed as a unit, usually as a part of IF-THEN/ELSE statements.
- The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop.
- The DO WHILE statement executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop.

\textbf{Examples}

\textbf{Example 1: Using Various Forms of the Iterative DO Statement}

- These iterative DO statements use a list of items for the value of \textit{start}:
  - \texttt{do month='JAN','FEB','MAR';}
  - \texttt{do count=2,3,5,7,11,13,17;}
• do i=5;
• do i=var1, var2, var3;
• do i='01JAN2001'd,'25FEB2001'd,'18APR2001'd;

• These iterative DO statements use the start TO stop syntax:
  • do i=1 to 10;
  • do i=1 to exit;
  • do i=1 to x-5;
  • do i=1 to k-1, k+1 to n;
  • do i=k+1 to n-1;

• These iterative DO statements use the BY increment syntax:
  • do i=n to 1 by -1;
  • do i=.1 to .9 by .1, 1 to 10 by 1,
    20 to 100 by 10;
  • do count=2 to 8 by 2;

• These iterative DO statements use WHILE and UNTIL clauses:
  • do i=1 to 10 while(x<y);
  • do i=2 to 20 by 2 until((x/3)>y);
  • do i=10 to 0 by -1 while(month='JAN');

• In this example, the DO loop is executed when I=1 and I=2; the WHILE condition is
  evaluated when I=3, and the DO loop is executed if the WHILE condition is true.
  DO I=1,2,3 WHILE {condition};

**Example 2: Using the Iterative DO Statement without Infinite Looping**

In each of the following examples, the DO group executes ten times. The first example
demonstrates the preferred approach.

```sas
/* correct coding */
do i=1 to 10;
  ...more SAS statements...
end;
```

The next example uses the TO and BY arguments.

```sas
do i=1 to n by m;
  ...more SAS statements...
  if i=10 then leave;
  end;
  if i=10 then put 'EXITED LOOP';
```

**Example 3: Stopping the Execution of the DO Loop**

In this example, setting the value of the index variable to the current value of EXIT
causes the loop to terminate.

```sas
data iterate1;
  input x;
  exit=10;
do i=1 to exit;
```
$y = x \ast \text{normal}(0);$  
/* if $y > 25,$ */  
/* changing i's value */  
/* stops execution */  
if $y > 25$ then $i = \text{exit};$  
output;  
end;  
datalines;  
5 000 2500 ;

See Also

Statements:

- “ARRAY Statement” on page 18
- “Array Reference Statement” on page 23
- “DO Statement” on page 61
- “DO UNTIL Statement” on page 66
- “DO WHILE Statement” on page 67
- “GO TO Statement” on page 185

DO UNTIL Statement

Executes statements in a DO loop repetitively until a condition is true.

Valid in: DATA step
Category: Control
Type: Executable

Syntax

DO UNTIL (expression);
…more SAS statements…
END;

Arguments

(expression)

(is any SAS expression, enclosed in parentheses. You must specify at least one expression.

Details

The expression is evaluated at the bottom of the loop after the statements in the DO loop have been executed. If the expression is true, the DO loop does not iterate again.

Note: The DO loop always iterates at least once.
Comparisons

There are three other forms of the DO statement:

- The DO statement, the simplest form of DO-group processing, designates a group of statements to be executed as a unit, usually as a part of IF-THEN/ELSE statements.

- The iterative DO statement executes statements between DO and END statements repetitively based on the value of an index variable.

- The DO WHILE statement executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop. The DO UNTIL statement evaluates the condition at the bottom of the loop; the DO WHILE statement evaluates the condition at the top of the loop.

  Note: The statements in a DO UNTIL loop always execute at least one time, whereas the statements in a DO WHILE loop do not iterate even once if the condition is false.

Example: Using a DO UNTIL Statement to Repeat a Loop

These statements repeat the loop until N is greater than or equal to 5. The expression N>=5 is evaluated at the bottom of the loop. There are five iterations in all (0, 1, 2, 3, 4).

```
n=0;
do until(n>=5);
   put n=;
   n+1;
end;
```

See Also

Statements:

- “DO Statement” on page 61
- “DO Statement, Iterative” on page 62
- “DO WHILE Statement” on page 67

---

**DO WHILE Statement**

Executes statements in a DO-loop repetitively while a condition is true.

- **Valid in:** DATA step
- **Category:** Control
- **Type:** Executable

**Syntax**

```
DO WHILE (expression);
   …more SAS statements…
END;
```
Arguments

(expression)

is any SAS expression, enclosed in parentheses. You must specify at least one expression.

Details

The expression is evaluated at the top of the loop before the statements in the DO loop are executed. If the expression is true, the DO loop iterates. If the expression is false the first time it is evaluated, the DO loop does not iterate even once.

Comparisons

There are three other forms of the DO statement:

• The DO statement, the simplest form of DO-group processing, designates a group of statements to be executed as a unit, usually as a part of IF-THEN/ELSE statements.

• The iterative DO statement executes statements between DO and END statements repetitively based on the value of an index variable.

• The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop. The DO WHILE statement evaluates the condition at the top of the loop; the DO UNTIL statement evaluates the condition at the bottom of the loop.

Note: If the expression is false, the statements in a DO WHILE loop do not execute. However, because the DO UNTIL expression is evaluated at the bottom of the loop, the statements in the DO UNTIL loop always execute at least once.

Example: Using a DO WHILE Statement

These statements repeat the loop while N is less than 5. The expression N<5 is evaluated at the top of the loop. There are five iterations in all (0, 1, 2, 3, 4).

```sas
n=0;
do while(n<5);
   put n=;
   n+1;
end;
```

See Also

Statements:

• “DO Statement” on page 61
• “DO Statement, Iterative” on page 62
• “DO UNTIL Statement” on page 66

DROP Statement

Excludes variables from output SAS data sets.

Valid in: DATA step
**Syntax**

DROP variable-list;

**Arguments**

variable-list

specifies the names of the variables to omit from the output data set.

Tip You can list the variables in any form that SAS allows.

**Details**

The DROP statement applies to all the SAS data sets that are created within the same DATA step and can appear anywhere in the step. The variables in the DROP statement are available for processing in the DATA step. If no DROP or KEEP statement appears, all data sets that are created in the DATA step contain all variables. Do not use both DROP and KEEP statements within the same DATA step.

**Comparisons**

- The DROP statement differs from the DROP= data set option in the following ways:
  - You cannot use the DROP statement in SAS procedure steps.
  - The DROP statement applies to all output data sets that are named in the DATA statement. To exclude variables from some data sets but not from others, use the DROP= data set option in the DATA statement.
  - The KEEP statement is a parallel statement that specifies a list of variables to write to output data sets. Use the KEEP statement instead of the DROP statement if the number of variables to include is significantly smaller than the number to omit.
  - Do not confuse the DROP statement with the DELETE statement. The DROP statement excludes variables from output data sets; the DELETE statement excludes observations.

**Examples**

**Example 1: Basic DROP Statement Usage**

These examples show the correct syntax for listing variables with the DROP statement:

- drop time shift batchnum;
- drop grade1-grade20;

**Example 2: Dropping Variables from the Output Data Set**

In this example, the variables PURCHASE and REPAIR are used in processing but are not written to the output data set INVENTORY:

data inrenty;
  drop purchase repair;
  infile file-specification;
input unit part purchase repair;
totcost=sum(purchase,repair);
run;

See Also

Data Set Options:
• “DROP= Data Set Option” in SAS Data Set Options: Reference

Statements:
• “DELETE Statement” on page 55
• “KEEP Statement” on page 266

END Statement

Ends a DO group or SELECT group processing.

Valid in: DATA step
Category: Control
Type: Declarative

Syntax
END;

Without Arguments
Use the END statement to end DO group or SELECT group processing.

Details
The END statement must be the last statement in a DO group or a SELECT group.

Example: Using the END Statement

This example shows a simple DO group and a simple SELECT group:
• do;
  ...more SAS statements...
  end;
• select(expression);
  when(expression) SAS statement;
  otherwise SAS statement;
  end;

See Also

Statements:
• “DO Statement” on page 61
**ENDSAS Statement**

Terminates a SAS job or session after the current DATA or PROC step executes.

**Valid in:** Anywhere  
**Category:** Program Control

**Syntax**

```
ENDSAS;
```

**Without Arguments**

The ENDSAS statement terminates a SAS job or session.

**Details**

ENDSAS is most useful in interactive or windowing sessions.

*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*

---

**ERROR Statement**

Sets _ERROR_ to 1. A message written to the SAS log is optional.

**Valid in:** DATA step  
**Category:** Action  
**Type:** Executable

**Syntax**

```
ERROR <message>;
```

**Without Arguments**

Using ERROR without an argument sets the automatic variable _ERROR_ to 1 writes a blank message to the log.
Arguments

message
    writes a message to the log.

Tip  message can include character literals (enclosed in quotation marks), variable names, formats, and pointer controls.

Details

The ERROR statement sets the automatic variable _ERROR_ to 1. Writing a message that you specify to the SAS log is optional. When _ERROR_ = 1, SAS writes the data lines that correspond to the current observation in the SAS log.

Using ERROR is equivalent to using these statements in combination:

- an assignment statement setting _ERROR_ to 1
- a FILE LOG statement
- a PUT statement (if you specify a message)
- a PUT; statement (if you do not specify a message)
- another FILE statement resetting FILE to any previously specified setting.

Example: Writing Error Messages

In the following examples, SAS writes the error message and the variable name and value to the log for each observation that satisfies the condition in the IF-THEN statement.

- In this example, the ERROR statement automatically resets the FILE statement specification to the previously specified setting.

```sas
file file-specification;
    if type='teen' & age > 19 then
        error 'type and age don"t match ' age=;
```

- This example uses a series of statements to produce the same results.

```sas
file file-specification;
    if type='teen' & age > 19 then
do;
    file log;
    put 'type and age don"t match ' age=;
    _error_=1;
    file file-specification;
end;
```

See Also

Statements:

- “PUT Statement” on page 335
EXECUTE Statement
Executes a stored compiled DATA step program.

Valid in: DATA step
Category: Action
Type: Executable
Restriction: Use EXECUTE with stored compiled DATA step programs only.
Requirement: You must specify the PGM= option in the DATA step.

Syntax
EXECUTE;

Without Arguments
The EXECUTE statement executes a stored compiled DATA step program.

Details
Use the DESCRIBE statement with the EXECUTE statement in the same DATA step to retrieve the source code and execute a stored compiled DATA step program. If you do not specify either statement, EXECUTE is assumed. The order in which you use the statements is interchangeable. The DATA step program executes when it reaches a step boundary. For information about how to use these statements with the DATA statement, see the “DATA Statement” on page 44.

See Also
Statements:
• “DATA Statement” on page 44
• “DESCRIBE Statement” on page 56

FILE Statement
Specifies the current output file for PUT statements.

Valid in: DATA step
Category: File-Handling
Type: Executable
Restriction: When SAS is in a locked-down state, the FILENAME statement is not available for files that are not in the locked-down path list. For more information, see “SAS Processing Restrictions for Servers in a Locked-Down State” in SAS Language Reference: Concepts.
See: FILE Statement under Windows, UNIX, and z/OS
Syntax

FILE file-specification <device-type> <options> <operating-environment-options>;

Arguments

file-specification

identifies an external file that the DATA step uses to write output from a PUT statement. File-specification can have these forms:

'external-file'
specifies the physical name of an external file, which is enclosed in quotation marks. The physical name is the name by which the operating environment recognizes the file.

fileref

specifies the fileref of an external file.

Requirement

You must have associated fileref with an external file in a FILENAME statement or function in a previous step or in an appropriate operating environment command. The only way to assign the fileref at run time is to use the FILEVAR= option in the FILE statement.

See

“FILENAME Statement” on page 94

fileref(file)

specifies a fileref that is previously assigned to an external file that is an aggregate grouping of files. Follow the fileref with the name of a file or member, which is enclosed in parentheses.

Requirement

You must previously associate fileref with an external file in a FILENAME statement or function, or in an appropriate operating environment command.

Operating environment

Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set. For details, 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.

See

“FILENAME Statement” on page 94

LOG

is a reserved fileref that directs the output that is produced by any PUT statements to the SAS log.

At the beginning of each execution of a DATA step, the fileref that indicates where the PUT statements write is automatically set to LOG. Therefore, the first PUT statement in a DATA step always writes to the SAS log, unless it is preceded by a FILE statement that specifies otherwise.

Tip

Because output lines are by default written to the SAS log, use a FILE LOG statement to restore the default action or to specify additional FILE statement options.
PRINT
is a reserved fileref that directs the output that is produced by any PUT statements to the same file as the output that is produced by SAS procedures.

Interaction
When you write to a file, the value of the N= option must be either 1 or PAGESIZE.

Operating environment
The carriage-control characters that are written to a file can be specific to the operating environment. For details, see the SAS documentation for your operating environment.

Tip
When PRINT is the fileref, SAS uses carriage-control characters and writes the output with the characteristics of a print file.

See
A complete discussion of print files in SAS Language Reference: Concepts

Tip
If the file does not exist in the directory that you specify for file-specification, SAS creates the file. If the directory specified in file-specification does not exist, SAS sets the SYSERR macro variable, which can be checked if the ERRORCHECK option is set to STRICT.

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 a location that is not a physical file:

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

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

CATALOG
specifies the CATALOG access method.

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
For a complete list of options that are available with the CATALOG access method, see the “FILENAME Statement, CATALOG Access Method” on page 103.

CLIPBOARD
specifies the CLIPBOARD access method.

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
For a complete list of options that are available with the CLIPBOARD access method, see the “FILENAME Statement, CLIPBOARD Access Method” on page 106.
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.

FTP
specifies the FTP access method.

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 For a complete list of options that are available with the FTP access method, see the “FILENAME Statement, FTP Access Method” on page 126.

Example infile dummy ftp user='myuid' pass='xxxx' filevar=file_to_read;

HADOOP
specifies the Hadoop access method

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 For a complete list of options that are available with the Hadoop access method, see the “FILENAME Statement, Hadoop Access Method” on page 141.

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

JMS
specifies a Java Message Service (JMS) destination.

PIPE
specifies an unnamed pipe.

Operating environment Some operating environments do not support pipes.

PLOTTER
specifies an unbuffered graphics output device.

PRINTER
specifies a printer or printer spool file.

SFTP
specifies the SFTP access method.

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 For a complete list of options that are available with the SFTP access method, see the “FILENAME Statement, SFTP Access Method” on page 148.

**SOCKET**
specifies the SOCKET access method.

**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 For a complete list of options that are available with the SOCKET access method, see the “FILENAME Statement, SOCKET Access Method” on page 155.

**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 the URL access method.

**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 For a complete list of options that are available with the URL access method, see the “FILENAME Statement, URL Access Method” on page 160.

**WEBDAV**
specifies the WEBDAV access method.

**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 For a complete list of options that are available with the WEBDAV access method, see the “FILENAME Statement, WebDAV Access Method” on page 166.
Alias

DEVICES=device-type

Default

DISK

Requirement

device-type or DEVICE=device-type must immediately follow file-specification 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.

Options

BLKSIZE=block-size

specifies the block size of the output file.

Default

Dependent on your operating environment. For details, see the FILE Statement in the SAS documentation for your operating environment.

COLUMN=variable

specifies a variable that SAS automatically sets to the current column location of the pointer. This variable, like automatic variables, is not written to the data set.

Alias

COL=

See

LINE= on page 82

DELIMITER=delimiter(s)

specifies an alternate delimiter (other than blank) to be used for LIST output where delimiter can be one of the following items.

'list-of-delimiting-characters'

specifies one or more characters to write as delimiters.

Requirement

Enclose the list of characters in quotation marks.

character-variable

specifies a character variable whose value becomes the delimiter.

Alias

DLM=

Default

blank space

Restriction

Even though a character string or character variable is accepted, only the first character of the string or variable is used as the output delimiter. The FILE DLM= processing differs from INFILE DELIMITER= processing.

Interaction

Output that contains embedded delimiters requires the delimiter sensitive data (DSD) option.

Tips

DELIMITER= can be used with the colon (:) modifier (modified LIST output).

The delimiter is case sensitive.
**FILE Statement** 79

**DLMSTR**=

A character string that is alternately used for LIST output. It can be one of the following items:

- A delimiting string
- A character variable

**Requirement**
Enclose the string in quotation marks.

**Default**
Blank space

**Interactions**
If you specify more than one DLMSTR= option in the FILE statement, the DLMSTR= option that is specified last will be used. If you specify both the DELIMITER= and DLMSTR= options, the option that is specified last will be used.

If you specify RECFM=N, make sure that the LRECL is large enough to hold the largest input item. Otherwise, it might be possible for the delimiter to be split across the record boundary.

**See**
DELIMITER= on page 78, DLMSOPT= on page 79, and DSD on page 80

**DLMSOPT**= 'T' | 't'

A parsing option for the DLMSTR= T option that removes trailing blanks of the string delimiter.

**Requirement**
The DLMSOPT=T option has an effect only when used with the DLMSTR= option.

**Tip**
The DLMSOPT=T option is useful when you use a variable as the delimiter string.

**See**
DLMSTR= on page 79

**DROPOVER**

Discards data items that exceed the output line length (as specified by the LINESIZE= or LRECL= options in the FILE statement).

- By default, data that exceeds the current line length is written on a new line.
- When you specify DROPOVER, SAS drops (or ignores) an entire item when there is not enough space in the current line to write it. When an entire item is dropped, the column pointer remains positioned after the last value that is written in the current line. Thus, the PUT statement might write other items in the current output line if they fit in the space that remains or if the column pointer is repositioned. When a data item is dropped, the DATA step continues normal execution (_ERROR_=0). At the end of the DATA step, a message is printed for each file from which data was lost.

**Default**
FLOWOVER
Tip Use DROPOVER when you want the DATA step to continue executing if the PUT statement attempts to write past the current line length, but you do not want the data item that exceeds the line length to be written on a new line.

See “FLOWOVER” on page 81 and “STOPOVER” on page 86

**DSD (delimiter sensitive data)**
specifies that data values that contain embedded delimiters, such as tabs or commas, be enclosed in quotation marks. The DSD option enables you to write data values that contain embedded delimiters to LIST output. This option is ignored for other types of output (for example, formatted, column, and named). Any double quotation marks that are included in the data value are repeated. When a variable value contains the delimiter and DSD is used in the FILE statement, the variable value will be enclosed in double quotation marks when the output is generated. For example, the following code

```
DATA _NULL_
   FILE log dsd;
   x="lions, tigers, and bears";
   put x ' "Oh, my!"';
run;
```

will result in the following output:

"""lions, tigers, and bears"", "Oh, my!"

If a quoted (text) string contains the delimiter and DSD is used in the FILE statement, then the quoted string will not be enclosed in double quotation marks when used in a PUT statement. For example, the following code

```
DATA _NULL_
   FILE log dsd;
   PUT 'lions, tigers, and bears';
run;
```

will result in the following output:

lions, tigers, and bears

**Interaction** If you specify DSD, the default delimiter is assumed to be the comma (,). Specify the DELIMITER= or DLMSTR= option if you want to use a different delimiter.

**Tip** By default, data values that do not contain the delimiter that you specify are not enclosed in quotation marks. However, you can use the tilde (~) modifier to force any data value, including missing values, to be enclosed in quotation marks, even if it contains no embedded delimiter.

**See** DELIMITER= on page 78 and DLMSTR= on page 79

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

When you write data to the output file, SAS transcodes the data from the session encoding to the specified encoding.

**Default** SAS uses the current session encoding.
### FILENAME=variable

Defines a character variable, whose name you supply, that SAS sets to the value of the physical name of the file currently open for PUT statement output. The physical name is the name by which the operating environment recognizes the file.

**Tips**
- This variable, like automatic variables, is not written to the data set.
- Use a LENGTH statement to make the variable length long enough to contain the value of the physical filename if the variable length is longer than eight bytes (the default length of a character variable).

### FILEVAR=variable

Defines a variable whose change in value causes the FILE statement to close the current output file and open a new one the next time the FILE statement executes. The next PUT statement that executes writes to the new file that is specified as the value of the FILEVAR= variable.

**Restriction**
- The value of a FILEVAR= variable is expressed as a character string that contains a physical filename.

**Interaction**
- When you use the FILEVAR= option, the file-specification is just a placeholder, not an actual filename or a fileref that has been previously assigned to a file. SAS uses this placeholder for reporting processing information to the SAS log. It must conform to the same rules as a fileref.

**Tips**
- This variable, like automatic variables, is not written to the data set.
- If any of the physical filenames is longer than eight bytes (the default length of a character variable), assign the FILEVAR= variable a longer length with another statement, such as a LENGTH statement or an INPUT statement.

### FLOWOVER

Causes data that exceeds the current line length to be written on a new line. When a PUT statement attempts to write beyond the maximum allowed line length (as specified by the LINESIZE= option in the FILE statement), the current output line is written to the file and the data item that exceeds the current line length is written to a new line.

**Default**
- FLOWOVER
Interaction
If the PUT statement contains a trailing @, the pointer is positioned after the data item on the new line, and the next PUT statement writes to that line. This process continues until the end of the input data is reached or until a PUT statement without a trailing @ causes the current line to be written to the file.

See “DROPOVER” on page 79 and “STOPOVER” on page 86

FOOTNOTES | NOFOOTNOTES
controls whether currently defined footnotes are printed.

Alias FOOTNOTE | NOFOOTNOTE
Default NOFOOTNOTES
Requirement In order to print footnotes in a DATA step report, you must set the FOOTNOTE option in the FILE statement.

HEADER=label
defines a statement label that identifies a group of SAS statements that you want to execute each time SAS begins a new output page.

Restrictions The first statement after the label must be an executable statement. Thereafter, you can use any SAS statement.

Tip To prevent the statements in this group from executing with each iteration of the DATA step, use two RETURN statements: one precedes the label and the other appears as the last statement in the group.

Example “Example 1: Executing Statements When Beginning a New Page” on page 90

LINE=variable
defines a variable whose value is the current relative line number within the group of lines available to the output pointer. You supply the variable name; SAS automatically assigns the value.

Range 1 to the value that is specified by the N= option or with the #n line pointer control. If neither is specified, the LINE= variable has a value of 1.

Tips This variable, like automatic variables, is not written to the data set.

The value of the LINE= variable is set at the end of PUT statement execution to the number of the next available line.

LINESIZE=line-size
sets the maximum number of columns per line for reports and the maximum record length for data files.

Alias LS=
Default The default LINESIZE= value is determined by one of two options:
1) the LINESIZE= system option when you write to file that
contains carriage-control characters or to the SAS log or 2) the LRECL= option in the FILE statement when you write to a file.

<table>
<thead>
<tr>
<th>Range</th>
<th>From 64 to the maximum logical record length that is allowed in your operating environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>If a PUT statement tries to write a line that is longer than the value that is specified by the LINESIZE= option, the action that is taken is determined by whether FLOWOVER, DROPOVER, or STOPOVER is in effect. By default (FLOWOVER), SAS writes the line as two or more separate records.</td>
</tr>
<tr>
<td>Operating environment</td>
<td>The highest value allowed for LINESIZE= is dependent on your operating environment. For details, see the SAS documentation for your operating environment.</td>
</tr>
<tr>
<td>Note</td>
<td>LINESIZE= tells SAS how much of the line to use. LRECL= specifies the physical record length of the file.</td>
</tr>
<tr>
<td>See</td>
<td>LRECL= on page 83, “DROPOVER” on page 79, “FLOWOVER” on page 81, and “STOPOVER” on page 86</td>
</tr>
<tr>
<td>Example</td>
<td>“Example 6: When the Output Line Exceeds the Line Length of the Output File” on page 92</td>
</tr>
</tbody>
</table>

**LINESLEFT=** variable

defines a variable whose value is the number of lines left on the current page. You supply the variable name; SAS assigns the value of the number of lines left on the current page to that variable. The value of the LINESLEFT= variable is set at the end of PUT statement execution.

<table>
<thead>
<tr>
<th>Alias</th>
<th>LL=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tip</td>
<td>This variable, like automatic variables, is not written to the data set.</td>
</tr>
<tr>
<td>Example</td>
<td>“Example 2: Determining New Page by Lines Left on the Current Page” on page 90</td>
</tr>
</tbody>
</table>

**LRECL=** variable

specifies the logical record length of the output file.

| Default | If you omit the LRECL= option, SAS chooses a value based on the operating environment's file characteristics. |
| Interaction | Alternatively, you can specify a global logical record length by using the LRECL system option. 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 logical-record-length are dependent on the operating environment. For details, see the SAS documentation for your operating environment. |
| Note | LINESIZE= tells SAS how much of the line to use; LRECL= specifies the physical line length of the file. |
MOD
writes the output lines after any existing lines in the file.

Default OLD

Restrictions MOD is not accepted under all operating environments. For more information, see the SAS documentation for your operating environment.

Do not use the MOD option with any ODS destination other than the Listing destination. Otherwise, you might receive unexpected output.

See “OLD” on page 85

N=available-lines
specifies the number of lines that you want available to the output pointer in the current iteration of the DATA step. Available-lines can be expressed as a number (n) or as the keyword PAGESIZE or PS.

n specifies the number of lines that are available to the output pointer. The system can move back and forth between the number of lines that are specified while composing them before moving on to the next set.

PAGESIZE specifies that the entire page is available to the output pointer.

Alias PS

Restrictions N=PAGESIZE is valid only when output is printed.

Interactions In addition to use in the N= option to control the number of lines available to the output pointer, you can also use the #n line pointer control in a PUT statement.

If the current output file is a file that is to be printed, available-lines must have a value of either 1 or PAGESIZE.

Tip Setting N=PAGESIZE enables you to compose a page of multiple columns one column at a time.

Example “Example 3: Arranging the Contents of an Entire Page” on page 90

ODS < = (ODS-suboptions) > specifies to use the Output Delivery System to format the output from a DATA step. It defines the structure of the data component and holds the results of the DATA step and binds that component to a table definition to produce an output object. ODS sends this object to all open ODS destinations, each of which formats the output

| **Defaults** | If you omit the ODS suboptions, the DATA step uses a default table definition (base.datastep.table) that is stored in the SASHELP.TMPLMST template store. This definition defines two generic columns: one for character variables, and one for numeric variables. ODS associates each variable in the DATA step with one of these columns and displays the variables in the order in which they are defined in the DATA step. Without suboptions, the default table definition uses the variable's label as its column heading. If no label exists, the definition uses the variable's name as the column heading. |
| **Restriction** | You cannot use _FILE_, FILEVAR=, HEADER=, and PAD with the ODS option. |
| **Requirement** | The ODS option is valid only when you use the fileref PRINT in the FILE statement. |
| **Interaction** | The DELIMITER= and DSD options have no effect on the ODS option. The FOOTNOTES|NOFOOTNOTES, LINESIZE, PAGESIZE, and TITLES | NOTITLES options have an effect only on the LISTING destination. |

**OLD** replaces the previous contents of the file.

| **Default** | OLD |
| **Restriction** | OLD is not accepted under all operating environments. For details, see the SAS documentation for your operating environment. |
| **See** | “MOD” on page 84 |

**PAD | NOPAD** controls whether records written to an external file are padded with blanks to the length that is specified in the LRECL= option.

| **Default** | NOPAD is the default when writing to a variable-length file; PAD is the default when writing to a fixed-length file. |
| **Tip** | PAD provides a quick way to create fixed-length records in a variable-length file. |
| **See** | LRECL= on page 83 |

**PAGESIZE=value** sets the number of lines per page for your reports.

| **Alias** | PS= |
| **Default** | the value of the PAGESIZE= system option. |
| **Range** | The value can range from 15 to 32767. |
**Interaction**  
If any TITLE statements are currently defined, the lines that they occupy are included in counting the number of lines for each page.

**Tips**  
After the value of the PAGESIZE= option is reached, the output pointer advances to line 1 of a new page.

If you specify FILE LOG, the number of lines that are output on the first page is reduced by the number of lines in the SAS start-up notes. For example, if PAGESIZE=20 and there are nine lines of SAS start-up notes, only 11 lines are available for output on the first page.

**See**  
“PAGESIZE= System Option” in *SAS System Options: Reference*

---

**PRINT | NOPRINT**  
controls whether carriage-control characters are placed in the output lines.

**Restriction**  
When you write to a file, the value of the N= option must be either 1 or PAGESIZE.

**Operating environment**  
The carriage-control characters that are written to a file can be specific to the operating environment. For details, see the SAS documentation for your operating environment.

**Tips**  
The PRINT option is not necessary if you are using fileref PRINT.

If you specify FILE PRINT in an interactive SAS session, then the Output window interprets the form-feed control characters as page breaks, and blank lines that are output before the form feed are removed from the output. Writing the results from the Output window to a flat file produces a file without page break characters. If a file needs to contain the form-feed characters, then the FILE statement should include a physical file location and the PRINT option.

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

**Range**  
Values are dependent on the operating environment. For more information, see the SAS documentation for your operating environment.

**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.

**STOPOVER**  
stops processing the DATA step immediately if a PUT statement attempts to write a data item that exceeds the current line length. In such a case, SAS discards the data item that exceeds the current line length, writes the portion of the line that was built before the error occurred, and issues an error message.

**Default**  
FLOWOVER

**See**  
“FLOWOVER” on page 81 and “DROPOVER” on page 79
TITLES | NOTITLES
controls the printing of the current title lines on the pages of files. When NOTITLES is omitted, or when TITLES is specified, SAS prints any titles that are currently defined.

Alias TITLE | NOTITLE

Default TITLES

_FILE_=variable
names a character variable that references the current output buffer of this FILE statement. You can use the variable in the same way as any other variable, even as the target of an assignment. The variable is automatically retained and initialized to blanks. Like automatic variables, the _FILE_= variable is not written to the data set.

Restriction variable cannot be a previously defined variable. Make sure that the _FILE_= specification is the first occurrence of this variable in the DATA step. Do not set or change the length of _FILE_= variable with the LENGTH or ATTRIB statements. However, you can attach a format to this variable with the ATTRIB or FORMAT statement.

Interaction The maximum length of this character variable is the logical record length (LRECL) for the specified FILE statement. However, SAS does not open the file to know the LRECL until before the execution phase. Therefore, the designated size for this variable during the compilation phase is 32,767 bytes.

Tips Modification of this variable directly modifies the FILE statement's current output buffer. Any subsequent PUT statement for this FILE statement outputs the contents of the modified buffer. The _FILE_= variable accesses only the current output buffer of the specified FILE statement even if you use the N= option to specify multiple output buffers.

To access the contents of the output buffer in another statement without using the _FILE_= option, use the automatic variable _FILE_.

See “Updating the _FILE_ Variable” on page 88

Operating Environment Options
For descriptions of operating-environment-specific options in the FILE statement, see the SAS documentation for your operating environment.

Details

Overview
By default, PUT statement output is written to the SAS log. Use the FILE statement to route this output to either the same external file to which procedure output is written or to a different external file. You can indicate whether carriage-control characters should be added to the file. See the PRINT | NOPRINT option on page 86.

You can use the FILE statement in conditional (IF-THEN) processing because it is executable. You can also use multiple FILE statements to write to more than one external file in a single DATA step.

Operating Environment Information
Using the FILE statement requires operating-environment-specific information. See the SAS documentation for your operating environment before you use this statement.

You can use the Output Delivery System with the FILE statement to write DATA step results. For details, see the “FILE Statement for ODS” in SAS Output Delivery System: User’s Guide.

**Updating an External File in Place**

You can use the FILE statement with the INFILE and PUT statements to update an external file in place, updating either an entire record or only selected fields within a record. Follow these guidelines:

- Always place the INFILE statement first.
- Specify the same fileref or physical filename in the INFILE and FILE statements.
- Use options that are common to both the INFILE and FILE statements in the INFILE statement. (Any such options that are used in the FILE statement are ignored.)
- Use the SHAREBUFFERS option in the INFILE statement to allow the INFILE and FILE statements to use the same buffer, which saves CPU time and enables you to update individual fields instead of entire records.

**Accessing the Contents of the Output Buffer**

In addition to the _FILE_ variable, you can use the automatic _FILE_ variable to reference the contents of the current output buffer for the most recent execution of the FILE statement. This character variable is automatically retained and initialized to blanks. Like other automatic variables, _FILE_ is not written to the data set.

When you specify the _FILE_ = option in a FILE statement, this variable is also indirectly referenced by the automatic _FILE_ variable. If the automatic _FILE_ variable is present and you omit _FILE_ = in a particular FILE statement, then SAS creates an internal _FILE_ = variable for that FILE statement. Otherwise, SAS does not create the _FILE_ = variable for a particular FILE.

During execution and at the point of reference, the maximum length of this character variable is the maximum length of the current _FILE_ variable. However, because _FILE_ only references other variables whose lengths are not known until before the execution phase, the designated length is 32,767 bytes during the compilation phase. For example, if you assign _FILE_ to a new variable whose length is undefined, the default length of the new variable is 32,767 bytes. You cannot use the LENGTH statement and the ATTRIB statement to set or override the length of _FILE_. You can use the FORMAT statement and the ATTRIB statement to assign a format to _FILE_.

**Updating the _FILE_ Variable**

Like other SAS variables, you can update the _FILE_ variable. The following two methods are available:

- Use _FILE_ in an assignment statement.
- Use a PUT statement.

You can update the _FILE_ variable by using an assignment statement that has the following form.

\[
_FILE_ = <'string-in-quotation-marks'| character-expression>
\]

The assignment statement updates the contents of the current output buffer and sets the buffer length to the length of 'string-in-quotation-marks' or character-expression.
However, using an assignment statement does not affect the current column pointer of the PUT statement. The next PUT statement for this FILE statement begins to update the buffer at column 1 or at the last known location when you use the trailing @ in the PUT statement.

In the following example, the assignment statement updates the contents of the current output buffer. The column pointer of the PUT statement is not affected:

```sas
file print;
_file_ = '_FILE_';
put 'This is PUT';
```

SAS creates the following output: **This is PUT**

In this example,

```sas
file print;
_file_ = 'This is from FILE, sir.';
put @14 'both';
```

SAS creates the following output: **This is from both, sir.**

You can also update the _FILE_ variable by using a PUT statement. The PUT statement updates the _FILE_ variable because the PUT statement formats data in the output buffer and _FILE_ points to that buffer. However, by default SAS clears the output buffers after a PUT statement executes and outputs the current record (or N= block of records). Therefore, if you want to examine or further modify the contents of _FILE_ before it is output, include a trailing @ or @@ in any PUT statement (when N=1). For other values of N=, use a trailing @ or @@ in any PUT statement where the last line pointer location is on the last record of the record block. In the following example, when N=1

```sas
file ABC;
put 'Something' @;
Y = _file_||' is here';
file ABC;
put 'Nothing' ;
Y = _file_||' is here';
```

Y is first assigned **Something is here** then Y is assigned **is here**.

Any modification of _FILE_ directly modifies the current output buffer for the current FILE statement. The execution of any subsequent PUT statements for this FILE statement will output the contents of the modified buffer.

_FILE_ only accesses the contents of the current output buffer for a FILE statement, even when you use the N= option to specify multiple buffers. You can access all the N= buffers, but you must use a PUT statement with the # line pointer control to make the desired buffer the current output buffer.

**Comparisons**

- The FILE statement specifies the output file for PUT statements. The INFILE statement specifies the input file for INPUT statements.
- Both the FILE and INFILE statements enable you to use options that provide SAS with additional information about the external file being used.
- In the Program Editor, Log, and Output windows, the FILE command specifies an external file and writes the contents of the window to the file.
Examples

Example 1: Executing Statements When Beginning a New Page
This DATA step illustrates how to use the HEADER= option:

- **Write a report.** Use DATA _NULL_ to write a report rather than create a data set.
  ```sas
  data _null_; 
  set sprint; 
  by dept; 
  ```

- **Route output to the SAS output window.** Point to the header information. The PRINT fileref routes output to the same location as procedure output. HEADER= points to the label that precedes the statements that create the header for each page:
  ```sas
  file print header=newpage; 
  ```

- **Start a new page for each department:**
  ```sas
  if first.dept then put _page_; 
  put @22 salesrep @34 salesamt; 
  ```

- **Write a header on each page.** These statements execute each time a new page is begun. RETURN is necessary before the label and as the final statement in a labeled group:
  ```sas
  return; 
  newpage: 
  put @20 'Sales for 1989' / 
  @20 dept=; 
  return; 
  run; 
  ```

Example 2: Determining New Page by Lines Left on the Current Page
This DATA step demonstrates using the LINESLEFT= option to determine where the page break should occur, according to the number of lines left on the current page.

- **Write a report.** Use DATA _NULL_ to write a report rather than create a data set:
  ```sas
  data _null_; 
  set info; 
  ```

- **Route output to the standard SAS output window.** The PRINT fileref routes output to the same location as procedure output. LINESLEFT indicates that the variable REMAIN contains the number of lines left on the current page:
  ```sas
  file print linesleft=remain pagesize=20; 
  put @5 name @30 phone @35 bldg @37 room; 
  ```

- **Begin a new page when there are fewer than seven lines left on the current page.** Under this condition, PUT _PAGE_ begins a new page and positions the pointer at line 1:
  ```sas
  if remain<7 then put _page_; 
  run; 
  ```

Example 3: Arranging the Contents of an Entire Page
This example shows how to use N=PAGESIZE in a DATA step to produce a two-column telephone book listing, each column containing a name and a phone number:
• Create a report and write it to a SAS output window. Use DATA _NULL_ to write a report rather than create a data set. PRINT is the fileref. SAS uses carriage-control characters to write the output with the characteristics of a print file. N=PAGESIZE makes the entire page available to the output pointer:

```sas
data _null_; file 'external-file' print n=pagesize;
```

• Specify the columns for the report. This DO loop iterates twice on each DATA step iteration. The COL value is 1 on the first iteration and 40 on the second:

```sas
do col=1, 40;
```

• Write 20 lines of data. This DO loop iterates 20 times to write 20 lines in column 1. When finished, the outer loop sets COL equal to 40, and this DO loop iterates 20 times again, writing 20 lines of data in the second column. The values of LINE and COL, which are set and incremented by the DO statements, control where the PUT statement writes the values of NAME and PHONE on the page:

```sas
do line=1 to 20;
   set info;
   put #line @col name $20. +1 phone 4.;
end;
```

• After composing two columns of data, write the page. This END statement ends the outer DO loop. The PUT _PAGE_ writes the current page and moves the pointer to the top of a new page:

```sas
data _null_; length name $ 200;
```

**Example 4: Identifying the Current Output File**

This DATA step causes a file identification message to print in the log and assigns the value of the current output file to the variable MYOUT. The PUT statement, demonstrating the assignment of the proper value to MYOUT, writes the value of that variable to the output file:

```sas
data _null_; length myout $ 200;
   file file-specification filename=myout;
   put myout=;
   stop;
run;
```

The PUT statement writes a line to the current output file that contains the physical name of the file:

MYOUT=your-output-file

**Example 5: Dynamically Changing the Current Output File**

This DATA step uses the FILEVAR= option to dynamically change the currently opened output file to a new physical file.

• Write a report. Create a long character variable. Use DATA _NULL_ to write a report rather than create a data set. The LENGTH statement creates a variable with length long enough to contain the name of an external file:

```sas
data _null_;
   length name $ 200;
```
• Read an in-stream data line and assign a value to the NAME variable:
  
  input name $;

• Close the current output file and open a new one when the NAME variable changes.
  The file-specification is just a place holder; it can be any valid SAS name:

  file file-specification filevar=name mod;
  date = date();

• Append a log record to currently open output file:

  put 'records updated ' date date.;;

• Supply the names of the external files:

  datalines;
  external-file-1
  external-file-2
  external-file-3
  ;

**Example 6: When the Output Line Exceeds the Line Length of the Output File**

Because the combined lengths of the variables are longer than the output line (80 characters), this PUT statement automatically writes three separate records:

  file file-specification linesize=80;
  put name $ 1-50 city $ 71-90 state $ 91-104;

The value of NAME appears in the first record, CITY begins in the first column of the second record, and STATE in the first column of the third record.

**Example 7: Reading Data and Writing Text through a TCP/IP Socket**

This example shows reading raw data from a file through a TCP/IP socket. The NBYTE= option is used in the INFILE statement:

/* Start this first as the server */
filename serve socket ':5205' server
  recfm=s
  lrecl=25 blocksize=2500;
data _null_;
  nb=25;
  infile serve nbyte=nb;
  input text $char25.;
  put _all_;
run;

This example shows writing text to a file through a TCP/IP socket:

/* While the server test is running, */
/* continue with this as the client. */
filename client socket "&hstname:5205"
  recfm=s
  lrecl=25 blocksize=2500;
data _null_;  
  file client;
  put 'Some text to length 25...';
run;
Example 8: Specifying an Encoding When Writing to an Output 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.

```sas
libname myfiles 'SAS-library';
filename outfile 'external-file';
data _null_; 
  set myfiles.cars;
  file outfile encoding="utf-8";
  put Make Model Year;
run;
```

Example 9: Using the FTP Access Method to Write Data to an Excel
Spreadsheet
The example uses the FTP access method and the FILEVAR option to write data to
several Microsoft Excel worksheets.

```sas
data _null_; 
  do i = 1 to 3;
    sheet = cats('excel\[test-sheet.xlsx\]Sheet', i, '!r1c1:r10c2');
    file area ftp filevar=sheet;
    do x = 1 to 10;
      y = 2*x;
      put x y;
    end;
  end;
run;
```

See Also
• “How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?”
in SAS Language Reference: Concepts

Statements:
• “FILENAME Statement” on page 94
• “INFILE Statement” on page 198
• “LABEL Statement” on page 267
• “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server
  Administration Guide
• “PUT Statement” on page 335
• “RETURN Statement” on page 382
• “TITLE Statement” on page 412
• “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging
  with SAS
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
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 Language Reference: Concepts.

See: FILENAME Statement under Windows, UNIX, and 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.

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:

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

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

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 103

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

See “FILENAME Statement, DATAURL Access Method” on page 108

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 111

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 126

GTERM
indicates that the output device type is a graphics device that will receive 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 141

JMS
specifies a Java Message Service (JMS) destination.

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.

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 148

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 155

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 160

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 166

ZIP
specifies an access method that enables you to access ZIP files.
See “FILENAME Statement, ZIP Access Method” on page 174
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 148

*external-file* is the physical name of an external file. The physical name is the name that is recognized by the operating environment.

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

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.

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.

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


Examples: “Example 5: Specifying an Encoding When Reading an External File” on page 101

“Example 6: Specifying an Encoding When Writing to an External File” on page 102

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**

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 100.

**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 details 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 INFIL, 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 DLM options in the FILENAME statement, the ENCODING= option is a requirement because these options require 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, they 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 the following 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, specifying 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:

```sas
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';
infile 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.

```sas
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 details, 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.

```sas
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);
infile machine $ jansales febsales marsales;
```
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;
```
**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.

```sas
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” on page 73
- “%INCLUDE Statement” on page 191
- “INFILE Statement” on page 198
- “FILENAME Statement, CATALOG Access Method” on page 103
- “FILENAME Statement, ACTIVEMQ Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, DATAURL Access Method” on page 108
- “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
- “FILENAME Statement, FTP Access Method” on page 126
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166
- “FILENAME Statement, ZIP Access Method” on page 174
- “LIBNAME Statement” on page 274
- “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide

**System Options:**
- “LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide”
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.

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 the following:

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 fixed-record format. Data is transferred in image (binary) mode.  
- **P**  
is print format.  
- **S**  
is stream-record format. Data is transferred in image (binary) mode.  
- **V**  
is variable-record format (the default). In this format, records have varying lengths, and they are separated by newlines. Data is transferred in image (binary) mode.

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 105.

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 94
- “FILENAME Statement, ACTIVEMQ Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, DATAURL Access Method” on page 108
- “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
- “FILENAME Statement, FTP Access Method” on page 126
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166
- “FILENAME Statement, ZIP Access Method” on page 174

---

**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.

`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.

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

```sas
filename _temp1_ temp;
filename _temp2_ clipbrd;
proc contents data=Sashelp."Air" 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_ d d 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

FILENAME Statement, DATAURL Access Method
Enables you to read data from user-specified text.
FILENAME Statement, DATAURL Access Method

Validity: Anywhere
Category: Data Access

Syntax

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

Arguments

fileref
is a valid fileref.

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 the following 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 110

data-url-options
can be any of the following:

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

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

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

```sas
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.

```sas
filename in dataurl "data:,dGhpcyBpcyBhIGJhc2UgNjQgZW5jb2RpbcgZXhhbXBsZS4= " ;
data _NULL_; infile in; input; list; run;
```

NOTE: The infile IN is:
Filename=data:,dGhpcyBpcyBhIGJhc2UgNjQgZW5jb2RpbcgZXhhbXBsZS4=, 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.
FILENAME Statement, EMAIL (SMTP) Access Method

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

See Also

Statements:

- “FILENAME Statement” on page 94
- “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging with SAS
- “FILENAME Statement, CATALOG Access Method” on page 103
- “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
- “FILENAME Statement, FTP Access Method” on page 126
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in Application Messaging with SAS
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166
- “FILENAME Statement, ZIP Access Method” on page 174

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.

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 the following 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.pda (member)"
```

The attachment-options include the following:

**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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>CT= and TYPE=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>If SAS cannot determine a content type based on the filename and extension, the default value is text/plain.</td>
</tr>
</tbody>
</table>

**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. The following 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 124

**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 LRCEL is 256.

**NAME='filename'**

specifies a different name to be used for the specified attachment. You must enclose the value in quotation marks. The following 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.


**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>"`

**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>"
```

**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 only includes 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.

<table>
<thead>
<tr>
<th>Alias</th>
<th><code>CT=</code> and <code>TYPE=</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>text/plain</code></td>
</tr>
</tbody>
</table>

See “Example 5: Using the MESSAGE/RFC822 Content Type” on page 124

**DELIVERY_RECEIPT**
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. The default value for FROM= is the email address of the user who is running SAS. For example, specify this option when the person who is sending the message from SAS 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>"

Requirement The FROM option is required if the EMAILFROM system option is set. For more information, see the “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 in the FROM= option.

See “SENDER='sender-address’” on page 116

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 LRCEL 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=’reply-to-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>"

SENDER=’sender-address’
specifies the return email address for the message that is being sent. In the event that a message cannot be delivered, a notification is sent to the sender email address. The default value for 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:

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

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

See  “FROM=’from-address’” on page 115

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>"

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' | 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 the physical name must be enclosed 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:

```sas
put '!em_attach! /u/userid/opinion.txt';
put '!em_attach! ("C:\Status\June2001.txt" "C:\Status\July2001.txt")';
put '!em_attach! user.misc.pds{member}';
```

The attachment-options include the following:

- **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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT= and TYPE=</td>
<td>If SAS cannot determine a content type based on the filename and extension, the default value is text/plain.</td>
</tr>
</tbody>
</table>

- **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. The following example results in the attachment `home.html` being received as `index.htm`.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT=</td>
<td>TXT</td>
</tr>
</tbody>
</table>

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

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

  put '!em_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.


'!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>;

'!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>;

'!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 one 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!”';
```

**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 120

```
"!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!”';
```
"!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. In the event that
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>';

Interaction The !EM_SENDER!= address can be different from the FROM= or !
EM_FROM!= address. This 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 119

"!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>';

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 the following 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 the “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.

```plaintext
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 (the email options like TO=, CC=, SUBJECT=, CONTENT_TYPE=, ATTACH=), or to 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 117.

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 the following program. The email options are specified in the FILENAME statement:

```sas
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;
```

The following 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.

```sas
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!';
```
**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, the following program produces an email message and attachments that are dependent on the department to which the recipient belongs. In the program, the following occurs:

- 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.

```plaintext
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. Note that ODS HTML procedure output must be sent with the RECORD_SEPARATOR (RS) option set to NONE.

```plaintext
options emailsys=smtp;

filename outbox email  
  to='susan@site.com';
```

*FILENAME Statement, EMAIL (SMTP) Access Method* 123
Example 4: Creating and Emailing an Image
The following example illustrates how to create a GIF image and send it from SAS as an attachment to an email message.

```sas
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
The following example sends an email from an .mht file.

```sas
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
The following example creates an email that contains an embedded image.

```sas
options emailsys=smtp;

filename myemail email
to="Jim@acme.com"
```
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 94
- “FILENAME Statement, ACTIVEMQ Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, CATALOG Access Method” on page 103
- “FILENAME Statement, DATAURL Access Method” on page 108
- “FILENAME Statement, FTP Access Method” on page 126
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166
- “FILENAME Statement, ZIP Access Method” on page 174
- “LOCKDOWN Statement” in *SAS Intelligence Platform: Application Server Administration Guide*

System Options:
FILENAME Statement, FTP Access Method

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

Valid in: Anywhere
Category: Data Access
Restriction: 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.

Syntax

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

Arguments

fileref

is a valid fileref.

Note

In the third maintenance release of SAS 9.4, 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, then 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 137.

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.

### 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 449.

See
“PBSZ=protection-buffer-size” on page 132
“PROT=protection-level” on page 133

**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.

**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 140

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.


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 130

Example "Example 10: Reading and Writing from Directories" on page 140
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 129

LRECL= Irecl
where Irecl 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*:
```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 the “PWENCODE” in Base SAS Procedures Guide.

**Example**

“Example 6: Using an Encoded Password” on page 138
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 449.

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 127

“PROT=protection-level” on page 133

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 135 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 the following 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 449.

See  “AUTHTLS” on page 127

“PBSZ=protection-buffer-size” on page 132

**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. See S370V and S370VS below.

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 the following 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. Some examples are as follows:
    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 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 stream-record format. 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. 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 on page 209 in the INFILE statement.

V
    is variable-record format (the default). In this format, records have varying lengths, and they are transferred in text (stream) 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. See S370V and S370VS below.

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 syntax. Both anonymous and user ID validation are supported.

Example “Example 1: Retrieving a Directory Listing” on page 137

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 the third maintenance release for SAS 9.4, the FTP access method also supports Secure FTP (FTPS). FTPS 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:

```sas
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. Because quotation marks are required by the syntax, however, 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:

```sas
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. See RCMD= option on page 133 for more information about RCMD='site rdw'.

```
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 131 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. The quotation marks, however, 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;
```

The following output appears in the SAS log:

```
(sas001)TX1QYXNzMQ==
```
You can now use the entire encoded password string in your batch program.

```plaintext
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.

```plaintext
%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.

```plaintext
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:

```plaintext
filename inp ftp 'mat65.cpo' user='calvin'
pass="xxxx" host='mvshost1'
lrecl=80 recfm=f blocksize=8000
rcmd='site blocksize=800 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 Language Reference: Concepts 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 the FILEEXT option on page 129.

**Note:** The DIR option is not needed for some ODS destinations.

The following 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. The following 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_;
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 Intelligence Platform: Application Server Administration Guide.”

See Also

Environment Variables:
- “SAS_FTP_AUTHTLS Environment Variable” on page 449

Statements:
- “FILENAME Statement” on page 94
- “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging with SAS
- “FILENAME Statement, CATALOG Access Method” on page 103
- “FILENAME Statement, DATAURL Access Method” on page 108
- “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in Application Messaging with SAS
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166
- “FILENAME Statement, ZIP Access Method” on page 174
- “LIBNAME Statement” on page 274
- “LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide

System Options:
- “SSLCALISTLOC= System Option” in Encryption in SAS

Requirements: In the third maintenance release for Base SAS 9.4, 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. For more information, see SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

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. For more information, see SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

To use the FILENAME statement, Hadoop access method through WebHDFS by using the REST API, the SAS environment variable SAS_HADOOP_RESTFUL must be defined. In addition, the Hadoop configuration file hdfs-site.xml must include the properties for the WebHDFS location. For more information, see SAS 9.4 Hadoop Configuration Guide for Base SAS and SAS/ACCESS.

Syntax
FILENAME fileref HADOOP 'external-file' <hadoop-options>;

Required Arguments
fileref
is a valid fileref.

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 details 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 the following 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.

### 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.

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.

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 144

“NEW” on page 144

### 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.


**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 144

**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 144

**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 143

 **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 the 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:

 **S**  
 is stream-record format. Data is read in binary mode.

 **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  The NBYTE= option in the INFILE statement in the *SAS Statements: Reference*

 **F**  
 is fixed-record format. In this format, records have fixed lengths, and they are read in binary mode.

 **V**  
 is variable-record format (the default). In this format, records have varying lengths, and they are read in text (stream) mode.

 **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 the following 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.

```sas
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: Creating and Using a Configuration File

This example accesses the file acctdata.dat at site xxx.unx.sas.com. The configuration file is accessed from the “cfg” fileref assignment.

```sas
filename cfg 'U:/test.cfg';

data _null_;    
  file cfg;  
  input;  
  put _infile_;  
  datalines4;  
<configuration>  
<property>  
  <name>fs.default.name</name>
</property>
</configuration>
```
Example 3: 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.

```sas
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_;
   input a $1024. ;
   put a;
run;
```

Example 4: Using the CONCAT Option
This example uses the CONCAT option to read all members of DIRECTORY1 as if they are one file.

```sas
set=SAS_HADOOP_CONFIG_PATH="/u/hadoopcfg/cdh52p1";

filename foo hadoop '/directory1/' user='user' pass='apass' recfm=s lrecl=1024 concat;

data _null_;
   input a $1024. ;
   put a;
run;
```

See Also

Statements:
- “FILENAME Statement” on page 94
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.

**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 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 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.

\textit{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 149

\textbf{SFTP Options}

\textit{sftp-options} can be any of the following values:

\textbf{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 154

\textbf{CD='directory'}

issues a command that changes the working directory for the file transfer to the directory that you specify.

\textbf{DEBUG}

writes informational messages to the SAS log.

\textbf{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 will be ignored if the directory exists.

If the NEW option is omitted and you specify an invalid directory, then a new directory will not be created and you will 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 will not display files with leading periods, for example, .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.

Tip To display files without leading periods, for example, .xAuthority, use the LS= option.

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:

```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 will be 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 154

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 154

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 two record formats:
F
is fixed-record format. Thus, all records are of size LRECL with no line delimiters.

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 stream-record format. 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. 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 on page 209 in the INFILE statement.

V
is variable-record format (the default). In this format, records have varying lengths, and they are separated by newlines. Data is transferred in image (binary) mode.

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 will disable the SFTP access method.

You must have the applicable client software installed to use the SFTP access method. The SFTP access method supports only the following 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 will ensure that your SSH or SSHD configuration and key authentication is setup correctly.

**SFTP Access Methods and SFTP Prompts**

The SFTP access method supports only the following prompts. Changing the prompt will disable the SFTP access method.

- For OpenSSH:
  
  sftp>
  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:

```sas
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:
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`:

```sas
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.

```sas
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.

```sas
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.

```sas
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;
```
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>;

See Also


Statements:

• “FILENAME Statement” on page 94
• “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging with SAS
• “FILENAME Statement, CATALOG Access Method” on page 103
• “FILENAME Statement, DATAURL Access Method” on page 108
• “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
• “FILENAME Statement, FTP Access Method” on page 126
• “FILENAME Statement, Hadoop Access Method” on page 141
• “FILENAME Statement, JMS Access Method” in Application Messaging with SAS
• “FILENAME Statement, SOCKET Access Method” on page 155
• “FILENAME Statement, URL Access Method” on page 160
• “FILENAME Statement, WebDAV Access Method” on page 166
• “FILENAME Statement, ZIP Access Method” on page 174
• “LIBNAME Statement” on page 274
Arguments

**fileref**

is a valid fileref.

**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 will choose 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 157 under TCP/IP Options.

tcpip-options

specifies details that are specific to your operating system such as the number of connections that the server will accept.

**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 156

TCP/IP Options
tcpip-options can be any of the following 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.


**LRECL=** *lrecl*

where *lrecl* is the logical record length.

<table>
<thead>
<tr>
<th>Default</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>Alternatively, you can specify a global logical record length by using the “LRECL= System Option” in <em>SAS System Options: Reference</em>. 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.</td>
</tr>
</tbody>
</table>

**RECFM=** *recfm*

where *recfm* is one of three record formats:

**F**

is 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 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 on page 209 in the INFILE statement. |

**V**

is variable record format (the default).

| Tips | In this format, records have varying lengths, and they are transferred in text (stream) 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 will accept.

| 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 will |
continue to accept connections, one at a time, until \textit{conn-limit} has been reached.

\textbf{TERMSTR=}'\textit{eol-char}'

where \textit{eol-char} is the line delimiter to use when RECFM=V. There are three valid values:

- \textbf{CRLF} carriage return (CR) followed by line feed (LF).
- \textbf{LF} line feed only (the default).
- \textbf{NULL} NULL character (0x00).

\begin{tabular}{|l|}
\hline
\textbf{Default} & LF \\
\hline
\textbf{Restriction} & Use this option only when RECFM=V. \\
\hline
\end{tabular}

\section*{Details}

\textbf{The Basics}

A TCP/IP socket is a communication link between two applications. The \textit{server} application creates the socket and waits for a connection. The \textit{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.

\subsection*{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 will also automatically terminate.

\subsection*{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 \textit{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:

```sas
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:

```sas
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 94
- “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging with SAS
- “FILENAME Statement, CATALOG Access Method” on page 103
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.

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:
FILENAME Statement, URL Access Method 161

- http://hostname/file
- https://hostname/file
- http://hostname:portno/file
- https://hostname:portno/file

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

**URL Options**

$url-options$ can be any of the following 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 metadata object 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 the *SAS Intelligence Platform: Security Administration Guide*.

**BLOCKSIZE=**blocksize

where blocksize is 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 163

**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**

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 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” 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” 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 161

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
where recfm is one of three record formats:

F
is 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 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 on page 209 in the INFILE statement.

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

Tip Any record larger than LRECL is truncated.

Default V
TERMSTR='eol-char'
where eol-char is 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'
where username 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 will go 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 Statement, URL Access Method 165

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 the First 15 Records from a URL File
This example reads the first 15 records from a URL file and writes them to the SAS log
with a PUT statement:
filename foo url
   'http://support.sas.com/techsup/service_intro.html';
data _null_
   infile foo length=len;
   input record $varying200. len;
   put record $varying200. len;
   if _n_=15 then stop;
run;

See Also
• “Transport Layer Security (TLS)” in Encryption in SAS

Statements:
• “FILENAME Statement” on page 94
• “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging
   with SAS
• “FILENAME Statement, CATALOG Access Method” on page 103
• “FILENAME Statement, DATAURL Access Method” on page 108
• “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
• “FILENAME Statement, FTP Access Method” on page 126
• “FILENAME Statement, Hadoop Access Method” on page 141
• “FILENAME Statement, JMS Access Method” in Application Messaging with SAS
• “FILENAME Statement, SOCKET Access Method” on page 155
• “FILENAME Statement, SFTP Access Method” on page 148
• “FILENAME Statement, WebDAV Access Method” on page 166
• “FILENAME Statement, ZIP Access Method” on page 174
• “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
Restriction: Access to WebDAV servers is not supported on Open VMS.

Syntax

FILENAME fileref WEBDAV 'external-file' <webdav-options>;

Arguments

fileref
is a valid fileref.

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

WebDAV Options

webdav-options can be any of the following:

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 167

**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 the `FILEEXT` option on page 167 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 168

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 167

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.

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” 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 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 on page 209 in the INFILE statement.
is variable-record format (the default). In this format, records have varying
lengths, and they are transferred in text (stream) 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.

Details

The Basics
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 filenames, see the SAS documentation for
your operating environment.

Writing Files into SharePoint
You can use the FILENAME statement, WebDAV access method to write a file to a
SharePoint document library. To write a file to a SharePoint site, set the
SHAREPOINT_COMP_MODE environment variable.
This is the syntax for the environment variable.

**SHAREPOINT_COMP_MODE** *write-value*

*write-value* can be one of the following values:

- **1 | YES | TRUE**
  specifies that a file can be written to a SharePoint document library

- **0 | NO | FALSE**
  specifies that a file cannot be written to a SharePoint document library

Default 0

This environment variable can be set in a number of ways (for example, in code, in an autoexec file, or in a configuration file). Here is an example of setting the environment variable by using the **SET=** system option.

```plaintext
options set=sharepoint_comp_mode 1;
```

If you do not set the SharePoint environment variable, the SharePoint server fails to accept the data sent to it by SAS. No error is returned and the file that you request to be written is not available.

*Note:* SharePoint 2003 is not supported.

## Examples

### Example 1: Accessing a File at a Website
This example accesses the file *rawFile.txt* at site *www.mycompany.com*.

```plaintext
filename foo webdav 'https://www.mycompany.com/production/files/rawFile.txt'
  user='wong' pass='jd75ld';
data _null_;  infile foo;
  input a $80.;
run;
```

### Example 2: Using a Proxy Server
This example accesses the file *acctgfile.dat* by using the proxy server *otherwebsvr:80*.

```plaintext
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*.

```plaintext
filename writeit webdav
 "https://webserver.com:8443/webdav/testing/"
  dir user="webuser" pass=XXXXXXXXX;
data _null_;  file writeit(shoes);
```
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=XXXXXXXX;
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 filenames. 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=XXXXXXXX;
  options SASAUTOS=(writeit);
  /* expects a file called MYTEST.SAS */
  %MYTEST;
```

Example 6: Accessing a Lowercased Autocall Macro Member
The following 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
The following 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 filename was in uppercase, the reference to the filename 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
The following 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.

```sas
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
The following 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`.

```sas
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.

```sas
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.

```sas
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
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 file compression.

Syntax

FILENAME fileref ZIP 'external-file' <zip-options>;

Arguments

fileref

is a valid fileref.

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.

Note
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.

Tips
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.

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.

ZIP Options
zip-options can be any of the following:

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.

**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 176

**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 176

**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 global LRECL system option is 32767.

**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 on page 209 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 (stream) mode.

Interaction

Any record larger than LRECL is truncated.

Default

V

TERMSTR='eol-termination-character'

where the terminating character is the line character for Read operations and is the terminating character for a record Write operation. 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

The following example reads the test1.txt ZIP file member from the testzip ZIP file.

```sas
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

The following example writes the shoes file to the testzip ZIP file.

```sas
filename foo ZIP 'U:\directory1\testzip.zip';

data _null_;
Example 3: Reading from a Member of a Directory

This example reads the file shoes from the testing1ZIP file.

```sas
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:

- “FILENAME Statement” on page 94
- “FILENAME Statement, ACTIVEMQ Access Method” in Application Messaging with SAS
- “FILENAME Statement, CATALOG Access Method” on page 103
- “FILENAME Statement, DATAURL Access Method” on page 108
- “FILENAME Statement, EMAIL (SMTP) Access Method” on page 111
- “FILENAME Statement, FTP Access Method” on page 126
- “FILENAME Statement, Hadoop Access Method” on page 141
- “FILENAME Statement, JMS Access Method” in Application Messaging with SAS
- “FILENAME Statement, SFTP Access Method” on page 148
- “FILENAME Statement, SOCKET Access Method” on page 155
- “FILENAME Statement, URL Access Method” on page 160
- “FILENAME Statement, WebDAV Access Method” on page 166

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 417

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
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 417

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

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 (""), the text indicated is substituted into the footnote. If the footnote is enclosed in double quotation marks (""), the text is not substituted.

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 417.

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 417.

footnote j=left height=20pt color=red "Prepared on"
c='#FF9900'
footnote2 j=center color=blue height=24pt "&SYSDATE9"
footnote3 link='http://support.sas.com' "SAS"

See Also
• “TEMPLATE Procedure: Overview” in SAS Output Delivery System: Procedures Guide

Statements:
• “TITLE Statement” on page 412

FORMAT Statement
Associates formats with variables.
Valid in: DATA step or PROC step
Category: Information
Type: Declarative

Syntax
FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;
FORMAT variable-1 <...variable-n> format <DEFAULT=default-format>;
FORMAT variable-1 <...variable-n> format variable-1 <...variable-n> format;

Arguments
variable
names one or more variables for SAS to associate with a format. You must specify at least one variable.

Tip To disassociate a format from a variable, use the variable in a FORMAT statement without specifying a format in a DATA step or in PROC DATASETS. In a DATA step, place this FORMAT statement after the SET statement. See “Example 3: Removing a Format” on page 185. You can also use PROC DATASETS.

format
specifies the format that is listed for writing the values of the variables.

Tip Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon modifier in a subsequent PUT statement. For details about using a colon modifier, see “PUT Statement, List” on page 360.
DEFAULT=default-format

specifies a temporary default format for displaying the values of variables that are not listed in the FORMAT statement. These default formats apply only to the current DATA step; they are not permanently associated with variables in the output data set.

A DEFAULT= format specification applies to

• variables that are not named in a FORMAT or ATTRIB statement
• variables that are not permanently associated with a format within a SAS data set
• variables that are not written with the explicit use of a format.

Default

If you omit DEFAULT=, SAS uses BESTw. as the default numeric format and $w. as the default character format.

Restriction

Use this option only in a DATA step.

Tip

A DEFAULT= specification can occur anywhere in a FORMAT statement. It can specify either a numeric default, a character default, or both.

Example

“Example 1: Assigning Formats and Defaults” on page 183

Details

The FORMAT statement can use standard SAS formats or user-written formats that have been previously defined in PROC FORMAT. A single FORMAT statement can associate the same format with several variables, or it can associate different formats with different variables. If a variable appears in multiple FORMAT statements, SAS uses the format that is assigned last.

You use a FORMAT statement in the DATA step to permanently associate a format with a variable. SAS changes the descriptor information of the SAS data set that contains the variable. You can use a FORMAT statement in some PROC steps, but the rules are different. For more information, see Base SAS Procedures Guide.

Comparisons

Both the ATTRIB and FORMAT statements can associate formats with variables, and both statements can change the format that is associated with a variable. You can use the FORMAT statement in PROC DATASETS to change or remove the format that is associated with a variable. You can also associate, change, or disassociate formats and variables in existing SAS data sets through the windowing environment.

Examples

Example 1: Assigning Formats and Defaults

This example uses a FORMAT statement to assign formats and default formats for numeric and character variables. The default formats are not associated with variables in the data set but affect how the PUT statement writes the variables in the current DATA step.

```sas
data tstfmt;
    format W $char3.
```
The following output shows a partial listing from PROC CONTENTS, as well as the report that PROC PRINT generates.

**Output 2.3** Partial Listing from PROC CONTENTS and the PROC PRINT Report

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>Char</td>
<td>3</td>
<td>$CHAR3.</td>
</tr>
<tr>
<td>X</td>
<td>Num</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Num</td>
<td>8</td>
<td>10.3</td>
</tr>
<tr>
<td>Z</td>
<td>Char</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

**Output 2.4** PROC PRINT Report

The default formats apply to variables X and Z while the assigned formats apply to the variables W and Y.

The PUT statement produces this result:

```
----+----1----+----2
Goo 12.10
13.200
Howdy-do
```

**Example 2: Associating Multiple Variables with a Single Format**

This example uses the FORMAT statement to assign a single format to multiple variables.

```
data report;
```
input Item $ 1–6 Material $ 8–14 Investment 16–22 Profit 24–31;
format Item Material $upcase9. Investment Profit dollar15.2;
datalines;
shirts cotton 2256354 83952175
ties silk 498678 2349615
suits silk 9482146 69839563
belts leather 7693 14893
shoes leather 7936712 22964
;
run;
options pageno=1 nodate ls=80 ps=64;
proc print data=report;
  title 'Profit Summary: Kellam Manufacturing Company';
run;

Output 2.5  Results from Associating Multiple Variables with a Single Format

Profit Summary: Kellam Manufacturing Company

<table>
<thead>
<tr>
<th>Obs</th>
<th>Item</th>
<th>Material</th>
<th>Investment</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHIRTS</td>
<td>COTTON</td>
<td>$2,256,354.00</td>
<td>$83,952,175.00</td>
</tr>
<tr>
<td>2</td>
<td>TIES</td>
<td>SILK</td>
<td>$498,678.00</td>
<td>$2,349,615.00</td>
</tr>
<tr>
<td>3</td>
<td>SUITS</td>
<td>SILK</td>
<td>$9,482,146.00</td>
<td>$69,839,563.00</td>
</tr>
<tr>
<td>4</td>
<td>BELTS</td>
<td>LEATHER</td>
<td>$7,693.00</td>
<td>$14,893.00</td>
</tr>
<tr>
<td>5</td>
<td>SHOES</td>
<td>LEATHER</td>
<td>$7,936,712.00</td>
<td>$22,964.00</td>
</tr>
</tbody>
</table>

Example 3: Removing a Format
This example disassociates an existing format from a variable in a SAS data set. The
order of the FORMAT and the SET statements is important.

data rtest;
  set rtest;
  format x;
run;

See Also
• “DATASETS” in Base SAS Procedures Guide

Statements:
• “ATTRIB Statement” on page 27

GO TO Statement
Directs program execution immediately to the statement label that is specified and, if followed by a
RETURN statement, returns execution to the beginning of the DATA step.
Valid in: DATA step
Category: Control
Type: Executable
Alias: GOTO

Syntax

GO TO label;

Arguments

label

specifies a statement label that identifies the GO TO destination. The destination must be within the same DATA step. You must specify the label argument.

Comparisons

The GO TO statement and the LINK statement are similar. However, a GO TO statement is often used without a RETURN statement, whereas a LINK statement is usually used with an explicit RETURN statement. The action of a subsequent RETURN statement differs between the GO TO and LINK statements. A RETURN statement after a LINK statement returns execution to the statement that follows the LINK statement. A RETURN after a GO TO statement returns execution to the beginning of the DATA step (unless a LINK statement precedes the GO TO statement. In that case, execution continues with the first statement after the LINK statement).

GO TO statements can often be replaced by DO-END and IF-THEN/ELSE programming logic.

Example: Using a RETURN Statement with the GO TO Statement

Use the GO TO statement as shown here.

• In this example, if the condition is true, the GO TO statement instructs SAS to jump to a label called ADD and to continue execution from there. If the condition is false, SAS executes the PUT statement and the statement that is associated with the GO TO label:

```sas
data info;
  input x;
  if 1<=x<=5 then go to add;
  put x=;
  add: sumx+x;
  datalines;
7
6
323
;
```

Because every DATA step contains an implied RETURN at the end of the step, program execution returns to the top of the step after the sum statement is executed. Therefore, an explicit RETURN statement at the bottom of the DATA step is not necessary.
• If you do not want the Sum statement to execute for observations that do not meet the condition, rewrite the code and include an explicit return statement.

data info;
  input x;
  if 1<=x<=5 then go to add;
  put x=;
  return;
  /* SUM statement not executed */
  /* if x<1 or x>5 */
  add: sumx+x;
  datalines;
7
  6
  323
;

See Also

Statements:
• “DO Statement” on page 61
• “label: Statement” on page 269
• “LINK Statement” on page 293
• “RETURN Statement” on page 382

**IF Statement, Subsetting**

Continues processing only those observations that meet the condition of the specified expression.

**Valid in:** DATA step  
**Category:** Action  
**Type:** Executable

**Syntax**

`IF expression;`

**Arguments**

`expression`  
is any SAS expression.

**Details**

**The Basics**
The subsetting IF statement causes the DATA step to continue processing only those raw data records or those observations from a SAS data set that meet the condition of the expression that is specified in the IF statement. That is, if the expression is true for the observation or record (its value is neither 0 nor missing), SAS continues to execute statements in the DATA step and includes the current observation in the data set. The
resulting SAS data set or data sets contain a subset of the original external file or SAS data set.

If the expression is false (its value is 0 or missing), no further statements are processed for that observation or record, the current observation is not written to the data set, and the remaining program statements in the DATA step are not executed. SAS immediately returns to the beginning of the DATA step because the subsetting IF statement does not require additional statements to stop processing observations.

**Using the Equivalent of the CONTAINS and LIKE Operators in an IF Statement**

The LIKE operator in a WHERE clause matches patterns in words. To get the equivalent result in an IF statement, the "=:'" operator can be used. This matches patterns that occur at the beginning of a string. Here is an example:

```sas
data test;
  input name $;
datalines;
  John
  Diana
  Diane
  Sally
  Doug
  David
;
run;

data test;
  set test;
  if name =: 'D';
run;

proc print;
run;
```

The CONTAINS operator in a WHERE clause checks for a character string within a value. To get the equivalent result in an IF statement, the INDEX function can be used. Here is an example:

```sas
data test;
  set test;
  if index(name,'ian') ge 1;
run;

proc print;
run;
```

**Comparisons**

- The subsetting IF statement is equivalent to this IF-THEN statement:

  ```sas
  if not (expression)
  then delete;
  ```

- When you create SAS data sets, use the subsetting IF statement when it is easier to specify a condition for including observations. When it is easier to specify a condition for excluding observations, use the DELETE statement.
The subsetting IF and the WHERE statements are not equivalent. The two statements work differently and produce different output data sets in some cases. The most important differences are summarized as follows:

- The subsetting IF statement selects observations that have been read into the program data vector. The WHERE statement selects observations before they are brought into the program data vector. The subsetting IF might be less efficient than the WHERE statement because it must read each observation from the input data set into the program data vector.

- The subsetting IF statement and WHERE statement can produce different results in DATA steps that interleave, merge, or update SAS data sets.

- When the subsetting IF statement is used with the MERGE statement, the SAS System selects observations after the current observations are combined. When the WHERE statement is used with the MERGE statement, the SAS System applies the selection criteria to each input data set before combining the current observations.

- The subsetting IF statement can select observations from an existing SAS data set or from raw data that are read with the INPUT statement. The WHERE statement can select observations only from existing SAS data sets.

- The subsetting IF statement is executable; the WHERE statement is not.

**Example: Limiting Observations**

- This example results in a data set that contains only those observations with the value 'F' for the variable SEX:
  
  ```sas
  if sex='F';
  ```

- This example results in a data set that contains all observations for which the value of the variable AGE is not missing or 0:

  ```sas
  if age;
  ```

**See Also**

- “WHERE-Expression Processing” in *SAS Language Reference: Concepts*

**Data Set Options:**

- “WHERE= Data Set Option” in *SAS Data Set Options: Reference*

**Statements:**

- “DELETE Statement” on page 55
- “IF-THEN/ELSE Statement” on page 189
- “WHERE Statement” on page 427

---

**IF-THEN/ELSE Statement**

Executes a SAS statement for observations that meet specific conditions.

Valid in:  
DATA step

Category:  
Control
Type: Executable

Syntax

IF expression THEN statement;
<ELSE statement;>

Arguments

expression

is any SAS expression and is a required argument.

statement

can be any executable SAS statement or DO group.

Details

SAS evaluates the expression in an IF-THEN statement to produce a result that is either nonzero, zero, or missing. A nonzero and nonmissing result causes the expression to be true; a result of zero or missing causes the expression to be false.

If the conditions that are specified in the IF clause are met, the IF-THEN statement executes a SAS statement for observations that are read from a SAS data set, for records in an external file, or for computed values. An optional ELSE statement gives an alternative action if the THEN clause is not executed. The ELSE statement, if used, must immediately follow the IF-THEN statement.

Using IF-THEN statements without the ELSE statement causes SAS to evaluate all IF-THEN statements. Using IF-THEN statements with the ELSE statement causes SAS to execute IF-THEN statements until it encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

Note: For greater efficiency, construct your IF-THEN/ELSE statement with conditions of decreasing probability.

Comparisons

• Use a SELECT group rather than a series of IF-THEN statements when you have a long series of mutually exclusive conditions.

• Use subsetting IF statements, without a THEN clause, to continue processing only those observations or records that meet the condition that is specified in the IF clause.

Example: Different Ways of Specifying the IF-THEN/ELSE Statements

These examples show different ways of specifying the IF-THEN/ELSE statement.

• if x then delete;

• if status='OK' and type=3 then count+1;

• if age ne agecheck then delete;

• if x=0 then
  if y ne 0 then put 'X ZERO, Y NONZERO';
  else put 'X ZERO, Y ZERO';
  else put 'X NONZERO';
• if answer=9 then
do;
  answer=.;
  put 'INVALID ANSWER FOR ' id=;
end;
else
do;
  answer=answer10;
  valid+1;
end;

• data region;
  input city $ 1-30;
  if city='New York City'
    or city='Miami' then
    region='ATLANTIC COAST';
  else if city='San Francisco'
    or city='Los Angeles' then
    region='PACIFIC COAST';
  datalines;
  ...more data lines...

See Also

Statements:
• “DO Statement” on page 61
• “IF Statement, Subsetting” on page 187
• “SELECT Statement” on page 391

%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)

\(<\langle SOURCE2 > <S2=length> <operating\text{-}environment\text{-}options > >;\)

Arguments

source(s)
describes the location of the information that you want to access with the
%INCLUDE statement. There are three possible sources:
<table>
<thead>
<tr>
<th>Source</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>file-specification</td>
<td>specifies an external file</td>
</tr>
<tr>
<td>internal-lines</td>
<td>specifies lines that are entered earlier in the same SAS job or session</td>
</tr>
<tr>
<td>keyboard-entry</td>
<td>specifies statements or data lines that you enter directly from the keyboard</td>
</tr>
</tbody>
</table>

**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.

  **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, space.

  This example instructs SAS to include the files testcode1.sas, testcode2.sas and testcode3.txt. These files are located in 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 are entered earlier in the same SAS job or session.

To include internal lines, use any of the following:

- \( n \) 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:

```
proc print;
   %include *
run;
```

To resume processing the original source program, enter a %RUN statement from the keyboard.
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 option in the %INCLUDE statement, it overrides the 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- 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 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. See the documentation for your operating environment for more information about additional software features and methods of referring to and accessing your files. See your documentation before you run the examples for this statement.

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.

When Useful
The %INCLUDE statement is most often used when running SAS in interactive line mode, noninteractive mode, or batch mode. Although you can use the %INCLUDE statement when you run SAS using windows, it might be more practical to use the INCLUDE and RECALL commands to access data lines and program statements, and submit these lines again.

Rules for Using %INCLUDE
• You can specify any number of sources in a %INCLUDE statement, and you can mix the types of included sources. Note, however, that 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.
• The %INCLUDE statement must begin at a statement boundary. That is, it must be the first statement in a SAS job or must immediately follow a semicolon ending another statement. A %INCLUDE statement cannot immediately follow a DATALINES, DATALINES4, CARDS, or CARDS4 statement (or PARMCARDS or PARMCARDS4 statement in procedures that use those statements). However, you can include data lines with the %INCLUDE statement using one of these methods:

• Make the DATALINES, DATALINES4, or CARDS, CARDS4 statement the first line in the file that contains the data.

• Place the DATALINES, DATALINES4, or CARDS, CARDS4 statement in one file, and the data lines in another file. Use both sources in a single %INCLUDE statement.

The %INCLUDE statement can be nested within a file that has been accessed with %INCLUDE. The maximum number of nested %INCLUDE statements that you can use depends on system-specific limitations of your operating environment (such as available memory or the number of files that you can have open concurrently).

• Because %INCLUDE is a global statement and global statements are not executable, the %INCLUDE statement cannot be used in conditional logic.

• 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 and includes it in a program to be written later. This program is stored in a file named MYFILE:

```sas
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:

```sas
%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:

```sas
filename in1 'MYFILE';
```

You can later access MYFILE with the fileref IN1:

```sas
%inc in1;
```
• If you want to use many files that are stored in a directory, PDS, or MACLIB (or whatever your operating environment calls an aggregate storage location), you can assign the fileref to the larger storage unit and then specify the filename. For example, this FILENAME statement assigns the fileref STORAGE to an aggregate storage location:

```sas
filename storage
   'aggregate-storage-location';
```

You can later include a file using this statement:

```sas
%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. The following %INCLUDE statement demonstrates this method:

```sas
%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.

```sas
%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:

```sas
%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:

```sas
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. You can enter statements such as

```sas
where month= 'January';
title 'Data for month of January';
```

After you enter statements, you can use %RUN to resume processing by entering

```sas
%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.

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

**See Also**

**Statements:**

- “%LIST Statement” on page 297
- “%RUN Statement” on page 384

---

**INFILE Statement**

Specifies an external file to read with an INPUT statement.

**Valid in:** DATA Step

**Category:** File-Handling

**Type:** Executable

**Restriction:** When SAS is in a locked-down state, the INFILE statement is not available for files that are not in the locked-down path list. For more information, see “SAS Processing Restrictions for Servers in a Locked-Down State” in SAS Language Reference: Concepts.

**Operating environment:** The INFILE statement contains operating environment-specific material. See the SAS documentation for your operating environment before using this statement.

**See:** INFILE Statement under Windows, UNIX, and z/OS

**Syntax**

```
INFILE file-specification <device-type> <options> <operating-environment-options>
```

```
INFILE DBMS-specifications;
```

**Arguments**

- `file-specification`
  
  identifies the source of the input data records, which is an external file or instream data. `File-specification` can have these forms:

  - `'external-file'`
    
    specifies the physical name of an external file. The physical name is the name that the operating environment uses to access the file.

  - `fileref`
    
    specifies the fileref of an external file.
### Requirements
You must have previously associated the fileref with an external file in a FILENAME statement, FILENAME function, or an appropriate operating environment command.

### See
“FILENAME Statement” on page 94

### fileref(file)
**specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location.**

### Requirements
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.

You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, or an appropriate operating environment command.

### Operating environment
Different operating environments call an aggregate grouping of files by different names, such as a directory, a MACLIB, or a partitioned data set. For details about how to specify external files, see the SAS documentation for your operating environment.

### See
“FILENAME Statement” on page 94

### CARDS | CARDS4
for a definition, see DATALINES on page 199.

### Alias
DATALINES | DATALINES4

### DATALINES | DATALINES4
**specifies that the input data immediately follows the DATALINES or DATALINES4 statement in the DATA step. Using DATALINES enables you to use the INFILE statement options to control how the INPUT statement reads instream data lines.**

### Alias
CARDS | CARDS4

### Example
“Example 1: Changing How Delimiters Are Treated” on page 219

### Tip
You can verify the existence of file-specification by using the SYSERR macro variable if the ERRORCHECK option is set to STRICT.

### 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:**

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

### 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
CATALOG
specifies the CATALOG access method.

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 For a complete list of options that are available with the CATALOG access method, see the “FILENAME Statement, CATALOG Access Method” on page 103.

CLIPBOARD
specifies the CLIPBOARD access method.

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 For a complete list of options that are available with the CLIPBOARD access method, see the “FILENAME Statement, CLIPBOARD Access Method” on page 106.

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.

FTP
specifies the FTP access method.

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 For a complete list of options that are available with the FTP access method, see the “FILENAME Statement, FTP Access Method” on page 126.

Example infile dummy ftp user='myuid' pass='xxxx' filevar=file_to_read;

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

HADOOP
specifies the Hadoop access method

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 For a complete list of options that are available with the Hadoop access method, see the “FILENAME Statement, Hadoop Access Method” on page 141.
JMS
specifies a Java Message Service (JMS) destination.

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.

SFTP
specifies the SFTP access method.

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 For a complete list of options that are available with the SFTP access method, see the “FILENAME Statement, SFTP Access Method” on page 148.

SOCKET
specifies the SOCKET access method.

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 For a complete list of options that are available with the SOCKET access method, see the “FILENAME Statement, SOCKET Access Method” on page 155.

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 that are 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 PRINTPATH options control which Universal Printer is used and the destination of the output.

URL
specifies the URL access method.
**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**
For a complete list of options that are available with the URL access method, see the “FILENAME Statement, URL Access Method” on page 160.

**WEBDAV**
specifies the WEBDAV access method.

**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**
For a complete list of options that are available with the WEBDAV access method, see the “FILENAME Statement, WebDAV Access Method” on page 166.

**Alias**
DEVICE= *device-type*

**Default**
DISK

**Requirement**
*device-type* or DEVICE= *device-type* must immediately follow *file-specification* 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.

---

**INFILE Options**

**BLKSIZE= *block-size***
specifies the block size of the input file.

**Default**
Dependent on the operating environment. For details, see the SAS documentation for your operating environment.

**COLUMN= *variable***
names a variable that SAS uses to assign the current column location of the input pointer. Like automatic variables, the COLUMN= variable is not written to the data set.

**Alias**
COL=

**See**
LINE= on page 207

**Example**
“Example 8: Listing the Pointer Location” on page 224

**DELIMITER= *delimiter(s)***
specifies an alternate delimiter (other than a blank) to be used for LIST input, where *delimiter(s)* can be one of the following items.

*list-of-delimiting-characters*
specifies one or more characters to read as delimiters.

**Requirement**
Enclose the list of characters in quotation marks.
**Example**  
“Example 1: Changing How Delimiters Are Treated” on page 219

<table>
<thead>
<tr>
<th><strong>character-variable</strong></th>
<th>specifies a character variable whose value becomes the delimiter.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alias</strong></td>
<td>DLM=</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>blank space</td>
</tr>
<tr>
<td><strong>Tips</strong></td>
<td>The delimiter is case sensitive.</td>
</tr>
<tr>
<td></td>
<td>Some common delimiters are the comma (,), vertical pipe (</td>
</tr>
<tr>
<td></td>
<td>semicolon (;), and tab. The tab is specified by a hexadecimal</td>
</tr>
<tr>
<td></td>
<td>character. Under UNIX and Windows, the value is '09'x. Under</td>
</tr>
<tr>
<td></td>
<td>z/OS, the value is '05'x.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Reading Delimited Data” on page 214, DLMSTR= on page 203, and</td>
</tr>
<tr>
<td></td>
<td>“DSD (delimiter-sensitive data)” on page 204</td>
</tr>
</tbody>
</table>

**DLMSTR= delimiter**  
specifies a character string as an alternate delimiter (other than a blank) to be used for LIST input, where delimiter can be one of the following items.

<table>
<thead>
<tr>
<th><strong>'delimiting-string'</strong></th>
<th>specifies a character string to read as a delimiter.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirement</strong></td>
<td>Enclose the string in quotation marks.</td>
</tr>
</tbody>
</table>

**Example**  
“Example 1: Changing How Delimiters Are Treated” on page 219

<table>
<thead>
<tr>
<th><strong>character-variable</strong></th>
<th>specifies a character variable whose value becomes the delimiter.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>blank space</td>
</tr>
<tr>
<td><strong>Interactions</strong></td>
<td>If you specify more than one DLMSTR= option in the INFILE</td>
</tr>
<tr>
<td></td>
<td>statement, the DLMSTR= option that is specified last is used. If</td>
</tr>
<tr>
<td></td>
<td>you specify both the DELIMITER= and DLMSTR= options, the option</td>
</tr>
<tr>
<td></td>
<td>that is specified last is used.</td>
</tr>
<tr>
<td></td>
<td>If you specify RECFM=N, make sure that the LRECL is large enough</td>
</tr>
<tr>
<td></td>
<td>to hold the largest input item. Otherwise, it might be possible</td>
</tr>
<tr>
<td></td>
<td>for the delimiter to be split across the record boundary.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>The delimiter is case sensitive. To make the delimiter case</td>
</tr>
<tr>
<td></td>
<td>insensitive, use the DLMSOPT='I' option.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Reading Delimited Data” on page 214, DELIMITER= on page 202,</td>
</tr>
<tr>
<td></td>
<td>DLMSOPT= on page 204, and DSD on page 204</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 1: Changing How Delimiters Are Treated” on page 219</td>
</tr>
</tbody>
</table>
DLMSOPT= 'option(s)'
specifies parsing options for the DLMSTR= option where option(s) can be one of the following values.

I
specifies that case-insensitive comparisons are done.

T
specifies that trailing blanks of the string delimiter are removed.

Tips  The $T$ option is useful when you use a variable as the delimiter string.

You can specify either $I$, $T$, or both.

Requirement  The DLMSOPT= option has an effect only when used with the DLMSTR= option.

See  DLMSTR= on page 203

Example  “Example 1: Changing How Delimiters Are Treated” on page 219

DSD (delimiter-sensitive data)
specifies that when data values are enclosed in quotation marks, delimiters within the value are treated as character data. The DSD option changes how SAS treats delimiters when you use LIST input and sets the default delimiter to a comma. When you specify DSD, SAS treats two consecutive delimiters as a missing value and removes quotation marks from character values.

Interaction  Use the DELIMITER= or DLMSTR= option to change the delimiter.

Tip  Use the DSD option and LIST input to read a character value that contains a delimiter within a string that is enclosed in quotation marks. The INPUT statement treats the delimiter as a valid character and removes the quotation marks from the character string before the value is stored. Use the tilde (~) format modifier to retain the quotation marks.

See  “Reading Delimited Data” on page 214, DELIMITER= on page 202, and DLMSTR= on page 203

Examples  “Example 1: Changing How Delimiters Are Treated” on page 219
“Example 2: Handling Missing Values and Short Records with List Input” on page 221

ENCODING= 'encoding-value'
specifies the encoding to use when reading from 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.

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

Example  “Example 11: Specifying an Encoding When Reading an External File” on page 227

END=variable
specifies a variable that SAS sets to 1 when the current input data record is the last in the input file. Until SAS processes the last data record, the END= variable is set to 0. Like automatic variables, this variable is not written to the data set.

Restriction  You cannot use the END= option with the UNBUFFERED option, the DATALINES or DATALINES4 statement, or an INPUT statement that reads multiple input data records.

Tip  Use the option EOF= on page 205 when END= is invalid.

Example  “Example 5: Reading from Multiple Input Files” on page 223

EOF=label
specifies a statement label that is the object of an implicit GO TO when the INFILE statement reaches end of file. When an INPUT statement attempts to read from a file that has no more records, SAS moves execution to the statement label indicated.

Interaction  Use EOF= instead of the END= option with the UNBUFFERED option, the DATALINES or DATALINES4 statement, an INPUT statement that reads multiple input data records.

Tip  The EOF= option is useful when you read from multiple input files sequentially.

See  END= on page 205, EOV= on page 205, and UNBUFFERED on page 211

EOV=variable
specifies a variable that SAS sets to 1 when the first record in a file in a series of concatenated files is read. The variable is set only after SAS encounters the next file. Like automatic variables, the EOV= variable is not written to the data set.

Tip  Reset the EOV= variable back to 0 after SAS encounters each boundary.

See  END= on page 205 and EOF= on page 205

EXPANDTABS | NOEXPANDTABS
specifies whether to expand tab characters to the standard tab setting, which is set at 8-column intervals that start at column 9.

Default  NOEXPANDTABS

Tip  EXPANDTABS is useful when you read data that contains the tab character that is native to your operating environment.

FILENAME=variable
specifies a variable that SAS sets to the physical name of the currently opened input file. In a series of concatenated files, the variable is updated only after SAS encounters the next file. Like automatic variables, the FILENAME= variable is not written to the data set.

Tip  Use a LENGTH statement to make the variable length long enough to contain the value of the filename.
FILEVAR=variable

specifies a variable whose change in value causes the INFILE statement to close the current input file and open a new one. When the next INPUT statement executes, it reads from the new file that the FILEVAR= variable specifies. Like automatic variables, this variable is not written to the data set.

Restriction

The FILEVAR= variable must contain a character string that is a physical filename.

Interaction

When you use the FILEVAR= option, the file-specification is just a placeholder, not an actual filename or a fileref that has been previously assigned to a file. SAS uses this placeholder for reporting processing information to the SAS log. It must conform to the same rules as a fileref.

Tips

Use FILEVAR= to dynamically change the currently opened input file to a new physical file.

When using FILEVAR=, it is not possible to know whether the input file that is currently open is the last file or not. When the DATA step comes to an end-of-file marker or the end of all open data sets, it performs an orderly shutdown. In addition, if you use FILEVAR with FIRSTOBS, a file with only a header record in a series of files triggers a normal shutdown of the DATA step. The shutdown occurs because SAS reads beyond the end-of-file marker and the DATA step terminates. You can use the EOF= option to avoid the shutdown.

FIRSTOBS=record-number

specifies a record number that SAS uses to begin reading input data records in the input file.

Default

1

Tips

Use FIRSTOBS= with OBS= to read a range of records from the middle of a file.

Use FIRSTOBS=2 to skip a header record in a file.

Example

This statement processes record 50 through record 100:
infile file-specification firstobs=50 obs=100;

FLOWOVER

causes an INPUT statement to continue to read the next input data record if it does not find values in the current input line for all the variables in the statement. FLOWOVER is the default behavior of the INPUT statement.

See

“Reading Past the End of a Line” on page 217, MISSOVER on page 208, STOPOVER on page 210, and TRUNCOVER on page 211
LENGTH=variable
specifies a variable that SAS sets to the length of the current input line. SAS does not assign the variable a value until an INPUT statement executes. Like automatic variables, the LENGTH= variable is not written to the data set.

Tip  This option in conjunction with the $VARYING informat is useful when the field width varies.

Examples  “Example 4: Reading Files That Contain Variable-Length Records” on page 222

“Example 7: Truncating Copied Records” on page 224

LINE=variable
specifies a variable that SAS sets to the line location of the input pointer in the input buffer. Like automatic variables, the LINE= variable is not written to the data set.

Range  1 to the value of the N= option

Interaction  The value of the LINE= variable is the current relative line number within the group of lines that is specified by the N= option or by the #n line pointer control in the INPUT statement.

See  COLUMN= on page 202 and N= on page 208

Example  “Example 8: Listing the Pointer Location” on page 224

LINESIZE=line-size
specifies the record length that is available to the INPUT statement.

Alias  LS=

Range  up to 32767

Interaction  If an INPUT statement attempts to read past the column that is specified by the LINESIZE= option, then the action that is taken depends on whether the FLOWOVER, MISSOVER, SCANOVER, STOPOVER, or TRUNCOVER option is in effect. FLOWOVER is the default.

Operating environment  Values for line-size are dependent on the operating environment record size. For details, see the SAS documentation for your operating environment.

Tip  Use LINESIZE= to limit the record length when you do not want to read the entire record.

Example  If your data lines contain a sequence number in columns 73 through 80, then use this INFILE statement to restrict the INPUT statement to the first 72 columns:

```
infile file-specification linesize=72;
```

LRECL=logical-record-length
specifies the logical record length.

Default  Dependent on the file characteristics of your operating environment
Restriction
LRECL is not valid when you use the DATALINES file specification.

Interaction
Alternatively, you can specify a global logical record length by using the “LRECL= System Option” in SAS System Options: Reference. 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 logical-record-length are dependent on the operating environment. For details, see the SAS documentation for your operating environment.

Tip
LRECL= specifies the physical line length of the file. LINESIZE= tells the INPUT statement how much of the line to read.

MISSOVER
prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement. When an INPUT statement reaches the end of the current input data record, variables without any values assigned are set to missing.

Tip
Use MISSOVER if the last field or fields might be missing and you want SAS to assign missing values to the corresponding variable.

See
“Reading Past the End of a Line” on page 217, FLOWOVER on page 206, SCANOVER on page 210, STOPOVER on page 210, and TRUNCOVER on page 211

Example
“Example 2: Handling Missing Values and Short Records with List Input” on page 221

N=available-lines
specifies the number of lines that are available to the input pointer at one time.

Default
The highest value following a # pointer control in any INPUT statement in the DATA step. If you omit a # pointer control, then the default value is 1.

Interaction
This option affects only the number of lines that the pointer can access at a time; it has no effect on the number of lines an INPUT statement reads.

Tips
When you use # pointer controls in an INPUT statement that are less than the value of N=, you might get unexpected results. To prevent unexpected results, include a # pointer control that equals the value of the N= option. Here is an example:

```
infile 'external file' n=5;
input #2 name : $25. #3 job : $25. #5;
```

The INPUT statement includes a #5 pointer control, even though no data is read from that record.

Example
“Example 8: Listing the Pointer Location” on page 224
NBYTE=variable
specifies the name of a variable that contains the number of bytes to read from a file when you are reading data in stream record format (RECFM=S in the FILENAME statement).

Default
The LRECL value of the file

Interaction
If the number of bytes to read is set to -1, then the FTP and SOCKET access methods return the number of bytes that are currently available in the input buffer.

See
The RECFM= option on page 157 in the FILENAME statement, SOCKET access method, and the RECFM= option on page 134 in the FILENAME statement, FTP access method

OBS=record-number | MAX
record-number
specifies the record number of the last record to read in an input file that is read sequentially.

MAX
specifies the maximum number of observations to process, which is at least as large as the largest signed, 32-bit integer. The absolute maximum depends on your host operating environment.

Default
MAX

Tip
Use OBS= with FIRSTOBS= to read a range of records from the middle of a file.

Example
This statement processes only the first 100 records in the file:

```
infile file-specification obs=100;
```

PAD |NOPAD
controls whether SAS pads the records that are read from an external file with blanks to the length that is specified in the LRECL= option.

Default
NOPAD

See
LRECL= option on page 207

PRINT | NOPRINT
specifies whether the input file contains carriage-control characters.

Tip
To read a file in a DATA step without having to remove the carriage-control characters, specify PRINT. To read the carriage-control characters as data values, specify NOPRINT.

RECFM=record-format
specifies the record format of the input 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 details, see the SAS documentation for your operating environment.
SCANOVER causes the INPUT statement to scan the input data records until the character string that is specified in the @’character-string’ expression is found.

Interaction The MISSOVER, TRUNCOVER, and STOPOVER options change how the INPUT statement behaves when it scans for the @’character-string’ expression and reaches the end of the record. By default (FLOWOVER option), the INPUT statement scans the next record while these other options cause scanning to stop.

Tip It is redundant to specify both SCANOVER and FLOWOVER.

See “Reading Past the End of a Line” on page 217, FLOWOVER on page 206, MISSOVER on page 208, STOPOVER on page 210, and TRUNCOVER on page 211

Example “Example 3: Scanning Variable-Length Records for a Specific Character String” on page 221

SHAREBUFFERS specifies that the FILE statement and the INFILE statement share the same buffer.

Alias SHAREBUFS

Tips Use SHAREBUFFERS with the INFILE, FILE, and PUT statements to update an external file in place. Updating an external file in place saves CPU time because the PUT statement output is written straight from the input buffer instead of the output buffer.

Use SHAREBUFFERS to update specific fields in an external file instead of an entire record.

Example “Example 6: Updating an External File” on page 224

CAUTION When using SHAREBUFFERS, RECFM=V, and _INFILE_, use caution if you read a record with one length and update the file with a record of a different length. The length of the record can change by modifying _INFILE_. One option to avoid this potential problem is to pad or truncate _INFILE_ so that the original record length is maintained.

START=variable specifies a variable whose value SAS uses as the first column number of the record that the PUT _INFILE_ statement writes. Like automatic variables, the START variable is not written to the data set.

See _INFILE_ option on page 337 in the PUT statement

STOPOVER causes the DATA step to stop processing if an INPUT statement reaches the end of the current record without finding values for all variables in the statement. When an input line does not contain the expected number of values, SAS sets _ERROR_ to 1, stops building the data set as if a STOP statement has executed, and prints the incomplete data line.

Tip Use FLOWOVER to reset the default behavior.
TRUNCOVER

overrides the default behavior of the INPUT statement when an input data record is shorter than the INPUT statement expects. By default, the INPUT statement automatically reads the next input data record. TRUNCOVER enables you to read variable-length records when some records are shorter than the INPUT statement expects. Variables without any values assigned are set to missing.

Tip

Use TRUNCOVER to assign the contents of the input buffer to a variable when the field is shorter than expected.

UNBUFFERED

tells SAS not to perform a buffered ("look ahead") read.

Alias

UNBUF

Interaction

When you use UNBUFFERED, SAS never sets the END= variable to 1.

Tip

When you read instream data with a DATALINES statement, UNBUFFERED is in effect.

_INFILE_=variable

specifies a character variable that references the contents of the current input buffer for this INFILE statement. You can use the variable in the same way as any other variable, even as the target of an assignment. The variable is automatically retained and initialized to blanks. Like automatic variables, the _INFILE_= variable is not written to the data set.

Restriction

variable cannot be a previously defined variable. Ensure that the _INFILE_= specification is the first occurrence of this variable in the DATA step. Do not set or change the length of _INFILE_= variable with the LENGTH or ATTRIB statements. However, you can attach a format to this variable with the ATTRIB or FORMAT statement.

Interaction

The maximum length of this character variable is the logical record length (LRECL= on page 207) for the specified INFILE statement. However, SAS does not open the file to know the LRECL= until before the execution phase. Therefore, the designated size for this variable during the compilation phase is 32,767 bytes.

Tips

Modification of this variable directly modifies the INFILE statement's current input buffer. Any PUT _INFILE_ (when this INFILE is current) that follows the buffer modification reflects the modified buffer.
contents. The _INFILE_ = variable accesses only the current input
buffer of the specified INFILE statement even if you use the N= option
to specify multiple buffers.

To access the contents of the input buffer in another statement without
using the _INFILE_ = option, use the automatic variable _INFILE_.

The _INFILE_ variable does not have a fixed width. When you assign
a value to the _INFILE_ variable, the length of the variable changes to
the length of the value that is assigned.

See
“Accessing the Contents of the Input Buffer” on page 213

Examples
“Example 9: Working with Data in the Input Buffer” on page 225
“Example 10: Accessing the Input Buffers of Multiple Files” on page 226

Operating Environment Options

options | host-options
Operating Environment Information
For descriptions of operating environment-specific options in the INFILE
statement, see the SAS documentation for your operating environment.

See
INFILE Statement under Windows, UNIX, and z/OS

DBMS Specifications

DBMS-Specifications
enable you to read records from some DBMS files. You must license SAS/ACCESS
software to be able to read from DBMS files. See the SAS/ACCESS documentation
for the DBMS that you use.

Details

How to Use the INFILE Statement
Because the INFILE statement identifies the file to read, it must execute before the
INPUT statement that reads the input data records. You can use the INFILE statement in
conditional processing, such as an IF-THEN statement, because it is executable. The
INFILE statement enables you to control the source of the input data records.

Usually, you use an INFILE statement to read data from an external file. When data is
read from the job stream, you must use a DATALINES statement. However, to take
advantage of certain data-reading options that are available only in the INFILE
statement, you can use an INFILE statement with the file-specification DATALINES and
a DATALINES statement in the same DATA step. See “Reading Long Instream Data
Records” on page 217 for more information.

When you use more than one INFILE statement for the same file specification and you
use options in each INFILE statement, the effect is additive. To avoid confusion, use all
the options in the first INFILE statement for a given external file.
**Reading Multiple Input Files**

You can read from multiple input files in a single iteration of the DATA step in one of two ways:

- to keep multiple files open and change which file is read, use multiple INFILE statements.
- to dynamically change the current input file within a single DATA step, use the FILEVAR= option in an INFILE statement. The FILEVAR= option enables you to read from one file, close it, and then open another. See “Example 5: Reading from Multiple Input Files” on page 223.

**Updating External Files in Place**

You can use the INFILE statement in combination with the FILE statement to update records in an external file. Follow these steps:

1. Specify the INFILE statement before the FILE statement.
2. Specify the same fileref or physical filename in each statement.
3. Use options that are common to both the INFILE and FILE statements in the INFILE statement instead of the FILE statement. (Any such options that are used in the FILE statement are ignored.)


To update individual fields within a record instead of the entire record, see the SHAREBUFFERS option on page 210.

**Accessing the Contents of the Input Buffer**

In addition to the _INFILE_ variable, you can use the automatic _INFILE_ variable to reference the contents of the current input buffer for the most recent execution of the INFILE statement. This character variable is automatically retained and initialized to blanks. Like other automatic variables, _INFILE_ is not written to the data set.

When you specify the _INFILE_= option in an INFILE statement, this variable is also indirectly referenced by the automatic _INFILE_ variable. If the automatic _INFILE_ variable is present and you omit _INFILE_ in a particular INFILE statement, then SAS creates an internal _INFILE_ variable for that INFILE statement. Otherwise, SAS does not create the _INFILE_ variable for a particular FILE.

During execution and at the point of reference, the maximum length of this character variable is the maximum length of the current _INFILE_ variable. However, because _INFILE_ only references other variables whose lengths are not known until before the execution phase, the designated length is 32,767 bytes during the compilation phase. For example, if you assign _INFILE_ to a new variable whose length is undefined, then the default length of the new variable is 32,767 bytes. You cannot use the LENGTH statement and the ATTRIB statement to set or override the length of _INFILE_. You can use the FORMAT statement and the ATTRIB statement to assign a format to _INFILE_.

Like other SAS variables, you can update the _INFILE_ variable in an assignment statement. You can also use a format with _INFILE_ in a PUT statement. For example, the following PUT statement writes the contents of the input buffer by using a hexadecimal format.

```
put _infile_ $hex100.;
```

Any modification of the _INFILE_ directly modifies the current input buffer for the current INFILE statement. The execution of any PUT _INFILE_ statement that follows this buffer modification reflects the contents of the modified buffer.
_INFILE_ only accesses the contents of the current input buffer for an INFILE statement, even when you use the N= option to specify multiple buffers. You can access all the N= buffers, but you must use an INPUT statement with the # line pointer control to make the desired buffer the current input buffer.

**Reading Delimited Data**

By default, the delimiter that is used to read input data records with list input is a blank space. The delimiter-sensitive data (DSD) option, the DELIMITER= option, the DLMSTR= option, and the DLMSOPT= option affect how list input handles delimiters. The DELIMITER= or DLMSTR= option specifies that the INPUT statement use a character other than a blank as a delimiter for data values that are read with list input. When the DSD option is in effect, the INPUT statement uses a comma as the default delimiter.

To read a value as missing between two consecutive delimiters, use the DSD option. By default, the INPUT statement treats consecutive delimiters as a unit. When you use DSD, the INPUT statement treats consecutive delimiters separately. Therefore, a value that is missing between consecutive delimiters is read as a missing value. To change the delimiter from a comma to another value, use the DELIMITER= or DLMSTR= option.

For example, this DATA step program uses list input to read data that is separated with commas. The second data line contains a missing value. Because SAS allows consecutive delimiters with list input, the INPUT statement cannot detect the missing value.

```sas
data scores;
  infile datalines delimiter=',';
  input test1 test2 test3;
  datalines;
  91,87,95
  97,,92
  ,,1,1
;
```

With the FLOWOVER option in effect, the data set SCORES contains two, not three, observations. The second observation is built incorrectly:

<table>
<thead>
<tr>
<th>OBS</th>
<th>TEST1</th>
<th>TEST2</th>
<th>TEST3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>91</td>
<td>87</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>97</td>
<td>92</td>
<td>1</td>
</tr>
</tbody>
</table>

To correct the problem, use the DSD option in the INFILE statement.

```sas
data scores;
  infile datalines delimiter=',';
  input test1 test2 test3;
  datalines;
  91,87,95
  97,,92
  ,,1,1
;
  infile datalines dsd;
```

Now the INPUT statement detects the two consecutive delimiters and therefore assigns a missing value to variable TEST2 in the second observation.
The DSD option also enables list input to read a character value that contains a delimiter within a quoted string. For example, if data is separated with commas, DSD enables you to place the character string in quotation marks and read a comma as a valid character. SAS does not store the quotation marks as part of the character value. To retain the quotation marks as part of the value, use the tilde (~) format modifier in an INPUT statement. See “Example 1: Changing How Delimiters Are Treated” on page 219.

Note: Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option on 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 INFILE statement. However, if you use the DSD and DLM options in the INFILE statement, the ENCODING= option is a requirement because these options require 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, files that contain both character and noncharacter fields.

If your data is longer than the default length, you need to use informats. For example, date or time values, names, and addresses can be longer than eight characters. In such cases, you either need to add an INFORMAT statement to the DATA step or add informats directly in the INPUT statement. If you add informats in an INPUT statement, you must add a colon (:) in front of the informat to indicate that SAS is to read from the first character after the current delimiter to the number of characters specified in the informat or to the next delimiter.

```
data a;
  infile 'c:\sas\example.csv' dlm='09'x dsd truncover;
  input fname :$20. lname :$30. address1 :$50. address2 :$50. city :$40. state :$2. zip phone :$12.;
run;
```

Here are some other INFILE options that you might consider using when you read delimited data:

**FIRSTOBS=**
Use this option to skip a header record by specifying FIRSTOBS=2.

**TERMSTR=**
Use this option to specify the end-of-line character. This option is useful when you want to share data files that are created on one operating system with another operating system. For example, if you are working in a UNIX environment and you need to read a file that was created under Windows, use TERMSTR=CRLF. Similarly, if you are in a Windows environment and you need to read a file that was created under UNIX, use TERMSTR=LF.

**TRUNCOVER=**
Use this option to specify that SAS only use the available input data in the current record to populate as many variables as possible. By default, if the INPUT statement reads past the end of a record without finding enough data to populate the variables listed, it continues to read data from the next record. Use the TRUNCOVER= option to keep SAS from reading the next record.
Here are some troubleshooting problems and solutions

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The SAS log contains the following message and all of the data in the</td>
<td>This is most likely not a problem.</td>
</tr>
<tr>
<td>data set seems to be read: One or more lines were truncated.</td>
<td>This message occurs when one of the following conditions are true:</td>
</tr>
<tr>
<td></td>
<td>1. The maximum record length is less than the active logical record</td>
</tr>
<tr>
<td></td>
<td>length.</td>
</tr>
<tr>
<td></td>
<td>2. All of the data is in the data set.</td>
</tr>
<tr>
<td></td>
<td>3. You are using a value for the FIRSTOBS= that is higher than 1.</td>
</tr>
<tr>
<td></td>
<td>The message indicates that the FIRSTOBS= option has skipped a line</td>
</tr>
<tr>
<td></td>
<td>because the line is longer than the LRECL= value. SAS recognizes that</td>
</tr>
<tr>
<td></td>
<td>a line is longer than the LRECL= value, but it does not recognize that</td>
</tr>
<tr>
<td></td>
<td>the line has been skipped by the FIRSTOBS= option.</td>
</tr>
<tr>
<td>Your last variable is numeric and it is missing in every observation.</td>
<td>Use the TERMSTR=CRLF option in your INFILE statement.</td>
</tr>
<tr>
<td></td>
<td>For releases prior to SAS 9, either convert the file to the proper UNIX</td>
</tr>
<tr>
<td></td>
<td>structure or add the carriage return (\textless CR\textgreater ) to</td>
</tr>
<tr>
<td></td>
<td>the list of delimiters.</td>
</tr>
<tr>
<td></td>
<td>If you are reading a CSV file, specify the carriage return as DL DLM='2C0D'x.</td>
</tr>
<tr>
<td></td>
<td>If you are reading a tab-delimited file, use DLM='090D'x.</td>
</tr>
<tr>
<td></td>
<td>If you have a different delimiter, contact SAS Technical Support for</td>
</tr>
<tr>
<td></td>
<td>help.</td>
</tr>
</tbody>
</table>
When you read a CSV file, the records are split inside a quoted string. For example, sometime a long comment field abruptly stops and continues on the next row of the file that you are reading. You can see it if you open the file in a text editor.

This problem happens most commonly in files that are created from Microsoft Excel worksheets. In Excel, the column contains soft returns to help in formatting. When you save the worksheet as a CSV file, the soft return incorrectly seems to SAS as end-of-record markers. This, in turn, causes the records to be split incorrectly.

In SAS 9 or later, under Windows, you can use the TERMSTR=CRLF option in the INFILE statement. This setting indicates that SAS accepts only the complete return and line feed as an end-of-record marker.

In releases prior to SAS 9, run the following program. It converts any line-feed character between double quotation marks into a space to enable SAS to read the file correctly. In this program, the INFILE and FILE statement refer to the same file. The SHAREBUFFERS option enables the external file to be updated in place, removing the offending characters.

```sas
data _null_;
  infile 'c:\_today\mike.csf' recfm=n sharebuffers;
  file 'c:\_today\mike.csv' recfm=n;
  input a $char1.;
  retain open 0;
  /* toggle the open flag */
  if a='''' then open=not open;
  if a='0A'x and open then put ' ';
run;
```

The program reads the file one byte at a time and replaces the line-feed character, as needed.

**Reading Long Instream Data Records**

You can use the INFILE statement with the DATALINES file specification to process instream data. An INPUT statement reads the data records that follow the DATALINES statement. If you use the CARDIMAGE system option, or if this option is the default for your system, then SAS processes the data lines exactly like 80-byte punched card images that are padded with blanks. The default FLOWOVER option in the INFILE statement causes the INPUT statement to read the next record if it does not find values in the current record for all of the variables in the statement. To ensure that your data is processed correctly, use an external file for input when record lengths are greater than 80 bytes.

**Note:** The NOCARDIMAGE system option (see the “CARDIMAGE System Option” in *SAS System Options: Reference*) specifies that data lines not be treated as if they were 80-byte card images. The end of a data line is always treated as the end of the last token, except for strings that are enclosed in quotation marks.

**Reading Past the End of a Line**

By default, if the INPUT statement tries to read past the end of the current input data record, then it moves the input pointer to column 1 of the next record to read the remaining values. This default behavior is specified by the FLOWOVER option. A message is written to the SAS log:

```sas
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

Several options are available to change the INPUT statement behavior when an end of line is reached. The STOPOVER option treats this condition as an error and stops building the data set. The MISSOVER and TRUNCOVER options do not allow the input
The SCANOVER option, used with '@character-string' scans the input record until it finds the specified character-string. The FLOWOVER option restores the default behavior.

The TRUNCOVER and MISSOVER options are similar. The MISSOVER option causes the INPUT statement to set a value to missing if the statement is unable to read an entire field because the value is shorter than the field length that is specified in the INPUT statement. The TRUNCOVER option writes whatever characters are read to the appropriate variable.

For example, an external file with variable-length records contains these records:

```
---+---+----+----2
1
22
333
4444
55555
```

The following DATA step reads this data to create a SAS data set. Only one of the input records is as long as the informatted length of the variable TESTNUM.

```
data numbers;
   infile 'external-file';
   input testnum 5.;
run;
```

This DATA step creates the three observations from the five input records because by default the FLOWOVER option is used to read the input records.

If you use the MISSOVER option in the INFILE statement, then the DATA step creates five observations. All the values that were read from records that were too short are set to missing. Use the TRUNCOVER option in the INFILE statement if you prefer to see what values were present in records that were too short to satisfy the current INPUT statement.

```
infile 'external-file' truncover;
```

The DATA step now reads the same input records and creates five observations. See the following table to compare the SAS data sets.

**Table 2.3 The Value of TESTNUM Using Different INFILE Statement Options**

<table>
<thead>
<tr>
<th>OBS</th>
<th>FLOWOVER</th>
<th>MISSOVER</th>
<th>TRUNCOVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4444</td>
<td>.</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>55555</td>
<td>.</td>
<td>333</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>4444</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55555</td>
<td>55555</td>
<td></td>
</tr>
</tbody>
</table>
Comparisons

- The INFILE statement specifies the input file for any INPUT statements in the DATA step. The FILE statement specifies the output file for any PUT statements in the DATA step.

- An INFILE statement usually identifies data from an external file. A DATALINES statement indicates that data follows in the job stream. You can use the INFILE statement with the file specification DATALINES to take advantage of certain data-reading options that affect how the INPUT statement reads instream data.

Examples

**Example 1: Changing How Delimiters Are Treated**

By default, the INPUT statement uses a blank as the delimiter. This DATA step uses a comma as the delimiter:

```plaintext
data num;
  infile datalines dsd;
  input x y z;
  datalines;
  ,2,3
  4,5,6
  7,8,9

The argument DATALINES in the INFILE statement enables you to use an INFILE statement option to read instream data lines. The DSD option sets the comma as the default delimiter. Because a comma precedes the first value in the first data line, a missing value is assigned to variable X in the first observation, and the value 2 is assigned to variable Y.

If the data uses multiple delimiters or a single delimiter other than a comma, then simply specify the delimiter values with the DELIMITER= option. In this example, the characters a and b function as delimiters:

```plaintext
data nums;
  infile datalines dsd delimiter='ab';
  input X Y Z;
  datalines;
  1aa2ab3
  4b5bab6
  7a8b9

The output that PROC PRINT generates shows the resulting NUM data set. Values are missing for variables in the first and second observations because DSD causes list input to detect two consecutive delimiters. If you omit DSD, the characters a, b, aa, ab, ba, or bb function as the delimiter and no variables are assigned missing values.
If you want to use a string as the delimiter, specify the delimiter values with the DLMSTR= option. In this example, the string PRD is used as the delimiter. Note that the string contains uppercase characters. By using the DLMSOPT= option, PRD, Prd, PRd, PrD, pRd, pRD, prD, and prd are all valid delimiters.

```sas
data test;
  infile datalines dsd dlmstr='PRD' dlmsopt='i';
  input X Y Z;
  datalines;
  1PRD2PRd3
  4PrD5Prd6
  7pRd8pRD9
;  
  proc print data=test; run;
```

The output from PROC PRINT shows all the observations in the TEST data set.

**Output 2.7  The TEST Data Set**

This DATA step uses modified list input and the DSD option to read data that is separated by commas and that might contain commas as part of a character value:

```sas
data scores;
  infile datalines dsd;
  input Name : $9. Score
    Team : $25. Div $;
  datalines;
  Mitchel,82,"Blue Bunnies, Richmond",AAA
  Sue Ellen,74,"Green Gazelles, Atlanta",AA
;  
  proc print; run;
```
The output that PROC PRINT generates shows the resulting SCORES data set. The delimiter (comma) is stored as part of the value of TEAM while the quotation marks are not.

**Output 2.8 The SCORES Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Score</th>
<th>Team</th>
<th>Div</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joseph</td>
<td>76</td>
<td>Red Racers, Washington</td>
<td>AAA</td>
</tr>
<tr>
<td>2</td>
<td>Mitchel</td>
<td>82</td>
<td>Blue Bunnies, Richmond</td>
<td>AAA</td>
</tr>
<tr>
<td>3</td>
<td>Sue Ellen</td>
<td>74</td>
<td>Green Gazelles, Atlanta</td>
<td>AA</td>
</tr>
</tbody>
</table>

**Example 2: Handling Missing Values and Short Records with List Input**

This example demonstrates how to prevent missing values from causing problems when you read the data with list input. Some data lines in this example contain fewer than five temperature values. Use the MISSOVER option so that these values are set to missing.

data weather;
   infile datalines missover;
   input temp1-temp5;
   datalines;
   97.9 98.1 98.3
   98.6 99.2 99.1 98.5 97.5
   96.2 97.3 98.3 97.6 96.5
;

SAS reads the three values on the first data line as the values of TEMP1, TEMP2, and TEMP3. The MISSOVER option causes SAS to set the values of TEMP4 and TEMP5 to missing for the first observation because no values for those variables are in the current input data record.

When you omit the MISSOVER option or use FLOWOVER, SAS moves the input pointer to line 2 and reads values for TEMP4 and TEMP5. The next time the DATA step executes, SAS reads a new line which, in this case, is line 3. This message appears in the SAS log:

```
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

You can also use the STOPOVER option in the INFILE statement. Using the STOPOVER option causes the DATA step to halt execution when an INPUT statement does not find enough values in a record of raw data:

```
infile datalines stopover;
```

Because SAS does not find a TEMP4 value in the first data record, it sets _ERROR_ to 1, stops building the data set, and prints data line 1.

**Example 3: Scanning Variable-Length Records for a Specific Character String**

This example shows how to use TRUNCOVER in combination with SCANOVER to pull phone numbers from a phone book. The phone number is always preceded by the...
word “phone:”. Because the phone numbers include international numbers, the maximum length is 32 characters.

```sas
filename phonebk host-specific-path;
data _null_;
  file phonebk;
  input line $80.;
  put line;
datalines;
  Jenny's Phone Book
  Jim Johanson phone: 619-555-9340
    Jim wants a scarf for the holidays.
  Jane Jovalley phone: (213) 555-4820
    Jane started growing cabbage in her garden.
    Her dog's name is Juniper.
  J.R. Hauptman phone: (49)12 34-56 78-90
    J.R. is my brother.
;
run;
```

Use @’phone:’ to scan the lines of the file for a phone number and position the file pointer where the phone number begins. Use TRUNCOVER in combination with SCANOVER to skip the lines that do not contain ‘phone:’ and write only the phone numbers to the log.

```sas
data _null_;
infile phonebk truncover scanover;
  input @'phone:' phone $32.;
  put phone=;
run;
```

The program writes the following lines to the SAS log:

```
phone=619-555-9340
phone=(213) 555-4820
phone=(49)12 34-56 78-90
```

### Example 4: Reading Files That Contain Variable-Length Records

This example shows how to use LENGTH=, in combination with the $VARYING. informat, to read a file that contains variable-length records:

```sas
data a;
infile file-specification length=linelen lrecl=510 pad;
  input firstvar 1-10 @; /* assign LINELEN */
  varlen=linelen-10; /* Calculate VARLEN */
  input @11 secondvar $varying500. varlen;
run;
```

The following occurs in this DATA step:

- The INFILE statement creates the variable LINELEN but does not assign it a value.
- When the first INPUT statement executes, SAS determines the line length of the record and assigns that value to the variable LINELEN. The single trailing @ holds the record in the input buffer for the next INPUT statement.
- The assignment statement uses the two known lengths (the length of FIRSTVAR and the length of the entire record) to determine the length of VARLEN.
• The second INPUT statement uses the VARLEN value with the informat $VARYING500. to read the variable SECONDVAR.

See the “$VARYINGw. Informat” in SAS Formats and Informats: Reference for more information.

Example 5: Reading from Multiple Input Files

The following DATA step reads from two input files during each iteration of the DATA step. As SAS switches from one file to the next, each file remains open. The input pointer remains in place to begin reading from that location the next time an INPUT statement reads from that file.

data qtrtot(drop=jansale febsale marsale aprsale maysale junsale);
  /* identify location of 1st file */
  infile file-specification-1;
  /* read values from 1st file     */
  input name $ jansale febsale marsale;
  qtr1tot=sum(jansale,febsale,marsale);
  /* identify location of 2nd file */
  infile file-specification-2;
  /* read values from 2nd file     */
  input @7 aprsale maysale junsale;
  qtr2tot=sum(aprsale,maysale,junsale);
run;

The DATA step terminates when SAS reaches an end of file on the shortest input file.

This DATA step uses FILEVAR= to read from a different file during each iteration of the DATA step:

data allsales;
  length fileloc myinfile $ 300;
  input fileloc $ ; /* read instream data       */
  /* The INFILE statement closes the current file */
  /* and opens a new one if FILELOC changes value */
  /* when INFILE executes */
  infile file-specification filevar=fileloc
    filename=myinfile end=done;
  /* DONE set to 1 when last input record read */
  do while(not done);
  /* Read all input records from the currently */
  /* opened input file, write to ALLSALES */
    input name $ jansale febsale marsale;
    output;
  end;
  put 'Finished reading ' myinfile=;
datalines;
external-file-1
external-file-2
external-file-3
;

The FILENAME= option assigns the name of the current input file to the variable MYINFILE. The LENGTH statement ensures that the FILENAME= variable and FILEVAR= variable have a length that is long enough to contain the value of the filename. The PUT statement prints the physical name of the currently open input file to the SAS log.
Example 6: Updating an External File
This example shows how to use the INFILE statement with the SHAREBUFFERS option and the INPUT, FILE, and PUT statements to update an external file in place:

data _null_;  
   /* The INFILE and FILE statements must specify the same file. */
   infile file-specification-1 sharebuffers;
   file file-specification-1;
   input state $ 1-2 phone $ 5-16;
   /* Replace area code for NC exchanges */
   if state= 'NC' and substr(phone,5,3)='333' then 
      phone='910-'||substr(phone,5,8);
   put phone 5-16;
run;

Example 7: Truncating Copied Records
The LENGTH= option is useful when you copy the input file to another file with the PUT _INFILE_ statement. Use LENGTH= to truncate the copied records. For example, these statements truncate the last 20 columns from each input data record before the input data record is written to the output file:

data _null_;  
   infile file-specification-1 length=a;
   input;
   a=a-20;
   file file-specification-2;
   put _infile_; 
run;

The START= option is also useful when you want to truncate what the PUT _INFILE_ statement copies. For example, if you do not want to copy the first 10 columns of each record, these statements copy from column 11 to the end of each record in the input buffer:

data _null_;  
   infile file-specification start=s;
   input;
   s=11;
   file file-specification-2;
   put _infile_; 
run;

Example 8: Listing the Pointer Location
This DATA step assigns the value of the current pointer location in the input buffer to the variables LINEPT and COLUMNPT:

data _null_;  
   infile datalines n=2 line=Linept col=Columnpt;
   input name $ 1-15 #2 @3 id;
   put linept= columnpt=;
   datalines;
   J. Brooks
   40974
   T. R. Ansen
   4032
;
These statements produce the following line for each execution of the DATA step because the input pointer is on the second line in the input buffer when the PUT statement executes:

Linept=2 Columnpt=9
Linept=2 Columnpt=8

**Example 9: Working with Data in the Input Buffer**

The _INFILE_ variable always contains the most recent record that is read from an INPUT statement. This example illustrates the use of the _INFILE_ variable to perform these tasks:

- read an entire record that you want to parse without using the INPUT statement.
- read an entire record that you want to write to the SAS log.
- modify the contents of the input record before parsing the line with an INPUT statement.

The example file contains phone bill information. The numeric data, minutes, and charge are enclosed in angle brackets (< >).

```sas
filename phonbill host-specific-filename;
data _null_; file phonbill;
input line $80.; put line;
datalines;
City Number Minutes Charge
Jackson 415-555-2384 <25> <2.45>
Jefferson 813-555-2356 <15> <1.62>
Joliet 913-555-3223 <65> <10.32>
.; run;
```

The following code reads each record and parses the record to extract the minute and charge values.

```sas
data _null_;
infile phonbill firstobs=2;
input;
city = scan(_infile_, 1, ' ');
char_min = scan(_infile_, 3, ' ');
char_min = substr(char_min, 2, length(char_min)-2);
minutes = input(char_min, BEST12.);
put city= minutes=;
run;
```

The program writes the following lines to the SAS log:

```
city=Jackson minutes=25
city=Jefferson minutes=15
city=Joliet minutes=65
```

The INPUT statement in the following code reads a record from the file. The automatic _INFILE_ variable is used in the PUT statement to write the record to the log.

```sas
data _null_;
infile phonbill;
input;
```
The program writes the following lines to the SAS log:

<table>
<thead>
<tr>
<th>City</th>
<th>Number</th>
<th>Minutes</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackson</td>
<td>415-555-2384</td>
<td>25</td>
<td>2.45</td>
</tr>
<tr>
<td>Jefferson</td>
<td>813-555-2356</td>
<td>15</td>
<td>1.62</td>
</tr>
<tr>
<td>Joliet</td>
<td>913-555-3223</td>
<td>65</td>
<td>10.32</td>
</tr>
</tbody>
</table>

In the following code, the first INPUT statement reads and holds the record in the input buffer. The _INFILE_ = option removes the angle brackets (< >) from the numeric data. The second INPUT statement parses the value in the buffer.

```sas
data _null_;
  length city number $16. minutes charge 8;
  infile phonbill firstobs=2;
  input @;
  _infile_ = compress(_infile_, '<>');
  input city number minutes charge;
  put city= number= minutes= charge=;
run;
```

The program writes the following lines to the SAS log:

<table>
<thead>
<tr>
<th>City</th>
<th>Number</th>
<th>Minutes</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackson</td>
<td>415-555-2384</td>
<td>25</td>
<td>2.45</td>
</tr>
<tr>
<td>Jefferson</td>
<td>813-555-2356</td>
<td>15</td>
<td>1.62</td>
</tr>
<tr>
<td>Joliet</td>
<td>913-555-3223</td>
<td>65</td>
<td>10.32</td>
</tr>
</tbody>
</table>

**Example 10: Accessing the Input Buffers of Multiple Files**

This example uses both the _INFILE_ automatic variable and the _INFILE_= option to read multiple files and access the input buffers for each of them. The following code creates four files: three data files and one file that contains the names of all the data files. The second DATA step reads the filenames file, opens each data file, and writes the contents to the log. Because the PUT statement needs _INFILE_ for the filenames file and the data file, one of the _INFILE_ variables is referenced with `fname`.

```sas
data _null_;
  do i = 1 to 3;
    fname = 'external-data-file' || put(i,1.) || '.dat';
    file datfiles filevar=fname;
    do j = 1 to 5;
      put i j;
    end;
    file 'external-filenames-file';
    put fname;
  end;
run;

data _null_;
  infile 'external-filenames-file' _infile_=fname;
  input;
  infile datfiles filevar=fname end=eof;
  do while('eof');
    input;
    put fname _infile_;
  end;
run;
```
The program writes the following lines to the SAS log:

```
NOTE: The infile 'external-filenames-file' is:
 File Name=external-filenames-file,
       RECFM=V, LRECL=256
NOTE: The infile DATFILES is:
 File Name=external-data-file1.dat,
       RECFM=V, LRECL=256
 external-data-file1.dat 1 1
 external-data-file1.dat 1 2
 external-data-file1.dat 1 3
 external-data-file1.dat 1 4
 external-data-file1.dat 1 5
NOTE: The infile DATFILES is
 File Name=external-data-file2.dat,
       RECFM=V, LRECL=256
 external-data-file2.dat 2 1
 external-data-file2.dat 2 2
 external-data-file2.dat 2 3
 external-data-file2.dat 2 4
 external-data-file2.dat 2 5
NOTE: The infile DATFILES is
 File Name=external-data-file3.dat,
       RECFM=V, LRECL=256
 external-data-file3.dat 3 1
 external-data-file3.dat 3 2
 external-data-file3.dat 3 3
 external-data-file3.dat 3 4
 external-data-file3.dat 3 5
```

**Example 11: Specifying an Encoding When Reading an External File**

This example creates a SAS data set from an external file. The external file's encoding is in UTF-8, and the current SAS session encoding is WLatin1. By default, SAS assumes that the 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';
data myfiles.unicode;
   infile extfile encoding="utf-8";
   input Make $ Model $ Year;
run;
```

See Also

- “How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?” in *SAS Language Reference: Concepts*

Statements:

- “FILENAME Statement” on page 94
- “FILENAME Statement, ACTIVEMQ Access Method” in *Application Messaging with SAS*
- “FILENAME Statement, JMS Access Method” in *Application Messaging with SAS*
INFORMAT Statement

Associates informats with variables.

Valid in: DATA step or PROC step
Category: Information
Type: Declarative

Syntax

INFORMAT variable-1 <...variable-n> <informat>;
INFORMAT <variable-1> <... variable-n> <DEFAULT=default-informat>;
INFORMAT variable-1 <...variable-n> informat <DEFAULT=default-informat>;

Arguments

variable

specifies one or more variables to associate with an informat. You must specify at least one variable when specifying an informat or when including no other arguments. Specifying a variable is optional when using a DEFAULT= informat specification.

Tip To disassociate an informat from a variable, use the variable's name in an INFORMAT statement without specifying an informat. Place the INFORMAT statement after the SET statement. See “Example 3: Removing an Informat” on page 231.

informat

specifies the informat for reading the values of the variables that are listed in the INFORMAT statement.

Tip If an informat is associated with a variable by using the INFORMAT statement, and that same informat is not associated with that same variable in the INPUT statement, then that informat will behave like informats that you specify with a colon (:) modifier in an INPUT statement. SAS reads the variables by using list input with an informat. For example, you can use the : modifier with an informat to read character values that are longer than eight bytes, or numeric values that contain nonstandard values. For details, see “INPUT Statement, List” on page 255.
See "SAS Formats and Informats: Reference"

Example “Example 2: Specifying Numeric and Character Informats” on page 230

**DEFAULT= default-informat**

specifies a temporary default informat for reading values of the variables that are listed in the INFORMAT statement. If no variable is specified, then the DEFAULT= informat specification applies a temporary default informat for reading values of all the variables of that type included in the DATA step. Numeric informats are applied to numeric variables, and character informats are applied to character variables. These default informats apply only to the current DATA step.

A DEFAULT= informat specification applies to

• variables that are not named in an INFORMAT or ATTRIB statement
• variables that are not permanently associated with an informat within a SAS data set
• variables that are not read with an explicit informat in the current DATA step.

Default

If you omit DEFAULT=, SAS uses \textit{w.d} as the default numeric informat and \textit{$w$} as the default character informat.

Restriction

Use this argument only in a DATA step.

Tip

A DEFAULT= specification can occur anywhere in an INFORMAT statement. It can specify either a numeric default, a character default, or both.

Example “Example 1: Specifying Default Informats” on page 230

---

**Details**

**The Basics**

An INFORMAT statement in a DATA step permanently associates an informat with a variable. You can specify standard SAS informats or user-written informats, previously defined in PROC FORMAT. A single INFORMAT statement can associate the same informat with several variables, or it can associate different informats with different variables. If a variable appears in multiple INFORMAT statements, SAS uses the informat that is assigned last.

**CAUTION:**

Because an INFORMAT statement defines the length of previously undefined character variables, you can truncate the values of character variables in a DATA step if an INFORMAT statement precedes a SET statement.

**How SAS Treats Variables When You Assign Informats with the INFORMAT Statement**

Informats that are associated with variables by using the INFORMAT statement behave like informats that are used with modified list input. SAS reads the variables by using the scanning feature of list input, but applies the informat. In modified list input, SAS

• does not use the value of \textit{w} in an informat to specify column positions or input field widths in an external file
• uses the value of \textit{w} in an informat to specify the length of previously undefined character variables
ignores the value of \( w \) in numeric informats

• uses the value of \( d \) in an informat in the same way it usually does for numeric informats

• treats blanks that are embedded as input data as delimiters unless you change their status with a DLM= or DLMSTR= option specification in an INFILE statement.

If you have coded the INPUT statement to use another style of input, such as formatted input or column input, that style of input is not used when you use the INFORMAT statement.

Comparisons

• Both the ATTRIB and INFORMAT statements can associate informats with variables, and both statements can change the informat that is associated with a variable. You can also use the INFORMAT statement in PROC DATASETS to change or remove the informat that is associated with a variable. The SAS windowing environment enables you to associate, change, or disassociate informats and variables in existing SAS data sets.

• SAS changes the descriptor information of the SAS data set that contains the variable. You can use an INFORMAT statement in some PROC steps, but the rules are different. For more information, see “FORMAT Procedure” in *Base SAS Procedures Guide*.

Examples

**Example 1: Specifying Default Informats**
This example uses an INFORMAT statement to associate a default numeric informat:

```sas
data tstinfmt;
  informat default=3.1;
  input x;
  put x;
  datalines;
111
222
333
;
```

The PUT statement produces these results:

11.1
22.2
33.3

**Example 2: Specifying Numeric and Character Informats**
This example associates a character informat and a numeric informat with SAS variables. Although the character variables do not fully occupy 15 column positions, the INPUT statement reads the data records correctly by using modified list input:

```sas
data name;
  informat FirstName LastName $15. n1 6.2 n2 7.3;
  input firstname lastname n1 n2;
  datalines;
Alexander Robinson 35 11
;
```
proc contents data=name;
run;
proc print data=name;
run;

The following output shows a partial listing from PROC CONTENTS, as well as the report PROC PRINT generates.

Output 2.9  Associating Numeric and Character Informats with SAS Variables

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FirstName</td>
<td>Char</td>
<td>15</td>
<td>$15.</td>
</tr>
<tr>
<td>2</td>
<td>LastName</td>
<td>Char</td>
<td>15</td>
<td>$15.</td>
</tr>
<tr>
<td>3</td>
<td>n1</td>
<td>Num</td>
<td>8</td>
<td>6.2</td>
</tr>
<tr>
<td>4</td>
<td>n2</td>
<td>Num</td>
<td>8</td>
<td>7.3</td>
</tr>
</tbody>
</table>

Output 2.10  PROC PRINT Report

<table>
<thead>
<tr>
<th></th>
<th>FirstName</th>
<th>LastName</th>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alexander</td>
<td>Robinson</td>
<td>0.35</td>
<td>0.011</td>
</tr>
</tbody>
</table>

Example 3: Removing an Informat
This example disassociates an existing informat. The order of the INFORMAT and SET statements is important.

data rtest;
  set rtest;
  informat x;
run;

See Also

Statements:
- “ATTRIB Statement” on page 27
- “INPUT Statement” on page 232
- “INPUT Statement, List” on page 255
INPUT Statement

Describes the arrangement of values in the input data record and assigns input values to the corresponding SAS variables.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

INPUT <specification(s)> <@ | @@>;

Without Arguments

The INPUT statement with no arguments is called a null INPUT statement. The null INPUT statement
• brings an input data record into the input buffer without creating any SAS variables
• releases an input data record that is held by a trailing @ or a double trailing @.

For an example, see “Example 2: Using a Null INPUT Statement” on page 244.

Arguments

specification(s) can include

variable
names a variable that is assigned input values.

(variable-list)
specifies a list of variables that are assigned input values.

Requirement The (variable-list) is followed by an (informat-list).
See “How to Group Variables and Informats” on page 253

$ specifies to store the variable value as a character value rather than as a numeric value.

Tip If the variable is previously defined as character, $ is not required.

Example “Example 1: Using Multiple Styles of Input in One INPUT Statement” on page 244

pointer-control
moves the input pointer to a specified line or column in the input buffer.

See “Column Pointer Controls” on page 234 and “Line Pointer Controls” on page 236

column-specifications specifies the columns of the input record that contain the value to read.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tip</strong></td>
<td>Informats are ignored. Only standard character and numeric data can be read correctly with this method.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Column Input” on page 237</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 1: Using Multiple Styles of Input in One INPUT Statement” on page 244</td>
</tr>
<tr>
<td><strong>format-modifier</strong></td>
<td>allows modified list input or controls the amount of information that is reported in the SAS log when an error in an input value occurs.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>Use modified list input to read data that cannot be read with simple list input.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“When to Use List Input” on page 257 and “Format Modifiers for Error Reporting” on page 236</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 6: Positioning the Pointer with a Character Variable” on page 246</td>
</tr>
<tr>
<td><strong>informat</strong></td>
<td>specifies an informat to use to read the variable value.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>You can use modified list input to read data with informats. Modified list input is useful when the data require informats but cannot be read with formatted input because the values are not aligned in columns.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Formatted Input” on page 238 and “List Input” on page 238</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 2: Using Informat Lists” on page 254</td>
</tr>
<tr>
<td><strong>(informat-list)</strong></td>
<td>specifies a list of informats to use to read the values for the preceding list of variables.</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>The (informat-list) must follow the (variable-list).</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“How to Group Variables and Informats” on page 253</td>
</tr>
<tr>
<td><strong>@</strong></td>
<td>holds an input record for the execution of the next INPUT statement within the same iteration of the DATA step. This line-hold specifier is called trailing @.</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>The trailing @ must be the last item in the INPUT statement.</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>The trailing @ prevents the next INPUT statement from automatically releasing the current input record and reading the next record into the input buffer. It is useful when you need to read from a record multiple times.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Using Line-Hold Specifiers” on page 240</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 3: Holding a Record in the Input Buffer” on page 244</td>
</tr>
<tr>
<td><strong>@@</strong></td>
<td>holds the input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called double trailing @.</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>The double trailing @ must be the last item in the INPUT statement.</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>The double trailing @ is useful when each input line contains values for several observations, or when a record needs to be reread on the next iteration of the DATA step.</td>
</tr>
<tr>
<td><strong>See</strong></td>
<td>“Using Line-Hold Specifiers” on page 240</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>“Example 4: Holding a Record across Iterations of the DATA Step” on page 245</td>
</tr>
</tbody>
</table>

**Column Pointer Controls**

@\(n\)

moves the pointer to column \(n\).

**Range**  
a positive integer

**Tip**  
If \(n\) is not an integer, SAS truncates the decimal value and uses only the integer value. If \(n\) is zero or negative, the pointer moves to column 1.

**Example**  
@15 moves the pointer to column 15:  
```plaintext```
```plaintext```
```
```
```

Example  “Example 7: Moving the Pointer Backward” on page 247

@\(\text{numeric-variable}\)

moves the pointer to the column given by the value of \(\text{numeric-variable}\).

**Range**  
a positive integer

**Tip**  
If \(\text{numeric-variable}\) is not an integer, SAS truncates the decimal value and only uses the integer value. If \(\text{numeric-variable}\) is zero or negative, the pointer moves to column 1.

**Example**  
The value of the variable A moves the pointer to column 15:  
```plaintext```
```plaintext```
```
```
```

Example  “Example 5: Positioning the Pointer with a Numeric Variable” on page 246

@\((\text{expression})\)

moves the pointer to the column that is given by the value of \(\text{expression}\).

**Restriction**  
\(\text{Expression}\) must result in a positive integer.

**Tip**  
If the value of \(\text{expression}\) is not an integer, SAS truncates the decimal value and only uses the integer value. If it is zero or negative, the pointer moves to column 1.

**Example**  
The result of the expression moves the pointer to column 15:  
```plaintext```
```plaintext```
```
```
```

Example  “Example 5: Positioning the Pointer with a Numeric Variable” on page 246
@‘character-string’
locates the specified series of characters in the input record and moves the pointer to
the first column after character-string.

@character-variable
locates the series of characters in the input record that is given by the value of
character-variable and moves the pointer to the first column after that series of
characters.

Example
The following statement reads in the WEEKDAY character variable. The
second @1 moves the pointer to the beginning of the input line. The value
for SALES is read from the next non-blank column after the value of
WEEKDAY:
input @1 day 1. @5 weekday $10.
@1 @weekday sales 8.2;

Example “Example 6: Positioning the Pointer with a Character Variable” on page
246

@ (character-expression)
locates the series of characters in the input record that is given by the value of
character-expression and moves the pointer to the first column after the series.

Example “Example 6: Positioning the Pointer with a Character Variable” on page
246

+n
moves the pointer n columns.

Range a positive integer or zero

Tip If n is not an integer, SAS truncates the decimal value and uses only the
integer value. If the value is greater than the length of the input buffer, the
pointer moves to column 1 of the next record.

Example This statement moves the pointer to column 23, reads a value for
LENGTH from columns 23 through 26, advances the pointer five
columns, and reads a value for WIDTH from columns 32 through 35:
input @23 length 4. +5 width 4.;

Example “Example 7: Moving the Pointer Backward” on page 247

+numeric-variable
moves the pointer the number of columns that is given by the value of numeric-variable.

Range a positive or negative integer or zero

Tip If numeric-variable is not an integer, SAS truncates the decimal value and
uses only the integer value. If numeric-variable is negative, the pointer
moves backward. If the current column position becomes less than 1, the
pointer moves to column 1. If the value is zero, the pointer does not
move. If the value is greater than the length of the input buffer, the
pointer moves to column 1 of the next record.

Example “Example 7: Moving the Pointer Backward” on page 247
+(expression)
  moves the pointer the number of columns given by expression.

  Range   expression must result in a positive or negative integer or zero.

  Tip     If expression is not an integer, SAS truncates the decimal value and uses only the integer value. If expression is negative, the pointer moves backward. If the current column position becomes less than 1, the pointer moves to column 1. If the value is zero, the pointer does not move. If the value is greater than the length of the input buffer, the pointer moves to column 1 of the next record.

Line Pointer Controls

#n
  moves the pointer to record n.

  Range   a positive integer

  Interaction The N= option in the INFILE statement can affect the number of records the INPUT statement reads and the placement of the input pointer after each iteration of the DATA step. See the option N= on page 208.

  Example   The #2 moves the pointer to the second record to read the value for ID from columns 3 and 4:
             input name $10. #2 id 3-4;

#numeric-variable
  moves the pointer to the record that is given by the value of numeric-variable.

  Range   a positive integer

  Tip     If the value of numeric-variable is not an integer, SAS truncates the decimal value and uses only the integer value.

#(expression)
  moves the pointer to the record that is given by the value of expression.

  Range   expression must result in a positive integer.

  Tip     If the value of expression is not an integer, SAS truncates the decimal value and uses only the integer value.

/  advances the pointer to column 1 of the next input record.

  Example   The values for NAME and AGE are read from the first input record before the pointer moves to the second record to read the value of ID from columns 3 and 4:
             input name age / id 3-4;

Format Modifiers for Error Reporting

?  suppresses printing the invalid data note when SAS encounters invalid data values.
See “How Invalid Data Is Handled” on page 242

suppresses printing the messages and the input lines when SAS encounters invalid data values. The automatic variable _ERROR_ is not set to 1 for the invalid observation.

See “How Invalid Data Is Handled” on page 242

Details

When to Use INPUT
Use the INPUT statement to read raw data from an external file or in-stream data. If your data are stored in an external file, you can specify the file in an INFILE statement. The INFILE statement must execute before the INPUT statement that reads the data records. If your data are in-stream, a DATALINES statement must precede the data lines in the job stream. If your data contain semicolons, use a DATALINES4 statement before the data lines. A DATA step that reads raw data can include multiple INPUT statements.

You can also use the INFILE statement to read in-stream data by specifying a filename of DATALINES in the INFILE statement before the INPUT statement. Using DATALINES in the INFILE statement enables you to use most of the options available in the INFILE statement with in-stream data.

To read data that are already stored in a SAS data set, use a SET statement. To read database or PC file-format data that are created by other software, use the SET statement after you access the data with the LIBNAME statement. See the SAS/ACCESS documentation for more information.

z/OS Specifics
LOG files that are generated under z/OS and captured with PROC PRINTTO contain an ASA control character in column 1. If you are using the INPUT statement to read a LOG file that was generated under z/OS, you must account for this character if you use column input or column pointer controls.

Input Styles

Overview of Input Styles
There are four ways to describe a record's values in the INPUT statement:

- column
- list (simple and modified)
- formatted
- named.

Each variable value is read by using one of these input styles. An INPUT statement can contain any or all of the available input styles, depending on the arrangement of data values in the input records. However, once named input is used in an INPUT statement, you cannot use another input style.

Column Input
With column input, the column numbers follow the variable name in the INPUT statement. These numbers indicate where the variable values are found in the input data records:

input name $ 1-8 age 11-12;
This INPUT statement can read the following data records:

```
----+----1----+----2----+
Peterson  21
Morgan    17
```

Because NAME is a character variable, a $ appears between the variable name and column numbers. For more information, see “INPUT Statement, Column” on page 248.

**List Input**

With list input, the variable names are simply listed in the INPUT statement. A $ follows the name of each character variable:

```
input name $ age;
```

This INPUT statement can read data values that are separated by blanks or aligned in columns (with at least one blank between):

```
----+----1----+----2----+
Peterson  21
Morgan    17
```

For more information, see “INPUT Statement, List” on page 255.

**Formatted Input**

With formatted input, an informat follows the variable name in the INPUT statement. The informat gives the data type and the field width of an input value. Informats also enable you to read data that are stored in nonstandard form, such as packed decimal, or numbers that contain special characters such as commas.

```
input name $char8. +2 income comma6. ;
```

This INPUT statement reads these data records correctly:

```
----+----1----+----2----+
Petcerson  21,000
Morgan  17,132
```

The pointer control of +2 moves the input pointer to the field that contains the value for the variable INCOME. For more information, see “INPUT Statement, Formatted” on page 251.

**Named Input**

With named input, you specify the name of the variable followed by an equal sign. SAS looks for a variable name and an equal sign in the input record:

```
input name= $ age=;
```

This INPUT statement reads the following data records correctly:

```
----+----1----+----2----+
name=Peterson age=21
name=Morgan age=17
```

For more information, see “INPUT Statement, Named” on page 262.

**Multiple Styles in a Single INPUT Statement**

An INPUT statement can contain any or all of the different input styles:

```
input idno name $18. team $ 25-30 startwght endwght;
```

This INPUT statement reads the following data records correctly:

```
----+----1----+----2----+----3----+----4----+
```
The value of IDNO, STARTWGHT, and ENDWGHT are read with list input, the value of NAME with formatted input, and the value of TEAM with column input.

Note: Once named input is used in an INPUT statement, you cannot change input styles.

**Pointer Controls**

**Overview of Pointers**

As SAS reads values from the input data records into the input buffer, it keeps track of its position with a pointer. The INPUT statement provides three ways to control the movement of the pointer:

- **column pointer controls**
  - reset the pointer's column position when the data values in the data records are read.

- **line pointer controls**
  - reset the pointer's line position when the data values in the data records are read.

- **line-hold specifiers**
  - hold an input record in the input buffer so that another INPUT statement can process it. By default, the INPUT statement releases the previous record and reads another record.

With column and line pointer controls, you can specify an absolute line number or column number to move the pointer or you can specify a column or line location relative to the current pointer position. The following table lists the pointer controls that are available with the INPUT statement.

<table>
<thead>
<tr>
<th>Pointer Controls</th>
<th>Relative</th>
<th>Absolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>column pointer controls</td>
<td>+n</td>
<td>@n</td>
</tr>
<tr>
<td></td>
<td>numeric-variable</td>
<td>numeric-variable</td>
</tr>
<tr>
<td></td>
<td>(expression)</td>
<td>(expression)</td>
</tr>
<tr>
<td></td>
<td>'character-string'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>character-variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(character-expression)</td>
<td></td>
</tr>
<tr>
<td>line pointer controls</td>
<td>/</td>
<td>#n</td>
</tr>
<tr>
<td></td>
<td>numeric-variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(expression)</td>
<td></td>
</tr>
<tr>
<td>line-hold specifiers</td>
<td>@</td>
<td>(not applicable)</td>
</tr>
<tr>
<td></td>
<td>@@</td>
<td>(not applicable)</td>
</tr>
</tbody>
</table>
Note: Always specify pointer controls before the variable to which they apply.

You can use the COLUMN= and LINE= options in the INFILE statement to determine the pointer's current column and line location.

Using Column and Line Pointer Controls

Column pointer controls indicate the column in which an input value starts.

Use line pointer controls within the INPUT statement to move to the next input record or to define the number of input records per observation. Line pointer controls specify which input record to read. To read multiple data records into the input buffer, use the N= option in the INFILE statement to specify the number of records. If you omit N=, you need to take special precautions. For more information, see “Reading More Than One Record per Observation” on page 241.

Using Line-Hold Specifiers

Line-hold specifiers keep the pointer on the current input record when

- a data record is read by more than one INPUT statement (trailing @)
- one input line has values for more than one observation (double trailing @)
- a record needs to be reread on the next iteration of the DATA step (double trailing @).

Use a single trailing @ to allow the next INPUT statement to read from the same record. Use a double trailing @ to hold a record for the next INPUT statement across iterations of the DATA step.

Normally, each INPUT statement in a DATA step reads a new data record into the input buffer. When you use a trailing @, the following occurs:

- The pointer position does not change.
- No new record is read into the input buffer.
- The next INPUT statement for the same iteration of the DATA step continues to read the same record rather than a new one.

SAS releases a record held by a trailing @ when

- a null INPUT statement executes:
  
  input;

- an INPUT statement without a trailing @ executes
- the next iteration of the DATA step begins.

Normally, when you use a double trailing @ (@@), the INPUT statement for the next iteration of the DATA step continues to read the same record. SAS releases the record that is held by a double trailing @

- immediately if the pointer moves past the end of the input record
- immediately if a null INPUT statement executes:
  
  input;

- when the next iteration of the DATA step begins if an INPUT statement with a single trailing @ executes later in the DATA step:
  
  input @;
**Pointer Location After Reading**

Understanding the location of the input pointer after a value is read is important, especially if you combine input styles in a single INPUT statement. With column and formatted input, the pointer reads the columns that are indicated in the INPUT statement and stops in the next column. With list input, however, the pointer scans data records to locate data values and reads a blank to indicate that a value has ended. After reading a value with list input, the pointer stops in the second column after the value.

For example, you can read these data records with list, column, and formatted input:

```
----+----1----+----2----+----3
REGION1    49670
REGION2    97540
REGION3    86342
```

This INPUT statement uses list input to read the data records:

```
input region $ jansales;
```

After reading a value for REGION, the pointer stops in column 9.

```
----+----1----+----2----+----3
REGION1    49670
↑
```

These INPUT statements use column and formatted input to read the data records:

- **column input**
  ```
  input region $ 1-7 jansales 12-16;
  ```

- **formatted input**
  ```
  input region $7. +4 jansales 5.;
  input region $7. @12 jansales 5.;
  ```

To read a value for the variable REGION, the INPUT statements instruct the pointer to read seven columns and stop in column 8.

```
----+----1----+----2----+----3
REGION1    49670
↑
```

**Reading More Than One Record per Observation**

**Using the # Pointer Control**

The highest number that follows the # pointer control in the INPUT statement determines how many input data records are read into the input buffer. Use the N= option in the INFILE statement to change the number of records. For example, in this statement, the highest value after the # is 3:

```
input @31 age 3. #3 id 3-4 #2 @6 name $20.;
```

Unless you use N= in the associated INFILE statement, the INPUT statement reads three input records each time the DATA step executes.

When each observation has multiple input records but values from the last record are not read, you must use a # pointer control in the INPUT statement or N= in the INFILE statement to specify the last input record. For example, if there are four records per observation, but only values from the first two input records are read, use this INPUT statement:

```
input name $ 1-10 #2 age 13-14 #4;
```
When you have advanced to the next record with the / pointer control, use the # pointer control in the INPUT statement or the N= option in the INFILE statement to set the number of records that are read into the input buffer. To move the pointer back to an earlier record, use a # pointer control. For example, this statement requires the #2 pointer control, unless the INFILE statement uses the N= option, to read two records:

```
input a / b #1 @52 c #2;
```

The INPUT statement assigns A a value from the first record. The pointer advances to the next input record to assign B a value. Then the pointer returns from the second record to column 1 of the first record and moves to column 52 to assign C a value. The #2 pointer control identifies two input records for each observation so that the pointer can return to the first record for the value of C.

If the number of input records per observation varies, use the N= option in the INFILE statement to give the maximum number of records per observation. For more information, see the N= option on page 208.

**Reading Past the End of a Line**

When you use @ or + pointer controls with a value that moves the pointer to or past the end of the current record and the next value is to be read from the current column, SAS goes to column 1 of the next record to read it. It also writes this message to the SAS log:

```
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

You can alter the default behavior (the FLOWOVER option) in the INFILE statement. Use the STOPOVER option in the INFILE statement to treat this condition as an error and to stop building the data set.

Use the MISSOVER option in the INFILE statement to set the remaining INPUT statement variables to missing values if the pointer reaches the end of a record.

Use the TRUNCOVER option in the INFILE statement to read column input or formatted input when the last variable that is read by the INPUT statement contains varying-length data.

**Positioning the Pointer before the Record**

When a column pointer control tries to move the pointer to a position before the beginning of the record, the pointer is positioned in column 1. For example, this INPUT statement specifies that the pointer is located in column −2 after the first value is read:

```
data test;
  input a @(a-3) b;
  datalines;
  2
;
```

Therefore, SAS moves the pointer to column 1 after the value of A is read. Both variables A and B contain the same value.

**How Invalid Data Is Handled**

When SAS encounters an invalid character in an input value for the variable indicated, it

- sets the value of the variable that is being read to missing or the value that is specified with the INVALIDDATA= system option. For more information see the “INVALIDDATA= System Option” in *SAS System Options: Reference*.
- prints an invalid data note in the SAS log.
• prints the input line and column number that contains the invalid value in the SAS log. Unprintable characters appear in hexadecimal. To help determine column numbers, SAS prints a rule line above the input line.

• sets the automatic variable _ERROR_ to 1 for the current observation.

The format modifiers for error reporting control the amount of information that is printed in the SAS log. Both the ? and ?? modifier suppress the invalid data message. However, the ?? modifier also resets the automatic variable _ERROR_ to 0. For example, these two sets of statements are equivalent:

- input x ?? 10-12;
- input x ? 10-12;
  _error_=0;

In either case, SAS sets invalid values of X to missing values. For information about the causes of invalid data, see SAS Language Reference: Concepts.

End-of-File
End-of-file occurs when an INPUT statement reaches the end of the data. If a DATA step tries to read another record after it reaches an end-of-file, then execution stops. If you want the DATA step to continue to execute, use the END= or EOF= option in the INFILE statement. Then you can write SAS program statements to detect the end-of-file, and to stop the execution of the INPUT statement but continue with the DATA step. For more information, see the “INFILE Statement” on page 198.

Arrays
The INPUT statement can use array references to read input data values. You can use an array reference in a pointer control if it is enclosed in parentheses. See “Example 6: Positioning the Pointer with a Character Variable” on page 246.

Use the array subscript asterisk (*) to input all elements of a previously defined explicit array. SAS allows single or multidimensional arrays. Enclose the subscript in braces, brackets, or parentheses. The form of this statement is

```
INPUT array-name{*};
```

You can use arrays with list, column, or formatted input. However, you cannot input values to an array that is defined with _TEMPORARY_ and that uses the asterisk subscript. For example, these statements create variables X1 through X100 and assign data values to the variables using the 2. informat:

```
array x{100};
input x{*} 2.;
```

Comparisons
• The INPUT statement reads raw data in external files or data lines that are entered in-stream (following the DATALINES statement) that need to be described to SAS. The SET statement reads a SAS data set, which already contains descriptive information about the data values.

• The INPUT statement reads data while the PUT statement writes data values, text strings, or both to the SAS log or to an external file.

• The INPUT statement can read data from external files; the INFILE statement points to that file and has options that control how that file is read.
Examples

Example 1: Using Multiple Styles of Input in One INPUT Statement
This example uses several input styles in a single INPUT statement:

```sas
data club1;
  input Idno Name $18.
      Team $ 25-30 Startwght Endwght;
datalines;
023 David Shaw red 189 165
049 Amelia Serrano yellow 189 165
... more data lines ...
;
```

The following table identifies the type of input styles.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type of Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idno, Startwght, Endwght</td>
<td>list input</td>
</tr>
<tr>
<td>Name</td>
<td>formatted input</td>
</tr>
<tr>
<td>Team</td>
<td>column input</td>
</tr>
</tbody>
</table>

Example 2: Using a Null INPUT Statement
This example uses an INPUT statement with no arguments. The DATA step copies records from the input file to the output file without creating any SAS variables:

```sas
data _null_;  
  infile file-specification-1;
  file file-specification-2;
  input;
  put _infile_;  
run;
```

Example 3: Holding a Record in the Input Buffer
This example reads a file that contains two types of input data records and creates a SAS data set from these records. One type of data record contains information about a particular college course. The second type of record contains information about the students enrolled in the course. You need two INPUT statements to read the two records and to assign the values to different variables that use different formats. Records that contain class information have a C in column 1; records that contain student information have an S in column 1, as shown here:

```
--------1--------2------
C HIST101 Watson
S Williams 0459
S Flores 5423
C MATH202 Sen
S Lee 7085
```

To know which INPUT statement to use, check each record as it is read. Use an INPUT statement that reads only the variable that tells whether the record contains class or student.
data schedule(drop=type);
   retain Course Professor;
   input type $1. @@;
   if type='C' then
      input course $ professor $;
   else if type='S' then
      do;
      input Name $10. Id;
      output schedule;
   end;
   datalines;
   C HIST101 Watson
   S Williams 0459
   S Flores 5423
   C MATH202 Sen
   S Lee 7085
   ;
   run;

   proc print;
   run;

The first INPUT statement reads the TYPE value from column 1 of every line. Because this INPUT statement ends with a trailing @, the next INPUT statement in the DATA step reads the same line. The IF-THEN statements that follow check whether the record is a class or student line before another INPUT statement reads the rest of the line. The INPUT statements without a trailing @ release the held line. The RETAIN statement saves the values about the particular college course. The DATA step writes an observation to the SCHEDULE data set after a student record is read.

The following output that PROC PRINT generates shows the resulting data set SCHEDULE.

Output 2.11  Data Set Schedule

<table>
<thead>
<tr>
<th>Obs</th>
<th>Course</th>
<th>Professor</th>
<th>Name</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HIST101</td>
<td>Watson</td>
<td>Williams</td>
<td>459</td>
</tr>
<tr>
<td>2</td>
<td>HIST101</td>
<td>Watson</td>
<td>Flores</td>
<td>5423</td>
</tr>
<tr>
<td>3</td>
<td>MATH202</td>
<td>Sen</td>
<td>Lee</td>
<td>7085</td>
</tr>
</tbody>
</table>

Example 4: Holding a Record across Iterations of the DATA Step
This example shows how to create multiple observations for each input data record. Each record contains several NAME and AGE values. The DATA step reads a NAME value and an AGE value, outputs an observation, and then reads another set of NAME and AGE values to output, and so on, until all the input values in the record are processed.

data test;
   input name $ age @@;
   datalines;
   John 13 Monica 12 Sue 15 Stephen 10
The INPUT statement uses the double trailing @ to control the input pointer across iterations of the DATA step. The SAS data set contains six observations.

**Example 5: Positioning the Pointer with a Numeric Variable**

This example uses a numeric variable to position the pointer. A raw data file contains records with the employment figures for several offices of a multinational company. The input data records are

```
----+----1----+----2----+----3----+
 8  New York   1 USA 14
 5   Cary       1 USA 2274
 3 Chicago    1 USA 37
22 Tokyo     5 ASIA 80
 5  Vancouver  2 CANADA 6
 9   Milano    4 EUROPE 123
```

The first column has the column position for the office location. The next numeric column is the region category. The geographic region occurs before the number of employees in that office.

You determine the office location by combining the @ numeric-variable pointer control with a trailing @. To read the records, use two INPUT statements. The first INPUT statement obtains the value for the @ numeric-variable pointer control. The second INPUT statement uses this value to determine the column that the pointer moves to.

```sas
data office (drop=x);
  infile file-specification;
  input x @;
  if 1<=x<=10 then
    input @x City $9.;
  else do;
    put 'Invalid input at line ' _n_;
    delete;
  end;
run;
```

The DATA step writes only five observations to the OFFICE data set. The fourth input data record is invalid because the value of X is greater than 10. Therefore, the second INPUT statement does not execute. Instead, the PUT statement writes a message to the SAS log and the DELETE statement stops processing the observation.

**Example 6: Positioning the Pointer with a Character Variable**

This example uses character variables to position the pointer. The OFFICE data set, created in “Example 5: Positioning the Pointer with a Numeric Variable” on page 246, contains a character variable CITY whose values are the office locations. Suppose you discover that you need to read additional values from the raw data file. By using another DATA step, you can combine the @character-variable pointer control with a trailing @ and the @character-expression pointer control to locate the values.

If the observations in OFFICE are still in the order of the original input data records, you can use this DATA step:

```sas
data office2;
  set office;
  infile file-specification;
  array region {5} $ _temporary_
```
The ARRAY statement assigns initial values to the temporary array elements. These elements correspond to the geographic regions of the office locations. The first INPUT statement uses an @character-variable pointer control. Each record is scanned for the series of characters in the value of CITY for that observation. Then the value of LOCATION is read from the next non-blank column. LOCATION is a numeric category for the geographic region of an office. The second INPUT statement uses an array reference in the @character-expression pointer control to determine the location POPULATION in the input records. The expression also uses the TRIM function to trim trailing blanks from the character value. In this way an exact match is found between the character string in the input data and the value of the array element.

The following output that PROC PRINT generates shows the resulting data set OFFICE2.

**Output 2.12  Data Set OFFICE2**

<table>
<thead>
<tr>
<th>Obs</th>
<th>City</th>
<th>Location</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New York</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Cary</td>
<td>1</td>
<td>2274</td>
</tr>
<tr>
<td>3</td>
<td>Chicago</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>Tokyo</td>
<td>5</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>Vancouver</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Milano</td>
<td>4</td>
<td>123</td>
</tr>
</tbody>
</table>

**Example 7: Moving the Pointer Backward**

This example shows several ways to move the pointer backward.

- This INPUT statement uses the @ pointer control to read a value for BOOK starting at column 26. Then the pointer moves back to column 1 on the same line to read a value for COMPANY:

  ```sas
  input @26 book $ @1 company;
  ```

- These INPUT statements use +numeric-variable or +(expression) to move the pointer backward one column. These two sets of statements are equivalent.

  ```sas
  m=-1;
  input x 1-10 +m y 2.;
  ```

  ```sas
  input x 1-10 +(-1) y 2.;
  ```

**See Also**

**Statements:**
INPUT Statement, Column

Reads input values from specified columns and assigns them to the corresponding SAS variables.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

```
INPUT variable <$> start-column <$> end-column >
<.decimals> <@ | @@> ;
```

Arguments

**variable**

specifies a variable that is assigned input values.

$ indicates that the variable has character values rather than numeric values.

Tip If the variable is previously defined as character, $ is not required.

**start-column**

specifies the first column of the input record that contains the value to read.

**end-column**

specifies the last column of the input record that contains the value to read.

Tip If the variable value occupies only one column, omit end-column.

Example Because end-column is omitted, the values for the character variable GENDER occupy only column 16:

```
input name $ 1-10 pulse 11-13 waist 14-15 gender $ 16;
```

**.decimals**

specifies the number of digits to the right of the decimal if the input value does not contain an explicit decimal point.

Tip An explicit decimal point in the input value overrides a decimal specification in the INPUT statement.

Example “Example 2: Read Input Data Using Decimals” on page 250

@ holds the input record for the execution of the next INPUT statement within the same iteration of the DATA step. This line-hold specifier is called trailing @.
Restriction  The trailing @ must be the last item in the INPUT statement.

Tip  The trailing @ prevents the next INPUT statement from automatically releasing the current input record and reading the next record into the input buffer. It is useful when you need to read from a record multiple times.

See  “Pointer Controls” on page 239

@@ holds the input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called double trailing @.

Restriction  The double trailing @ must be the last item in the INPUT statement.

Tip  The double trailing @ is useful when each input line contains values for several observations.

See  “Using Line-Hold Specifiers” on page 240

Details

When to Use Column Input
With column input, the column numbers that contain the value follow a variable name in the INPUT statement. To read with column input, data values must be in

• the same columns in all the input data records
• standard numeric form or character form.\(^1\)

Useful features of column input are that

• Character values can contain embedded blanks.
• Character values can be from 1 to 32,767 characters long.
• Input values can be read in any order, regardless of their position in the record.
• Values or parts of values can be read multiple times. For example, this INPUT statement reads an ID value in columns 10 through 15 and then reads a GROUP value from column 13:

  \[
  \text{input id 10-15 group 13;}
  \]

• Both leading and trailing blanks within the field are ignored. Therefore, if numeric values contain blanks that represent zeros or if you want to retain leading and trailing blanks in character values, read the value with an informat. See the “INPUT Statement, Formatted” on page 251.

Missing Values
Missing data do not require a place-holder. The INPUT statement interprets a blank field as missing and reads other values correctly. If a numeric or character field contains a single period, the variable value is set to missing.

\(^1\) See SAS Language Reference: Concepts for the definition of standard and nonstandard data values.
Reading Data Lines
SAS always pads the data records that follow the DATALINES statement (in-stream data) to a fixed length in multiples of 80. The CARDIMAGE system option determines whether to read or to truncate data past the 80th column.

Reading Variable-Length Records
By default, SAS uses the FLOWOVER option to read varying-length data records. If the record contains fewer values than expected, the INPUT statement reads the values from the next data record. To read varying-length data, you might need to use the TRUNCOVER option in the INFILE statement. The TRUNCOVER option is more efficient than the PAD option, which pads the records to a fixed length. For more information, see “Reading Past the End of a Line” on page 217.

Examples

Example 1: Read Input Records with Column Input
This DATA step demonstrates how to read input data records with column input:

data scores;
  input name $ 1-18 score1 25-27 score2 30-32
  score3 35-37;
datalines;
Joseph     11   32   76
Mitchel     13   29   82
Sue Ellen   14   27   74;
;

Example 2: Read Input Data Using Decimals
This INPUT statement reads the input data for a numeric variable using two decimal places:

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Statement</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2314</td>
<td>input number 1-5 .2;</td>
<td>23.14</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>.02</td>
</tr>
<tr>
<td>400</td>
<td></td>
<td>4.00</td>
</tr>
<tr>
<td>-140</td>
<td></td>
<td>-1.40</td>
</tr>
<tr>
<td>12.234</td>
<td></td>
<td>12.234</td>
</tr>
<tr>
<td>12.2</td>
<td></td>
<td>12.2</td>
</tr>
</tbody>
</table>

* The decimal specification in the INPUT statement is overridden by the input data value.
See Also

Statements:

• “INPUT Statement” on page 232

INPUT Statement, Formatted

Reads input values with specified informats and assigns them to the corresponding SAS variables.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

INPUT <pointer-control> variable informat. <@ | @@>;
INPUT <pointer-control> (variable-list) (informat-list)
  <@ | @@>;
INPUT <pointer-control> (variable-list) (<n*> informat.)
  <@ | @@>;

Arguments

pointer-control
moves the input pointer to a specified line or column in the input buffer.

See “Column Pointer Controls” on page 234 and “Line Pointer Controls” on page 236

variable
specifies a variable that is assigned input values.

Requirement The (variable-list) is followed by an (informat-list).

Example “Example 1: Formatted Input with Pointer Controls” on page 254

(variable-list)
specifies a list of variables that are assigned input values.

See “How to Group Variables and Informats” on page 253

Example “Example 2: Using Informat Lists” on page 254

informat.
specifies a SAS informat to use to read the variable values.

Tip Decimal points in the actual input values override decimal specifications in a numeric informat.

See SAS Informats in SAS Formats and Informats: Reference
Example  “Example 1: Formatted Input with Pointer Controls” on page 254

(informat-list)
specifies a list of informats to use to read the values for the preceding list of variables

In the INPUT statement, (informat-list) can include

informat
  specifies an informat to use to read the variable values.

pointer-control
  specifies one of these pointer controls to use to position a value: @, #, /, or +.

n*
specifies to repeat n times the next informat in an informat list.

Example  This statement uses the 7.2 informat to read GRADES1, GRADES2, and GRADES3 and the 5.2 informat to read GRADES4 and GRADES5:
  input (grades1-grades5)(3*7.2, 2*5.2);

Restriction  The (informat-list) must follow the (variable-list).

See  “How to Group Variables and Informats” on page 253

Example  “Example 2: Using Informat Lists” on page 254

@
holds an input record for the execution of the next INPUT statement within the same iteration of the DATA step. This line-hold specifier is called trailing @.

Restriction  The trailing @ must be the last item in the INPUT statement.

Tip  The trailing @ prevents the next INPUT statement from automatically releasing the current input record and reading the next record into the input buffer. It is useful when you need to read from a record multiple times.

See  “Using Line-Hold Specifiers” on page 240

@@
holds an input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called double trailing @.

Restriction  The double trailing @ must be the last item in the INPUT statement.

Tip  The double trailing @ is useful when each input line contains values for several observations.

See  “Using Line-Hold Specifiers” on page 240

Details

When to Use Formatted Input
With formatted input, an informat follows a variable name and defines how SAS reads the values of this variable. An informat gives the data type and the field width of an
input value. Informats also read data that are stored in nonstandard form, such as packed
decimal, or numbers that contain special characters such as commas. See “Definition of
Informats” in SAS Formats and Informats: Reference for descriptions of SAS informats.

Simple formatted input requires that the variables be in the same order as their
matching values in the input data. You can use pointer controls to read variables in
any order. For more information, see the “INPUT Statement” on page 232.

Missing Values

Generally, SAS represents missing values in formatted input with a single period for a
numeric value and with blanks for a character value. The informat that you use with
formatted input determines how SAS interprets a blank. For example, $CHAR.w reads
the blanks as part of the value, whereas BZ.w converts a blank to zero.

Reading Variable-Length Records

By default, SAS uses the FLOWOVER option to read varying-length data records. If the
record contains fewer values than expected, the INPUT statement reads the values from
the next data record. To read varying-length data, you might need to use the
TRUNCOVER option in the INFILE statement. For more information, see “Reading
Past the End of a Line” on page 217.

How to Group Variables and Informats

When the input values are arranged in a pattern, you can group the informat list. A
grouped informat list consists of two lists:

• the names of the variables to read enclosed in parentheses
• the corresponding informats separated by either blanks or commas and enclosed in
  parentheses.

Informat lists can make an INPUT statement shorter because the informat list is recycled
until all variables are read and the numbered variable names can be used in abbreviated
form. Using informat lists avoids listing the individual variables.

For example, if the values for the five variables SCORE1 through SCORE5 are stored as
four columns per value without intervening blanks, this INPUT statement reads the
values:

```plaintext
input (score1-score5) (4. 4. 4. 4. 4.);
```

However, if you specify more variables than informats, the INPUT statement reuses the
informat list to read the remaining variables. A shorter version of the previous statement
is

```plaintext
input (score1-score5) (4.);
```

You can use as many informat lists as necessary in an INPUT statement, but do not nest
the informat lists. After all the values in the variable list are read, the INPUT statement
ignores any directions that remain in the informat list. For an example, see “Example 3:
Including More Informat Specifications Than Necessary” on page 254.

The n* modifier in an informat list specifies to repeat the next informat n times. Here is
an example.

```plaintext
input (name score1-score5) ($10. 5*4.);
```

---

1 See SAS Language Reference: Concepts for information about standard and nonstandard data values.
**How to Store Informats**

The informats that you specify in the INPUT statement are not stored with the SAS data set. Informats that you specify with the INFORMAT or ATTRIB statement are permanently stored. Therefore, you can read a data value with a permanently stored informat in a later DATA step without having to specify the informat or use PROC FSEDIT to enter data in the correct format.

**Comparisons**

When a variable is read with formatted input, the pointer movement is similar to the pointer movement of column input. The pointer moves the length that the informat specifies and stops at the next column. To read data with informats that are not aligned in columns, use modified list input. Using modified list input enables you to take advantage of the scanning feature in list input. See “When to Use List Input” on page 257.

**Examples**

**Example 1: Formatted Input with Pointer Controls**

This INPUT statement uses informats and pointer controls:

```sas
data sales;
  infile file-specification;
  input item $10. +5 jan comma5. +5 feb comma5. +5 mar comma5.;
run;
```

It can read these input data records:

```
----+----1----+----2----+----3----+----4
trucks         1,382     2,789     3,556
vans           1,265     2,543     3,987
sedans         2,391     3,011     3,658
```

The value for ITEM is read from the first 10 columns in a record. The pointer stops in column 11. The trailing blanks are discarded and the value of ITEM is written to the program data vector. Next, the pointer moves five columns to the right before the INPUT statement uses the COMMA5. informat to read the value of JAN. This informat uses five as the field width to read numeric values that contain a comma. Once again, the pointer moves five columns to the right before the INPUT statement uses the COMMA5. informat to read the values of FEB and MAR.

**Example 2: Using Informat Lists**

This INPUT statement uses the character informat $10. to read the values of the variable NAME and uses the numeric informat 4. to read the values of the five variables SCORE1 through SCORE5:

```sas
data scores;
  input (name score1-score5) ($10. 5*4.);
datalines;
Whittaker 121 114 137 156 142
Smythe    111 97  122 143 127
;
```

**Example 3: Including More Informat Specifications Than Necessary**

This informat list includes more specifications than are necessary when the INPUT statement executes:
data test;
  input (x y z) (2.,+1);
datalines;
2 24 36
0 20 30
;
The INPUT statement reads the value of X with the 2. informat. Then, the +1 column pointer control moves the pointer forward one column. Next, the value of Y is read with the 2. informat. Again, the +1 column pointer moves the pointer forward one column. Then, the value of Z is read with the 2. informat. For the third iteration, the INPUT statement ignores the +1 pointer control.

See Also

Statements:

• “INPUT Statement” on page 232
• “INPUT Statement, List” on page 255

INPUT Statement, List

Scans the input data record for input values and assigns them to the corresponding SAS variables.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

INPUT <pointer-control> variable <$><>&<@ | @@>;

INPUT <pointer-control> variable <; | & | -> <informat.><@ | @@>;

Arguments

pointer-control
moves the input pointer to a specified line or column in the input buffer.

See “Column Pointer Controls” on page 234 and “Line Pointer Controls” on page 236

Example “Example 2: Reading Character Data That Contains Embedded Blanks” on page 260

variable
specifies a variable that is assigned input values.

$ indicates to store a variable value as a character value rather than as a numeric value.

Tip If the variable is previously defined as character, $ is not required.
## &

indicates that a character value can have one or more single embedded blanks. This format modifier reads the value from the next non-blank column until the pointer reaches two consecutive blanks, the defined length of the variable, or the end of the input line, whichever comes first.

### Restriction

The & modifier must follow the variable name and $ sign that it affects.

### Tip

If you specify an informat after the & modifier, the terminating condition for the format modifier remains two blanks.

### See

“Modified List Input” on page 258

### Example

“Example 2: Reading Character Data That Contains Embedded Blanks” on page 260

## :

enables you to specify an informat that the INPUT statement uses to read the variable value. For a character variable, this format modifier reads the value from the next non-blank column until the pointer reaches the next blank column, the defined length of the variable, or the end of the data line, whichever comes first. For a numeric variable, this format modifier reads the value from the next non-blank column until the pointer reaches the next blank column or the end of the data line, whichever comes first.

### Tips

- If the length of the variable has not been previously defined, then its value is read and stored with the informat length.
- The pointer continues to read until the next blank column is reached. However, if the field is longer than the formatted length, then the value is truncated to the length of variable.

### See

“Modified List Input” on page 258

### Examples

“Example 3: Reading Unaligned Data with Informats” on page 260

“Example 5: Reading Delimited Data with Modified List Input” on page 261

## ~

indicates to treat single quotation marks, double quotation marks, and delimiters in character values in a special way. This format modifier reads delimiters within quoted character values as characters instead of as delimiters and retains the quotation marks when the value is written to a variable.

### Restriction

You must use the DSD option in an INFILE statement. Otherwise, the INPUT statement ignores this option.

### See

“Modified List Input” on page 258

### Example

“Example 5: Reading Delimited Data with Modified List Input” on page 261
The `informat.` specifies an informat to use to read the variable values.

**Tip**  
Decimal points in the actual input values always override decimal specifications in a numeric informat.

**See**  
SAS Informs in *SAS Formats and Informats: Reference*

**Examples**  
“Example 3: Reading Unaligned Data with Informats” on page 260  
“Example 5: Reading Delimited Data with Modified List Input” on page 261

@ holds an input record for the execution of the next INPUT statement within the same iteration of the DATA step. This line-hold specifier is called `trailing @`.

**Restriction**  
The `trailing @` must be the last item in the INPUT statement.

**Tip**  
The `trailing @` prevents the next INPUT statement from automatically releasing the current input record and reading the next record into the input buffer. It is useful when you need to read from a record multiple times.

**See**  
“Using Line-Hold Specifiers” on page 240

@@ holds an input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called `double trailing @`.

**Restriction**  
The `double trailing @` must be the last item in the INPUT statement.

**Tip**  
The `double trailing @` is useful when each input line contains values for several observations.

**See**  
“Using Line-Hold Specifiers” on page 240

**Details**

**When to Use List Input**

List input requires that you specify the variable names in the INPUT statement in the same order that the fields appear in the input data records. SAS scans the data line to locate the next value but ignores additional intervening blanks. List input does not require that the data are located in specific columns. However, you must separate each value from the next by at least one blank unless the delimiter between values is changed. By default, the delimiter for data values is one blank space or the end of the input record. List input will not skip over any data values to read subsequent values, but it can ignore all values after a given point in the data record. However, pointer controls enable you to change the order that the data values are read.

There are two types of list input:

- simple list input
- modified list input.
Modified list input makes the INPUT statement more versatile because you can use a format modifier to overcome several of the restrictions of simple list input. See “Modified List Input” on page 258.

**Simple List Input**

Simple list input places several restrictions on the type of data that the INPUT statement can read:

- By default, at least one blank must separate the input values. Use the DLM= or DLMSTR= option or the DSD option in the INFILE statement to specify a delimiter other than a blank.
- Represent each missing value with a period, not a blank, or two adjacent delimiters.
- Character input values cannot be longer than 8 bytes unless the variable is given a longer length in an earlier LENGTH, ATTRIB, or INFORMAT statement.
- Character values cannot contain embedded blanks unless you change the delimiter.
- Data must be in standard numeric or character format.

**Modified List Input**

List input is more versatile when you use format modifiers. The format modifiers are as follows:

<table>
<thead>
<tr>
<th>Format Modifier</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>reads character values that contain embedded blanks.</td>
</tr>
<tr>
<td>:</td>
<td>reads data values that need the additional instructions that informats can provide but that are not aligned in columns.*</td>
</tr>
<tr>
<td>~</td>
<td>reads delimiters within quoted character values as characters and retains the quotation marks.</td>
</tr>
</tbody>
</table>

* Use formatted input and pointer controls to quickly read data values that are aligned in columns.

For example, use the : modifier with an informat to read character values that are longer than 8 bytes or numeric values that contain nonstandard values.

Because list input interprets a blank as a delimiter, use modified list input to read values that contain blanks. The & modifier reads character values that contain single embedded blanks. However, the data values must be separated by two or more blanks. To read values that contain leading, trailing, or embedded blanks with list input, use the DLM= or DLMSTR= option in the INFILE statement to specify another character as the delimiter. See “Example 5: Reading Delimited Data with Modified List Input” on page 261. If your input data use blanks as delimiters and they contain leading, trailing, or embedded blanks, you might need to use either column input or formatted input. If quotation marks surround the delimited values, you can use list input with the DSD option in the INFILE statement.

---

1. See *SAS Language Reference: Concepts* for the information about standard and nonstandard data values.
Comparisons

How Modified List Input and Formatted Input Differ

*Modified list input* has a scanning feature that can use informats to read data which are not aligned in columns. *Formatted input* causes the pointer to move like that of column input to read a variable value. The pointer moves the length that is specified in the informat and stops at the next column.

This DATA step uses modified list input to read the first data value and formatted input to read the second:

```
data jansales;
   input item : $10. amount comma5.;
datalines;
trucks 1,382
vans 1,235
sedans 2,391
;
```

The value of ITEM is read with modified list input. The INPUT statement stops reading when the pointer finds a blank space. The pointer then moves to the second column after the end of the field, which is the correct position to read the AMOUNT value with formatted input.

Formatted input, on the other hand, continues to read the entire width of the field. This INPUT statement uses formatted input to read both data values:

```
input item $10. +1 amount comma5.;
```

To read this data correctly with formatted input, the second data value must occur after the 10th column of the first value, as shown here:

```
---+---1---+---2
trucks 1,382
vans 1,235
sedans 2,391
```

Also, after the value of ITEM is read with formatted input, you must use the pointer control +1 to move the pointer to the column where the value AMOUNT begins.

When Data Contains Quotation Marks

When you use the DSD option in an INFILE statement, which sets the delimiter to a comma, the INPUT statement removes quotation marks before a value is written to a variable. When you also use the tilde (~) modifier in an INPUT statement, the INPUT statement maintains quotation marks as part of the value.

Examples

**Example 1: Reading Unaligned Data with Simple List Input**

The INPUT statement in this DATA step uses simple list input to read the input data records:

```
data scores;
   input name $ score1 score2 score3 team $;
datalines;
Joe 11 32 76 red
Mitchel 13 29 82 blue
Susan 14 27 74 green
;```
The next INPUT statement reads only the first four fields in the previous data lines, which demonstrates that you are not required to read all the fields in the record:

```
input name $ score1 score2 score3;
```

**Example 2: Reading Character Data That Contains Embedded Blanks**

The INPUT statement in this DATA step uses the & format modifier with list input to read character values that contain embedded blanks.

```sas
data list;
   infile file-specification;
   input name $ & score;
run;
```

It can read these input data records:

```
----+----1----+----2----+----3----+
Joseph   11 Joergensen  red
Mitchel  13 Mc Allister  blue
Su Ellen  14 Fischer-Simon  green
```

The & modifier follows the variable that it affects in the INPUT statement. Because this format modifier follows NAME, at least two blanks must separate the NAME field from the SCORE field in the input data records.

You can also specify an informat with a format modifier, as shown here:

```
input name $ & +3 lastname & $15. team $;
```

In addition, this INPUT statement reads the same data to demonstrate that you are not required to read all the values in an input record. The +3 column pointer control moves the pointer past the score value in order to read the value for LASTNAME and TEAM.

**Example 3: Reading Unaligned Data with Informats**

This DATA step uses modified list input to read data values with an informat:

```sas
data jansales;
   input item : $10. amount;
   datalines;
   trucks 1382
   vans 1235
   sedans 2391
;
```

The $10. informat allows a character variable of up to ten characters to be read.

**Example 4: Reading Comma-Delimited Data with List Input and an Informat**

This DATA step uses the DELIMITER= option in the INFILE statement to read list input values that are separated by commas instead of blanks. The example uses an informat to read the date, and a format to write the date.

```sas
data scores2;
   length Team $ 14;
   infile datalines delimiters=',';
   input Name $ Score1-Score3 Team $ Final_Date:MMDDYY10.;
   format final_date weekdate17.;
   datalines;
```

260  Chapter 2  •  Dictionary of SAS Statements
Mitchell,13,29,82,Blue Bunnies,4/5/2007
Susan,14,27,74,Green Gazelles,11/13/2007
;
proc print data=scores2;
  var Name Team Score1-Score3 Final_Date;
  title 'Soccer Player Scores';
run;

Output 2.13  Output from Comma-Delimited Data

Soccer Player Scores

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Team</th>
<th>Score1</th>
<th>Score2</th>
<th>Score3</th>
<th>Final_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Red Racers</td>
<td>11</td>
<td>32</td>
<td>76</td>
<td>Sat, Feb 3, 2007</td>
</tr>
<tr>
<td>2</td>
<td>Mitchell</td>
<td>Blue Bunnies</td>
<td>13</td>
<td>29</td>
<td>82</td>
<td>Thu, Apr 5, 2007</td>
</tr>
<tr>
<td>3</td>
<td>Susan</td>
<td>Green Gazelles</td>
<td>14</td>
<td>27</td>
<td>74</td>
<td>Tue, Nov 13, 2007</td>
</tr>
</tbody>
</table>

Example 5: Reading Delimited Data with Modified List Input
This DATA step uses the DSD option in an INFILE statement and the tilde (~) format modifier in an INPUT statement to retain the quotation marks in character data and to read a character in a string that is enclosed in quotation marks as a character instead of as a delimiter.

data scores;
  infile datalines dsd;
  input Name : $9. Score1-Score3
              Team ~ $25. Div $;
  datalines;
Mitchel,13,29,82,"Blue Bunnies, Richmond",AAA
Sue Ellen,14,27,74,"Green Gazelles, Atlanta",AA
;
proc print; run;
The output that PROC PRINT generates shows the resulting SCORES data set. The values for TEAM contain the quotation marks.

Output 2.14  SCORES Data Set

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Score1</th>
<th>Score2</th>
<th>Score3</th>
<th>Team</th>
<th>Div</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joseph</td>
<td>11</td>
<td>32</td>
<td>76</td>
<td>&quot;Red Racers, Washington&quot;</td>
<td>AAA</td>
</tr>
<tr>
<td>2</td>
<td>Mitchel</td>
<td>13</td>
<td>29</td>
<td>82</td>
<td>&quot;Blue Bunnies, Richmond&quot;</td>
<td>AAA</td>
</tr>
<tr>
<td>3</td>
<td>Sue Ellen</td>
<td>14</td>
<td>27</td>
<td>74</td>
<td>&quot;Green Gazelles, Atlanta&quot;</td>
<td>AA</td>
</tr>
</tbody>
</table>
INPUT Statement, Named

Reads data values that appear after a variable name that is followed by an equal sign and assigns them to corresponding SAS variables.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

INPUT <pointer-control> variable= <$> <@ | @@>;
INPUT <pointer-control> variable=informat. <@ | @@>;
INPUT variable= <$> start-column <end-column> <decimals> <@ | @@>;

Arguments

pointer-control
moves the input pointer to a specified line or column in the input buffer.

See “Column Pointer Controls” on page 234 and “Line Pointer Controls” on page 236

variable= specifies a variable whose value is read by the INPUT statement. In the input data record, the field has the form

variable=value

Example “Example 3: Using Named Input with Another Input Style” on page 265

$ indicates to store a variable value as a character value rather than as a numeric value.

Tip If the variable is previously defined as character, $ is not required.

Example “Example 3: Using Named Input with Another Input Style” on page 265

informat.
specifies an informat that indicates the data type of the input values, but not how the values are read.
Tip | Use the INFORMAT statement to associate an informat with a variable.
---|---
See | SAS Informs in *SAS Formats and Informats: Reference*
Example | “Example 3: Using Named Input with Another Input Style” on page 265

**start-column**
specifies the column that the INPUT statement uses to begin scanning in the input data records for the variable. The variable name does not have to begin here.

**end-column**
determines the default length of the variable.

**.decimals**
specifies the number of digits to the right of the decimal if the input value does not contain an explicit decimal point.

Tip | An explicit decimal point in the input value overrides a decimal specification in the INPUT statement.

@ holds an input record for the execution of the next INPUT statement within the same iteration of the DATA step. This line-hold specifier is called *trailing @*.

Restriction | The trailing @ must be the last item in the INPUT statement.
Tip | The trailing @ prevents the next INPUT statement from automatically releasing the current input record and reading the next record into the input buffer. It is useful when you need to read from a record multiple times.

See | “Using Line-Hold Specifiers” on page 240

@@ holds an input record for the execution of the next INPUT statement across iterations of the DATA step. This line-hold specifier is called *double trailing @*.

Restriction | The double trailing @ must be the last item in the INPUT statement.
Tip | The double trailing @ is useful when each input line contains values for several observations.

See | “Using Line-Hold Specifiers” on page 240

**Details**

**When to Use Named Input**
Named input reads the input data records that contain a variable name followed by an equal sign and a value for the variable. The INPUT statement reads the input data record at the current location of the input pointer. If the input data records contain data values at the start of the record that the INPUT statement cannot read with named input, use another input style to read them. However, once the INPUT statement starts to read named input, SAS expects that all the remaining values are in this form. See “Example 3: Using Named Input with Another Input Style” on page 265.

You do not have to specify the variables in the INPUT statement in the same order that they occur in the data records. Also, you do not have to specify a variable for each field.
in the record. However, if you do not specify a variable in the INPUT statement that
another statement uses (for example, ATTRIB, FORMAT, INFORMAT, LENGTH
statement) and it occurs in the input data record, the INPUT statement automatically
reads the value. SAS writes a note to the log that the variable is uninitialized.

When you do not specify a variable for all the named input data values, SAS sets
_ERROR_ to 1 and writes a note to the log. Here is an example.

```
data list;
  input name= $ age=;
  datalines;
  name=John age=34  gender=M
;
```

The note that SAS writes to the log states that GENDER is not defined and _ERROR_ is
set to 1.

**Restrictions**

- After you start to read with named input, you cannot switch to another input style or
  use pointer controls. All the remaining values in the input data record must be in the
  form variable=value. SAS treats the values that are not in named input form as
  invalid data.

- If named input values continue after the end of the current input line, use a slash (/)
  at the end of the input line. The slash tells SAS to move the pointer to the next line
  and to continue to read with named input. For example,

  ```
  input name= $ age=;
  can read this input data record:
  ```

  ```
  name=John / age=34
  ```

- If you use named input to read character values that contain embedded blanks, put
two blanks before and after the data value, as you would with list input. See
  “Example 4: Reading Character Variables with Embedded Blanks” on page 265.

- You cannot reference an array with an asterisk or an expression subscript.

**Examples**

**Example 1: Using List and Named Input**

This DATA step uses list input with named input to read input data records.

```
data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8. ;
  input id name= gender= age= dob=;
  datalines;
4798 name=COLIN gender=m age=23 dob=16/02/75
2653 name=MICHELE gender=f age=46 dob=17/02/73
;
proc print data=list; run;
```

The INPUT statement uses list input to read the ID variable. The remaining variables
NAME, GENDER, AGE, and DOB are read with named input. The LENGTH statement
prevents the INPUT statement from truncating the character values for the variable name
to a length of eight.
Example 2: Using Named Input with Variables in Random Order

Using the same data as in the previous example, this DATA step also uses list input and named input to read input data records. However, in this example, the order of the values in the data is different for the two rows, except for the ID value, which must come first.

```sas
data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8.;
  input id dob= name= age= gender=;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE dob=17/02/73 age=46 gender=f
;
proc print data=list; run;
```

Example 3: Using Named Input with Another Input Style

This DATA step uses list input and named input to read input data records:

```sas
data list;
  input id name=$20. gender=$;
  informat dob ddmmyy8.;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE age=46 gender=f
;
proc print data=list; run;
```

The INPUT statement uses list input to read the first variable, ID. The remaining variables NAME, GENDER, and DOB are read with named input. These variables are not read in order. The $20. informat with NAME= prevents the INPUT statement from truncating the character value to a length of eight. The INPUT statement reads the DOB= field because the INFORMAT statement refers to this variable. It skips the AGE= field altogether. SAS writes notes to the log that DOB is uninitialized, AGE is not defined, and _ERROR_ is set to 1.

Example 4: Reading Character Variables with Embedded Blanks

This DATA step reads character variables that contain embedded blanks with named input:

```sas
data list2;
  informat header $30. name $15.;
  input header= name=;
  datalines;
header= age=60 AND UP name=PHILIP
;
```

Two spaces precede and follow the value of the variable HEADER, which is AGE=60 AND UP. The field also contains an equal sign.

See Also

Statements:

- “INPUT Statement” on page 232
**KEEP Statement**

Specifies the variables to include in output SAS data sets.

- **Valid in:** DATA step
- **Category:** Information
- **Type:** Declarative

**Syntax**

```
KEEP variable-list;
```

**Arguments**

`variable-list`

specifies the names of the variables to write to the output data set.

**Tip**  List the variables in any form that SAS allows.

**Details**

The KEEP statement causes a DATA step to write only the variables that you specify to one or more SAS data sets. The KEEP statement applies to all SAS data sets that are created within the same DATA step and can appear anywhere in the step. If no KEEP or DROP statement appears, all data sets that are created in the DATA step contain all variables.

**Note:** Do not use both the KEEP and DROP statements within the same DATA step.

**Comparisons**

- The KEEP statement cannot be used in SAS PROC steps. The KEEP=`data set option` can.
- The KEEP statement applies to all output data sets that are named in the DATA statement. To write different variables to different data sets, you must use the KEEP=`data set option`.
- The DROP statement is a parallel statement that specifies variables to omit from the output data set.
- The KEEP and DROP statements select variables to include in or exclude from output data sets. The subsetting IF statement selects observations.
- Do not confuse the KEEP statement with the RETAIN statement. The RETAIN statement causes SAS to hold the value of a variable from one iteration of the DATA step to the next iteration. The KEEP statement does not affect the value of variables but only specifies which variables to include in any output data sets.

**Examples**

**Example 1: KEEP Statement Basic Usage**

These examples show the correct syntax for listing variables in the KEEP statement:
Example 2: Keeping Variables in the Data Set
This example uses the KEEP statement to include only the variables NAME and AVG in the output data set. The variables SCORE1 through SCORE20, from which AVG is calculated, are not written to the data set AVERAGE.

data average;
  keep name avg;
  infile file-specification;
  input name $ score1-score20;
  avg=mean(of score1-score20);
run;

See Also

Data Set Options:
- “KEEP= Data Set Option” in SAS Data Set Options: Reference

Statements:
- “DROP Statement” on page 68
- “IF Statement, Subsetting” on page 187
- “RETAIN Statement” on page 378

LABEL Statement
Assigns descriptive labels to variables.

- Valid in: DATA step
- Category: Information
- Type: Declarative

Syntax

```
LABEL variable-1=label-1…<variable-n=label-n>;
LABEL variable-1=' '…<variable-n=' '>
```

Arguments

- **variable**
  - specifies the variable that you want to label.
  - **Tip**: You can specify additional pairs of labels and variables.

- **label**
  - specifies a label of up to 256 characters, including blank spaces.
Restrictions

If the label includes a semicolon (;) or an equal sign (=), you must enclose the label in either single or double quotation marks.

If the label includes single quotation marks (‘), then you must enclose the label in double quotation marks.

There is partial support for variable name substitution with the ActiveX and Java devices. When the label text for the variable exceeds the space allowed, the variable name is used instead by procedures that are labeling an axis or a legend title. With the exception of the SAS/GRAPH GPLOT procedure, the variable name will not be used when ActiveX or Java devices are specified.

Tips

You can specify additional pairs of labels and variables.

For more information about including quotation marks as part of the label, see “Character Constants” in SAS Language Reference: Concepts.

'' removes a label from a variable. Enclose a single blank space in quotation marks to remove an existing label.

Details

Using a LABEL statement in a DATA step permanently associates labels with variables by affecting the descriptor information of the SAS data set that contains the variables. You can associate any number of variables with labels in a single LABEL statement.

You can use a LABEL statement in a PROC step, but the rules are different. See the Base SAS Procedures Guide for more information.

Comparisons

Both the ATTRIB and LABEL statements can associate labels with variables and change a label that is associated with a variable.

Examples

Example 1: Specifying Labels

Here are several LABEL statements:

- label compound=Type of Drug;
- label date="Today's Date";
- label n='Mark''s Experiment Number';
- label score1="Grade on April 1 Test"
  score2="Grade on May 1 Test";

Example 2: Removing a Label

This example removes an existing label:

data rtest;
  set rtest;
  label x=' ';

run;

See Also

Statements:
• “ATTRIB Statement” on page 27

label: Statement

Identifies a statement that is referred to by another statement.

Valid in: DATA step
Category: Control
Type: Declarative

Syntax
label: statement;

Arguments

label
specifies any SAS name, which is followed by a colon (:). You must specify the label argument.

statement
specifies any executable statement, including a null statement (;). You must specify the statement argument.

Restrictions
No two statements in a DATA step can have the same label.

Tip
If a statement in a DATA step is labeled, it should be referenced by a statement or option in the same step.

A null statement can have a label:
ABC: ;

Details

The statement label identifies the destination of either a GO TO statement, a LINK statement, the HEADER= option in a FILE statement, or the EOF= option in an INFILE statement.

Comparisons

The LABEL statement assigns a descriptive label to a variable. A statement label identifies a statement or group of statements that are referred to in the same DATA step by another statement, such as a GO TO statement.
Example: Jumping to Another Statement

In this example, if Stock=0, the GO TO statement causes SAS to jump to the statement that is labeled reorder. When Stock is not 0, execution continues to the RETURN statement and then returns to the beginning of the DATA step for the next observation.

```sas
data Inventory Order;
  input Item $ Stock @;
  /* go to label reorder: */
  if Stock=0 then go to reorder;
  output Inventory;
  return;
  /* destination of GO TO statement */
reorder: input Supplier $;
  put 'ORDER ITEM ' Item ' FROM ' Supplier;
  output Order;
datalines;
milk  0 A
bread 3 B
;```

See Also

Statements:
- “GO TO Statement” on page 185
- “LINK Statement” on page 293

Statement Options:
- “EOF=label” on page 205 in the INFILE statement
- “HEADER=label” on page 82 option in the FILE statement

---

**LEAVE Statement**

Stops processing the current loop and resumes with the next statement in the sequence.

**Valid in:** DATA step  
**Category:** Control  
**Type:** Executable

**Syntax**

```sas
LEAVE;
```

**Without Arguments**

The LEAVE statement stops the processing of the current DO loop or SELECT group and continues DATA step processing with the next statement following the DO loop or SELECT group.
Details

You can use the LEAVE statement to exit a DO loop or SELECT group prematurely based on a condition.

Comparisons

• The LEAVE statement causes processing of the current loop to end. The CONTINUE statement stops the processing of the current iteration of a loop and resumes with the next iteration.

• You can use the LEAVE statement in a DO loop or in a SELECT group. You can use the CONTINUE statement only in a DO loop.

Example: Stop Processing a DO Loop under a Given Condition

This DATA step demonstrates using the LEAVE statement to stop the processing of a DO loop under a given condition. In this example, the IF/THEN statement checks the value of BONUS. When the value of BONUS reaches 500, the maximum amount allowed, the LEAVE statement stops the processing of the DO loop.

```plaintext
data week;
  input name $ idno start_yr status $ dept $;
  bonus=0;
  do year= start_yr to 1991;
    if bonus ge 500 then leave;
    bonus+50;
  end;
datalines;
Jones 9011 1990 PT PUB
Thomas 876 1976 PT HR
Barnes 7899 1991 PT TECH
Harrell 1250 1975 PT HR
Richards 1002 1990 PT DEV
Kelly 85 1981 PT PUB
Stone 091 1990 PT MAIT
;```

See Also

Statements:

• “DO Statement” on page 61
• “SELECT Statement” on page 391

LENGTH Statement

Specifies the number of bytes for storing variables.

Valid in: DATA step
Category: Information
Type: Declarative
Avoid shortening numeric variables that contain fractions. The precision of a numeric variable is closely tied to its length, especially when the variable contains fractional values. You can safely shorten variables that contain integers according to the rules that are given in the SAS documentation for your operating environment, but shortening variables that contain fractions might eliminate important precision.

Syntax

LENGTH variable-specification(s) <DEFAULT=n>;

Arguments

variable-specification

is a required argument and has the form

variable(s)$>length

variable

specifies one or more variables that are to be assigned a length. This includes any variables in the DATA step, including those dropped from the output data set.

Restriction

Array references are not allowed.

Tip

If the variable is character, the length applies to the program data vector and the output data set. If the variable is numeric, the length applies only to the output data set.

$ 

specifies that the preceding variables are character variables.

Default

SAS assumes that the variables are numeric.

length

specifies a numeric constant that is the number of bytes used for storing variable values.

Range

For numeric variables, 2 to 8 or 3 to 8, depending on your operating environment. For character variables, 1 to 32767 under all operating environments.

DEFAULT=n

changes the default number of bytes that SAS uses to store the values of any newly created numeric variables.

Default

8

Range

2 to 8 or 3 to 8, depending on your operating environment.

Details

In general, the length of a variable depends on the following:

- whether the variable is numeric or character
- how the variable was created
- whether a LENGTH or ATTRIB statement is present.
Subject to the rules for assigning lengths, lengths that are assigned with the LENGTH statement can be changed in the ATTRIB statement and vice versa. See “SAS Variables” in SAS Language Reference: Concepts for information about assigning lengths to variables.

Operating Environment Information
Valid variable lengths depend on your operating environment. For details, see the SAS documentation for your operating environment.

Comparisons
The ATTRIB statement can assign the length as well as other attributes of variables.

Example
This example uses a LENGTH statement to set the length of the character variable NAME to 25 bytes. The LENGTH statement also changes the default number of bytes that SAS uses to store the values of newly created numeric variables from 8 to 4 bytes. The TRIM function removes trailing blanks from LASTNAME before it is concatenated with these items:
• a comma (,)
• a blank space
• the value of FIRSTNAME
If you omit the LENGTH statement, SAS sets the length of NAME to 32 bytes.

```sas
data testlength;
  informat FirstName LastName $15. n1 6.2;
  input firstname lastname n1 n2;
  length name $25 default=4;
  name=trim(lastname)||', '||firstname;
  datalines;
Alexander Robinson 35 11
;
proc contents data=testlength;
run;
proc print data=testlength;
run;
```

The following output shows a partial listing from PROC CONTENTS, as well as the report that PROC PRINT generates.
Output 2.15  Partial PROC CONTENTS for TESTLENGTH

<table>
<thead>
<tr>
<th></th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FirstName</td>
<td>Char</td>
<td>15</td>
<td>$15.</td>
</tr>
<tr>
<td>2</td>
<td>LastName</td>
<td>Char</td>
<td>15</td>
<td>$15.</td>
</tr>
<tr>
<td>3</td>
<td>n1</td>
<td>Num</td>
<td>4</td>
<td>6.2</td>
</tr>
<tr>
<td>4</td>
<td>n2</td>
<td>Num</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>name</td>
<td>Char</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

Output 2.16  Setting the Length of a Variable

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>FirstName</th>
<th>LastName</th>
<th>n1</th>
<th>n2</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alexander</td>
<td>Robinson</td>
<td>0.35000</td>
<td>11</td>
<td>Robinson, Alexander</td>
</tr>
</tbody>
</table>

See Also

- For information about the use of the LENGTH statement in PROC steps, see the Base SAS Procedures Guide.

Statements:

- “ATTRIB Statement” on page 27

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 Language Reference: Concepts.

See: LIBNAME Statement under Windows, UNIX, and z/OS
Syntax

Form 1: `LIBNAME libref <engine> 'SAS-library' <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>) <options>;`

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.

'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 details 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 284

Example: “Example 2: Logically Concatenating SAS Libraries” on page 286

`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.
For a list of valid engines, see the SAS documentation for your operating environment. For more background information, see “SAS Engines” in SAS Language Reference: Concepts.

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.

LIBNAME Options

ACCESS=READONLY|TEMP

READONLY

Assigns a read-only attribute to an entire SAS library. SAS will 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 will 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.

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 285

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 AUTHADMIN=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 | YES | 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).

**YES | 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.

**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 the COPY Statement in the DATASETS procedure. This interaction does not apply when using SAS/SHARE or SAS/CONNECT.

CVPBYTES=\texttt{bytes}  specifies the number of bytes to expand character variable lengths when processing a SAS data file that requires transcoding.


CVPENGINE|CVPENG=\texttt{engine}  specifies the engine to use in order to process a SAS data file that requires transcoding.


CVPMULTIPLIER|CVPMULT=\texttt{multiplier}  specifies a multiplier value in order to expand character variable lengths when processing a SAS data file that requires transcoding.


EXTENDOBSCOUNTER=\texttt{YES} | \texttt{NO}  specifies whether to extend the maximum observation count in output SAS data files for a SAS library.

\texttt{YES}  requests an enhanced file format in a newly created SAS data file that counts observations beyond the 32-bit limitation. Although this SAS data file is created for an operating environment that stores the number of observations with a 32-bit integer, the file behaves like a 64-bit file with respect to counters. This is the default.

Restrictions  A SAS data file that is created with an extended observation count is incompatible with releases prior to SAS 9.3.

Use with the BASE engine only.

EXTENDOBSCOUNTER=\texttt{YES} is valid only for an output SAS data file whose internal data representation stores the observation count as a 32-bit integer. EXTENDOBSCOUNTER=\texttt{YES} is ignored for SAS data files with a 64-bit integer. For a table that lists the operating environments and OUTREP= values that are appropriate with EXTENDOBSCOUNTER=\texttt{YES}, see “When

The extended observation count attribute cannot be inherited by a new file. If you create a new file without an extended observation count attribute, you must specify EXTENDOBSCOUNTER=NO for the new file. For more information, see “Extended Observation Count Behavior Considerations” in SAS Language Reference: Concepts.

See “Extending the Observation Count for a 32-Bit SAS Data File” in SAS Language Reference: Concepts

NO specifies that the maximum observation count in a newly created SAS data file is determined by the long integer size for the operating environment. In operating environments with a 32-bit integer, the maximum number is $2^{31} – 1$ or approximately two billion observations (2,147,483,647). In operating environments with a 64-bit integer, the number is $2^{63} – 1$ or approximately 9.2 quintillion observations.

Alias EOC=

Default YES

INENCODING=ANY | ASCIIANY | EBCDICANY | encoding-value overrides the encoding when you are reading (input processing) SAS data sets in the SAS library.


OUTENCODING= OUTENCODING=ANY | ASCIIANY | EBCDICANY | encoding-value overrides the encoding when you are creating (output processing) SAS data sets in the SAS library.


OUTREP=format specifies the data representation for the SAS library, which is the form in which data is stored in a particular operating environment. Different operating environments use different standards or conventions for storing floating-point numbers (for example, IEEE or IBM mainframe); for character encoding (for example, ASCII or EBCDIC); for the ordering of bytes in memory (for example, big Endian or little Endian); for word alignment (for example, 4-byte boundaries or 8-byte boundaries); for integer data-type length (for example, 16-bit, 32-bit, or 64-bit); and for doubles (for example, byte-swapped or not).

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 “Processing Data Using Cross-Environment Data Access (CEDA)” in SAS Language Reference: Concepts.
Values for OUTREP= are listed in the following table.

<table>
<thead>
<tr>
<th>OUTREP= Value</th>
<th>Alias*</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA_TRU64</td>
<td>ALPHA_OSF</td>
<td>Tru64 UNIX</td>
</tr>
<tr>
<td>ALPHA_VMS_32</td>
<td>ALPHA_VMS</td>
<td>OpenVMS Alpha</td>
</tr>
<tr>
<td>ALPHA_VMS_64</td>
<td></td>
<td>OpenVMS Alpha</td>
</tr>
<tr>
<td>HP_IA64</td>
<td>HP_ITANIUM</td>
<td>HP-UX for the Itanium Processor Family Architecture</td>
</tr>
<tr>
<td>HP_UX_32</td>
<td>HP_UX</td>
<td>HP-UX for PA-RISC</td>
</tr>
<tr>
<td>HP_UX_64</td>
<td></td>
<td>HP-UX for PA-RISC, 64-bit</td>
</tr>
<tr>
<td>INTEL_ABI</td>
<td></td>
<td>ABI for Intel architecture</td>
</tr>
<tr>
<td>LINUX_32</td>
<td>LINUX</td>
<td>Linux for Intel architecture</td>
</tr>
<tr>
<td>LINUX_IA64</td>
<td></td>
<td>Linux for Itanium-based systems</td>
</tr>
<tr>
<td>LINUX_X86_64</td>
<td></td>
<td>Linux for x64</td>
</tr>
<tr>
<td>MIPS_ABI</td>
<td></td>
<td>MIPS ABI</td>
</tr>
<tr>
<td>MVS_32</td>
<td>MVS</td>
<td>31-bit SAS on z/OS</td>
</tr>
<tr>
<td>MVS_64_BFP</td>
<td></td>
<td>64-bit SAS on z/OS</td>
</tr>
<tr>
<td>OS2</td>
<td></td>
<td>OS/2 for Intel</td>
</tr>
<tr>
<td>RS_6000_AIX_32</td>
<td>RS_6000_AIX</td>
<td>AIX</td>
</tr>
<tr>
<td>RS_6000_AIX_64</td>
<td></td>
<td>AIX</td>
</tr>
<tr>
<td>SOLARIS_32</td>
<td>SOLARIS</td>
<td>Solaris for SPARC</td>
</tr>
<tr>
<td>SOLARIS_64</td>
<td></td>
<td>Solaris for SPARC</td>
</tr>
<tr>
<td>SOLARIS_X86_64</td>
<td></td>
<td>Solaris for x64</td>
</tr>
<tr>
<td>VAX_VMS</td>
<td></td>
<td>OpenVMS VAX</td>
</tr>
<tr>
<td>VMS_IA64</td>
<td></td>
<td>OpenVMS on HP Integrity</td>
</tr>
<tr>
<td>WINDOWS_32</td>
<td>WINDOWS</td>
<td>32-bit SAS on Microsoft Windows</td>
</tr>
</tbody>
</table>
OUTREP= Value | Alias* | Environment
--- | --- | ---
WINDOWS_64 | 64-bit SAS on Microsoft Windows (for both Itanium-based systems and x64)

* It is recommended that you use the current values. The aliases are available for compatibility only.

**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 the 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.

Transcoding could result in character data loss when encodings are incompatible. For more information, see the SAS National Language Support (NLS): Reference Guide.

**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 like-named temporary or permanent SAS data sets when the new one is empty.

YES

specifies that a new empty data set with a given name replace an existing data set with the same name. 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 with a given name not replace an existing data set with the same name.

Tips  Use REPEMPTY=NO to prevent the following syntax error from replacing the existing data set MYLIB.B with the new empty data set MYLIB.B that is created by mistake:

```sas
libname libref SAS-library REPEMPTY=NO;
data mylib.a set mylib.b;
```

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*

**Engine Host Options**

*engine-host-options*

are one or more options that are listed in the general form *keyword=value*.

**Restriction**

When concatenating libraries, you cannot specify options that are specific to an engine or an operating environment.

**Operating environment**

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

**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:

```sas
LIBNAME libref 'SAS-library';
```

See “Example 1: Assigning and Using a Libref” on page 286.
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 286.

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 286.

```
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 will 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 the “**DLCREATEDIR System Option**” in *SAS System Options: Reference*.

**z/OS Specifics**

For more information, see the “**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 the *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 the “CATNAME Statement” on page 38.
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:

```sas
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

- “How Many Characters Can I Use When I Measure SAS Name Lengths in Bytes?” in *SAS Language Reference: Concepts*

Data Set Options:


Statements:

- “CATNAME Statement” on page 38 for a discussion of concatenating SAS catalogs
- “FILENAME Statement” on page 94
- LIBNAME option character variable attributes used to transcode SAS files
- 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 Statement, 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 for the 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.

Base 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 for 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.

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.

'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.

eengine
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.

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 the “LIBNAME Statement” on page 274.

WebDAV Specific Options

WEBDAV
specifies that the libref access a WebDAV server.

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

<table>
<thead>
<tr>
<th><strong>UID</strong></th>
</tr>
</thead>
</table>

### Tip

If PROMPT is specified, but USER= is not, then the user is prompted for an ID as well as a password.

### 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.

### 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*.

### PROXY= *url*

specifies the Uniform Resource Locator (URL) for the proxy server in one of these forms:

- "http://hostname"
- "http://hostname:port"

### 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

Details

Data Set Options That Function Differently with a WebDAV Server

The following 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 2.6 Data Set Option Functionality with a WebDAV Server

<table>
<thead>
<tr>
<th>Data Set Option</th>
<th>WebDAV Storage Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTLLEV=</td>
<td><strong>LIB</strong> 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. <strong>MEM</strong> 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. <strong>REC</strong> is not supported. WebDAV allows updates to the entire data set only.</td>
</tr>
<tr>
<td>FILECLOSE</td>
<td>The VxTAPE engine is not supported. Therefore, this option is ignored.</td>
</tr>
<tr>
<td>GENMAX=</td>
<td>This functionality is not supported because the maximum number of revisions to keep cannot be specified in the WebDAV server.</td>
</tr>
<tr>
<td>GENNUM=</td>
<td>This functionality is not supported in WebDAV.</td>
</tr>
<tr>
<td>IDXNAME=</td>
<td>Users can specify an index to use if one exists.</td>
</tr>
<tr>
<td>INDEX=</td>
<td>Indexes can be created in the local cache and saved on the WebDAV server.</td>
</tr>
<tr>
<td>TOBSNO=</td>
<td>Remote engines are not supported. Therefore, this option is ignored.</td>
</tr>
</tbody>
</table>

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 will be 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**

The following 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 166
- “LIBNAME Statement” on page 274

---

**LINK Statement**

Directs program execution immediately to the statement label that is specified and, if followed by a RETURN statement, returns execution to the statement that follows the LINK statement.

**Valid in:** DATA step

**Category:** Control

**Type:** Executable

**Syntax**

```
LINK label;
```
**Arguments**

`label`  
specifies a statement label that identifies the LINK destination. You must specify the `label` argument.

**Details**

The LINK statement tells SAS to jump immediately to the statement label that is indicated in the LINK statement and to continue executing statements from that point until a RETURN statement is executed. The RETURN statement sends program control to the statement immediately following the LINK statement.

The LINK statement and the destination must be in the same DATA step. The destination is identified by a statement label in the LINK statement.

The LINK statement can branch to a group of statements that contain another LINK statement. This arrangement is known as nesting. To avoid infinite looping, SAS has set a default number of nested LINK statements. You can have up to 10 LINK statements with no intervening RETURN statements. When more than one LINK statement has been executed, a RETURN statement tells SAS to return to the statement that follows the last LINK statement that was executed. However, you can use the /STACK option in the DATA statement to increase the number of nested LINK statements.

**Comparisons**

The difference between the LINK statement and the GO TO statement is in the action of a subsequent RETURN statement. A RETURN statement after a LINK statement returns execution to the statement that follows LINK. A RETURN statement after a GO TO statement returns execution to the beginning of the DATA step, unless a LINK statement precedes GO TO. In that case, execution continues with the first statement after LINK. In addition, a LINK statement is usually used with an explicit RETURN statement, whereas a GO TO statement is often used without a RETURN statement.

When your program executes a group of statements at several points in the program, using the LINK statement simplifies coding and makes program logic easier to follow. If your program executes a group of statements at only one point in the program, using DO-group logic rather than LINK-RETURN logic is simpler.

**Example: Diverting Program Execution**

In this example, when the value of variable TYPE is `aluv`, the LINK statement diverts program execution to the statements that are associated with the label CALCUL. The program executes until it encounters the RETURN statement, which sends program execution back to the first statement that follows LINK. SAS executes the assignment statement, writes the observation, and then returns to the top of the DATA step to read the next record. When the value of TYPE is not `aluv`, SAS executes the assignment statement, writes the observation, and returns to the top of the DATA step.

```sas
data hydro;
  input type $ depth station $;
  /* link to label calcu: */
  if type = 'aluv' then link calcu;
  date=today();
  /* return to top of step */
  return;
  calcu: if station='site_1'
    then elevatn=6650-depth;
```

else if station='site_2'
    then elevatn=5500-depth;
    /* return to date=today(); */
    return;
datalines;
   aluv 523 site_1
   uppa 234 site_2
   aluv 666 site_2
   ...more data lines...

See Also

Statements:

• “DATA Statement” on page 44
• “DO Statement” on page 61
• “GO TO Statement” on page 185
• “label: Statement” on page 269
• “RETURN Statement” on page 382

LIST Statement

Writes to the SAS log the input data record for the observation that is being processed.

Valid in:  DATA step
Category:  Action
Type:     Executable

Syntax

LIST;

Without Arguments

The LIST statement causes the input data record for the observation being processed to be written to the SAS log.

Details

The LIST statement operates only on data that is read with an INPUT statement; it has no effect on data that is read with a SET, MERGE, MODIFY, or UPDATE statement.

In the SAS log, a ruler that indicates column positions appears before the first record listed.

For variable-length records (RECFM=V), SAS writes the record length at the end of the input line. SAS does not write the length for fixed-length records (RECFM=F), unless the amount of data read does not equal the record length (LRECL).
Comparisons

The following table compares the LIST and PUT statements.

<table>
<thead>
<tr>
<th>Action</th>
<th>LIST Statement</th>
<th>PUT Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writes when</td>
<td>at the end of each iteration of the DATA step</td>
<td>immediately</td>
</tr>
<tr>
<td>Writes what</td>
<td>the input data records exactly as they appear</td>
<td>the variables or literals specified</td>
</tr>
<tr>
<td>Writes where</td>
<td>only to the SAS log</td>
<td>to the SAS log, the SAS output destination, or to any external file</td>
</tr>
<tr>
<td>Works with</td>
<td>INPUT statement only</td>
<td>any data-reading statement</td>
</tr>
<tr>
<td>Handles hexadecimal values</td>
<td>automatically prints a hexadecimal value if it encounters an unprintable character</td>
<td>represents characters in hexadecimal only when a hexadecimal format is given</td>
</tr>
</tbody>
</table>

Examples

**Example 1: Listing Records That Contain Missing Data**

This example uses the LIST statement to write to the SAS log any input records that contain missing data. Because of the #3 line pointer control in the INPUT statement, SAS reads three input records to create a single observation. Therefore, the LIST statement writes the three current input records to the SAS log each time a value for W2AMT is missing.

```sas
data employee;
  input ssn 1-9 #3 w2amt 1-6;
  if w2amt=. then list;
datalines;
23456789
JAMES SMITH
356.79
345671234
Jeffrey Thomas
.
;
```

**Output 2.17 Log Listing of Missing Data**

```
RULE:--------1--------2--------3--------4--------5--------
 9 345671234
10  Jeffrey Thomas
11 .
```

The numbers 9, 10, and 11 are line numbers in the SAS log.
Example 2: Listing the Record Length of Variable-Length Records
This example uses as input an external file that contains variable-length ID numbers. The RECFM=V option is specified in the INFILE statement, and the LIST statement writes the records to the SAS log. When the file has variable-length records, as indicated by the RECFM=V option in this example, SAS writes the record length at the end of each record that is listed in the SAS log.

```sas
data employee;
  infile 'your-external-file' recfm=v;
  input id $;
  list;
run;
```

Output 2.18  Log Listing of Variable-Length Records and Record Lengths

<table>
<thead>
<tr>
<th>RULE:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23456789</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>123456789</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5555555555</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>345671234</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2345678910</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2345678</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also

Statements:
- “PUT Statement” on page 335

%LIST Statement
Displays lines that are entered in the current session.

Valid in: Anywhere
Category: Program Control

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.

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 191

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: 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.

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 the 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.

Restriction  The SAS file must be created before you can request a lock. For information about locking a member of a SAS library when the member does not exist, see the SAS/SHARE User's Guide.

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 the 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 the 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 the 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 warning 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 errors, 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 not 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. The following 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**
The following 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;
data mydata.census;
  modify mydata.census;
  (statements to remove obsolete observations)
run;
proc sort force data=mydata.census;
  by CrimeRate;
run;
proc datasets library=mydata;
  modify census;
  index create CrimeRate;
quit;
lock mydata.census clear;

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 it 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 the “LOCK Statement” in SAS/SHARE User's Guide.

Data Set Options:

• “CNTLLEV= Data Set Option” in SAS Data Set Options: Reference

LOSTCARD Statement

Resynchronizes the input data when SAS encounters a missing or invalid record in data that has multiple records per observation.

Valid in: DATA step
Category: Action
Type: Executable
Syntax
LOSTCARD;

Without Arguments
The LOSTCARD statement prevents SAS from reading a record from the next group when the current group has a missing record.

Details

When to Use LOSTCARD
When SAS reads multiple records to create a single observation, it does not discover that a record is missing until it reaches the end of the data. If there is a missing record in your data, the values for subsequent observations in the SAS data set might be incorrect. Using LOSTCARD prevents SAS from reading a record from the next group when the current group has fewer records than SAS expected.

LOSTCARD is most useful when the input data have a fixed number of records per observation and when each record for an observation contains an identification variable that has the same value. LOSTCARD usually appears in conditional processing such as in the THEN clause of an IF-THEN statement, or in a statement in a SELECT group.

When LOSTCARD Executes
When LOSTCARD executes, SAS takes several steps:

1. Writes three items to the SAS log: a lost card message, a ruler, and all the records that it read in its attempt to build the current observation.
2. Discards the first record in the group of records being read, does not write an observation, and returns processing to the beginning of the DATA step.
3. Does not increment the automatic variable _N_ by 1. (Normally, SAS increments _N_ by 1 at the beginning of each DATA step iteration.)
4. Attempts to build an observation by beginning with the second record in the group, and reads the number of records that the INPUT statement specifies.
5. Repeats steps 1 through 4 when the IF condition for a lost card is still true. To make the log more readable, SAS prints the message and ruler only once for a given group of records. In addition, SAS prints each record only once, even if a record is used in successive attempts to build an observation.
6. Builds an observation and writes it to the SAS data set when the IF condition for a lost card is no longer true.

Example: Resynchronizing Input Data
This example uses the LOSTCARD statement in a conditional construct to identify missing data records and to resynchronize the input data:

```
data inspect;
  input id 1-3 age 8-9 #2 id2 1-3 loc
    #3 id3 1-3 wt;
  if id ne id2 or id ne id3 then do;
    put 'DATA RECORD ERROR: ' id= id2= id3=;
    lostcard;
  end;
```
The DATA step reads three input records before writing an observation. If the identification number in record 1 (variable ID) does not match the identification number in the second record (ID2) or third record (ID3), a record is incorrectly entered or omitted. The IF-THEN DO statement specifies that if an identification number is invalid, SAS prints the message that is specified in the PUT statement message and executes the LOSTCARD statement.

In this example, the third record for the second observation (ID3=400) is missing. The second record for the third observation is incorrectly entered (ID=400 while ID2=409). Therefore, the data set contains two observations with ID values 301 and 411. There are no observations for ID=302 or ID=400. The PUT and LOSTCARD statements write these statements to the SAS log when the DATA step executes:

```
DATA RECORD ERROR: id=302 id2=302 id3=400
NOTE: LOST CARD.
RULE:---------1---------2---------3---------4---------5---------
14 302  61
15 302  83171
16 400  46
DATA RECORD ERROR: id=302 id2=400 id3=409
NOTE: LOST CARD.
17 409  23145
DATA RECORD ERROR: id=400 id2=409 id3=400
NOTE: LOST CARD.
18 400  197
DATA RECORD ERROR: id=409 id2=400 id3=411
NOTE: LOST CARD.
19 411  53
DATA RECORD ERROR: id=400 id2=411 id3=411
NOTE: LOST CARD.
20 411  99551
```

The numbers 14, 15, 16, 17, 18, 19, and 20 are line numbers in the SAS log.

See Also

Statements:

- “IF-THEN/ELSE Statement” on page 189
MERGE Statement

Joins observations from two or more SAS data sets into a single observation.

Valid in: DATA step  
Category: File-Handling  
Type: Executable  
Note: The variables read using the MERGE statement are retained in the PDV. For more information, see “Overview of DATA Step Processing” in SAS Language Reference: Concepts and the “RETAIN Statement” on page 378.

Syntax

MERGE SAS-data-set-1 <(data-set-options)>  
SAS-data-set-2 <(data-set-options)>  
<...SAS-data-set-n<(data-set-options)> >  
<END=variable>;  

Arguments

SAS-data-set  
specifies at least two existing SAS data sets from which observations are read. You can specify individual data sets, data set lists, or a combination of both.

Tips  
Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

You can specify additional SAS data sets.

See  
“Using Data Set Lists with MERGE” on page 305

(data-set-options)  
specifies one or more SAS data set options in parentheses after a SAS data set name.

Note  
The data set options specify actions that SAS is to take when it reads observations into the DATA step for processing. For a list of data set options, see the SAS Data Set Options: Reference

Tip  
Data set options that apply to a data set list apply to all of the data sets in the list.

END=variable  
names and creates a temporary variable that contains an end-of-file indicator.

Note  
The variable, which is initialized to 0, is set to 1 when the MERGE statement processes the last observation. If the input data sets have different numbers of observations, the END= variable is set to 1 when MERGE processes the last observation from all data sets.

Tip  
The END= variable is not added to any SAS data set that is being created.
Details

Overview
The MERGE statement is flexible and has a variety of uses in SAS programming. This section describes basic uses of MERGE. Other applications include using more than one BY variable, merging more than two data sets, and merging a few observations with all observations in another data set.

For more information, see “How to Prepare Your Data Sets” in SAS Language Reference: Concepts.

Using Data Set Lists with MERGE
You can use data set lists with the MERGE statement. Data set lists provide a quick way to reference existing groups of data sets. These data set lists must be either name prefix lists or numbered range lists.

Name prefix lists refer to all data sets that begin with a specified character string. For example, `merge SALES1;;` tells SAS to merge all data sets starting with "SALES1" such as SALES1, SALES10, SALES11, and SALES12.

Numbered range lists require you to have a series of data sets with the same name, except for the last character or characters, which are consecutive numbers. In a numbered range list, you can begin with any number and end with any number. For example, these lists refer to the same data sets:

```
sales1 sales2 sales3 sales4
sales1-sales4
```

Note: If the numeric suffix of the first data set name contains leading zeros, the number of digits in the numeric suffix of the last data set name must be greater than or equal to the number of digits in the first data set name. Otherwise, an error will occur. For example, the data set lists `sales001–sales99` and `sales01–sales9` will cause an error. The data set list `sales001–sales999` is valid. If the numeric suffix of the first data set name does not contain leading zeros, the number of digits in the numeric suffix of the first and last data set names do not have to be equal. For example, the data set list `sales1–sales999` is valid.

Some other rules to consider when using numbered data set lists are as follows:

- You can specify groups of ranges.
  
  ```
  merge cost1-cost4 cost11-cost14 cost21-cost24;
  ```
  
- You can mix numbered range lists with name prefix lists.
  
  ```
  merge cost1-cost4 cost2: cost33-37;
  ```
  
- You can mix single data sets with data set lists.
  
  ```
  merge cost1 cost10-cost20 cost30;
  ```
  
- Quotation marks around data set lists are ignored.
  
  ```
  /* these two lines are the same */
  merge sales1-sales4;
  merge 'sales1'n-'sales4'n;
  ```
  
- Spaces in data set names are invalid. If quotation marks are used, trailing blanks are ignored.
  
  ```
  /* blanks in these statements will cause errors */
  merge sales 1-sales 4;
  merge 'sales 1'n - 'sales 4'n;
  ```
One-to-One Merging

One-to-one merging combines observations from two or more SAS data sets into a single observation in a new data set. To perform a one-to-one merge, use the MERGE statement without a BY statement. SAS combines the first observation from all data sets that are named in the MERGE statement into the first observation in the new data set, the second observation from all data sets into the second observation in the new data set, and so on. In a one-to-one merge, the number of observations in the new data set is equal to the number of observations in the largest data set named in the MERGE statement. See Example 1 for an example of a one-to-one merge. For more information, see “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts.

CAUTION:

Use care when you combine data sets with a one-to-one merge. One-to-one merges can sometimes produce undesirable results. Test your program on representative samples of the data sets before you use this method.

Match-Merging

Match-merging combines observations from two or more SAS data sets into a single observation in a new data set according to the values of a common variable. The number of observations in the new data set is the sum of the largest number of observations in each BY group in all data sets. To perform a match-merge, use a BY statement immediately after the MERGE statement. The variables in the BY statement must be common to all data sets. Only one BY statement can accompany each MERGE statement in a DATA step. The data sets that are listed in the MERGE statement must be sorted in order of the values of the variables that are listed in the BY statement, or they must have an appropriate index. See Example 2 for an example of a match-merge. For more information, see “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts.

Note: The MERGE statement does not produce a Cartesian product on a many-to-many match-merge. Instead, it performs a one-to-one merge while there are observations in the BY group in at least one data set. When all observations in the BY group have been read from one data set and there are still more observations in another data set, SAS performs a one-to-many merge until all observations have been read for the BY group.

Comparisons

- MERGE combines observations from two or more SAS data sets. UPDATE combines observations from exactly two SAS data sets. UPDATE changes or updates the values of selected observations in a master data set as well. UPDATE also might add observations.
Like UPDATE, MODIFY combines observations from two SAS data sets by changing or updating values of selected observations in a master data set.

The results that are obtained by reading observations using two or more SET statements are similar to the results that are obtained by using the MERGE statement with no BY statement. However, with the SET statements, SAS stops processing before all observations are read from all data sets if the number of observations are not equal. In contrast, SAS continues processing all observations in all data sets named in the MERGE statement.

Examples

Example 1: One-to-One Merging
This example shows how to combine observations from two data sets into a single observation in a new data set:

```sas
data benefits.qtr1;
  merge benefits.jan benefits.feb;
run;
```

Example 2: Match-Merging
This example shows how to combine observations from two data sets into a single observation in a new data set according to the values of a variable that is specified in the BY statement:

```sas
data inventory;
  merge stock orders;
  by partnum;
run;
```

Example 3: Merging with a Data Set List
This example uses a data list to define the data sets that are merged.

```sas
data d008; job=3; emp=19; run;
data d009; job=3; sal=50; run;
data d010; job=4; emp=97; run;
data d011; job=4; sal=15; run;
data comb;
  merge d008-d011;
  by job;
run;
proc print data=comb;
run;
```

See Also

- “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts

Statements:

- “BY Statement” on page 31
- “MODIFY Statement” on page 309
- “SET Statement” on page 394
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 contain 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” on page 422

System Options:
- “MISSING= System Option” in SAS System Options: Reference

MODIFY Statement
Replaces, deletes, and appends observations in an existing SAS data set in place but does not create an additional copy.

Valid in: DATA step
Category: File-Handling
Type: Executable
Restriction: Cannot modify the descriptor portion of a SAS data set, such as adding a variable
Notes: If you modify a password-protected data set, specify the password with the appropriate data set option (ALTER= or PW=) in the MODIFY statement, and not in the DATA statement.

The variables read using the MODIFY statement are retained in the PDV. For more information, see “Overview of DATA Step Processing” in SAS Language Reference: Concepts and the “RETAIN Statement” on page 378.

CAUTION: Damage to the SAS data set can occur if the system terminates abnormally during a DATA step that contains the MODIFY statement. Observations in native SAS data files might have incorrect data values, or the data file might become unreadable. DBMS tables that are referenced by views are not affected.

Syntax

Form 1: MODIFY master-data-set<(data-set-options)> transaction-data-set<(data-set-options)> <CUROBS=variable> <NOBS=variable> <END=variable> <UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;  
BY by-variable; 

Form 2: MODIFY master-data-set<(data-set-options)> KEY=index </ UNIQUE> <KEYRESET=variable> <NOBS=variable> <END=variable>; 

Form 3: MODIFY master-data-set<(data-set-options)> <NOBS=variable> POINT=variable; 

Form 4: MODIFY master-data-set<(data-set-options)> <NOBS=variable> <END=variable>; 

Arguments

master-data-set
specifies the SAS data set that you want to modify.
Restrictions

This data set must also appear in the DATA statement.

For sequential and matching access, the master data set can be a SAS data file, a SAS/ACCESS view, an SQL view, or a DBMS engine for the LIBNAME statement. It cannot be a DATA step view or a pass-through view.

For random access using POINT=, the master data set must be a SAS data file or an SQL view that references a SAS data file.

For direct access using KEY=, the master data set can be a SAS data file or the DBMS engine for the LIBNAME statement. If it is a SAS file, it must be indexed and the index name must be specified on the KEY= option.

For a DBMS, the KEY= is set to the keyword DBKEY and the column names to use as an index must be specified on the DBKEY= data set option. These column names are used in constructing a WHERE expression that is passed to the DBMS.

Tip

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

(data-set-options)

specifies one or more SAS data set options in parentheses after a SAS data set name.

Note

The data set options specify actions that SAS is to take when it reads observations into the DATA step for processing. For a list of data set options, see the SAS Data Set Options: Reference

Tip

Data set options that apply to a data set list apply to all of the data sets in the list.

transaction-data-set

specifies the SAS data set that provides the values for matching access. These values are the values that you want to use to update the master data set.

Restriction

Specify this data set only when the DATA step contains a BY statement.

Tip

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

by-variable

specifies one or more variables by which you identify corresponding observations.

CUROBS=variable

creates and names a variable that contains the observation number that was just read from the data set.

END=variable

creates and names a temporary variable that contains an end-of-file indicator.

Restriction

Do not use this argument in the same MODIFY statement with the POINT= argument. POINT= indicates that MODIFY uses random
The value of the END= variable is never set to 1 for random access.

### Notes

The variable, which is initialized to zero, is set to 1 when the MODIFY statement reads the last observation of the data set being modified (for sequential access) or the last observation of the transaction data set (for matching access). It is also set to 1 when MODIFY cannot find a match for a KEY= value (random access).

This variable is not added to any data set.

### KEY= index

specifies a simple or composite index of the SAS data file that is being modified. The KEY= argument retrieves observations from that SAS data file based on index values that are supplied by like-named variables in another source of information.

**Default**

If the KEY= value is not found, the automatic variable _ERROR_ is set to 1, and the automatic variable _IORC_ receives the value corresponding to the SYSRC autocall macro's mnemonic _DSENOM_. See “Automatic Variable _IORC_ and the SYSRC Autocall Macro” on page 315.

**Restriction**

KEY= processing is different for SAS/ACCESS engines. See the SAS/ACCESS documentation for more information.

**Tips**

Use the KEYRESET= option to control whether a KEY= search should begin at the top of the index for the data set that is being read.

Examples of sources for index values include a separate SAS data set named in a SET statement and an external file that is read by an INPUT statement.

If duplicates exist in the master file, only the first occurrence is updated unless you use a DO-LOOP to execute a SET statement for the data set that is listed on the KEY= option for all duplicates in the master data set.

If duplicates exist in the transaction data set, and they are consecutive, use the UNIQUE option to force the search for a match in the master data set to begin at the top of the index. Write an accumulation statement to add each duplicate transaction to the observation in master. Without the UNIQUE option, only the first duplicate transaction observation updates the master.

If the duplicates in the transaction data set are not consecutive, the search begins at the beginning of the index each time, so that each duplicate is applied to the master. Write an accumulation statement to add each duplicate to the master.

**See**

“KEYRESET= variable” on page 312

UNIQUE on page 313

**Examples**

“Example 5: Modifying Observations Located by an Index” on page 323

“Example 6: Handling Duplicate Index Values” on page 324
KEYRESET=variable
controls whether a KEY= search should begin at the top of the index for the data set that is being read. When the value of the KEYRESET variable is 1, the index lookup begins at the top of the index. When the value of the KEYRESET variable is 0, the index lookup is not reset and the lookup continues where the prior lookup ended.

Interaction The KEYRESET= option is similar to the UNIQUE option, except the KEYRESET= option enables you to determine when the KEY= search should begin at the top of the index again.

See “KEY= index” on page 311
“ UNIQUE” on page 313

NOBS=variable
creates and names a temporary variable whose value is usually the total number of observations in the input data set. For certain SAS views and sequential engines such as the TAPE and XML engines, SAS cannot determine the number of observations. In these cases, SAS sets the value of the NOBS= variable to the largest positive integer value available in the operating environment.

Note At compilation time, SAS reads the descriptor portion of the data set and assigns the value of the NOBS= variable automatically. Thus, you can refer to the NOBS= variable before the MODIFY statement. The variable is available in the DATA step but is not added to the new data set.

Tip The NOBS= and POINT= options are independent of each other.

Example “Example 4: Modifying Observations Located by Observation Number” on page 322

POINT=variable
reads SAS data sets using random (direct) access by observation number. variable names a variable whose value is the number of the observation to read. The POINT= variable is available anywhere in the DATA step, but it is not added to any SAS data set.

Restrictions You cannot use the POINT= option with any of the following:
• BY statement
• WHERE statement
• WHERE= data set option
• transport format data sets
• sequential data sets (on tape or disk)
• a table from another vendor's relational database management system.

You can use POINT= with compressed data sets only if the data set was created with the POINTOBS= data set option set to YES, the default value.

You can use the random access method on compressed files only with SAS version 7 and beyond.
Requirements

When using the POINT= argument, include one or both of the following programming constructs:

- a STOP statement
- programming logic that checks for an invalid value of the POINT= variable

Because POINT= reads only the specified observations, SAS cannot detect an end-of-file condition as it would if the file were being read sequentially. Because detecting an end-of-file condition terminates a DATA step automatically, failure to substitute another means of terminating the DATA step when you use POINT= can cause the DATA step to go into a continuous loop.

Tip

If the POINT= value does not match an observation number, SAS sets the automatic variable _ERROR_ to 1.

Example

“Example 4: Modifying Observations Located by Observation Number” on page 322

**UNIQUE**

causes a KEY= search always to begin at the top of the index for the data file being modified.

Restriction

UNIQUE can appear only with the KEY= option.

Tip

Use UNIQUE when there are consecutive duplicate KEY= values in the transaction data set, so that the search for a match in the master data set begins at the top of the index file for each duplicate transaction. You must include an accumulation statement or the duplicate values overwrite each other causing only the last transaction value to be the result in the master observation.

See

“KEYRESET=variable” on page 312

Example

“Example 6: Handling Duplicate Index Values” on page 324

**UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK**

specifies whether missing variable values in a transaction data set are to be allowed to replace existing variable values in a master data set.

**MISSINGCHECK**

prevents missing variable values in a transaction data set from replacing values in a master data set.

**NOMISSINGCHECK**

allows missing variable values in a transaction data set to replace values in a master data set by preventing the check from being performed.

Default

MISSINGCHECK

Requirement

The UPDATEMODE argument must be accompanied by a BY statement that specifies the variables by which observations are matched.

Tip

However, special missing values are the exception and they replace values in the master data set even when MISSINGCHECK is in effect.
Details

**Matching Access (Form 1)**
The matching access method uses the BY statement to match observations from the transaction data set with observations in the master data set. The BY statement specifies a variable that is in the transaction data set and the master data set.

When the MODIFY statement reads an observation from the transaction data set, it uses dynamic WHERE processing to locate the matching observation in the master data set. The observation in the master data set can be either

- replaced in the master data set with the value from the transaction data set
- deleted from the master data set
- appended to the master data set.

“Example 3: Modifying Observations Using a Transaction Data Set” on page 320 shows the matching access method.

**Duplicate BY Values (Form 1)**
Duplicates in the master and transaction data sets affect processing.

- If duplicates exist in the master data set, only the first occurrence is updated because the generated WHERE statement always finds the first occurrence in the master.
- If duplicates exist in the transaction data set, the duplicates are applied one on top of another unless you write an accumulation statement to add all of them to the master observation. Without the accumulation statement, the values in the duplicates overwrite each other so that only the value in the last transaction is the result in the master observation.

**Direct Access by Indexed Values (Form 2)**
This method requires that you use the KEY= option in the MODIFY statement to name an indexed variable from the data set that is being modified. Use another data source (typically a SAS data set named in a SET statement or an external file read by an INPUT statement) to provide a like-named variable whose values are supplied to the index. MODIFY uses the index to locate observations in the data set that is being modified.

“Example 5: Modifying Observations Located by an Index” on page 323 shows the direct-access-by-indexed-values method.

**Duplicate Index Values (Form 2)**

- If there are duplicate values of the indexed variable in the master data set, only the first occurrence is retrieved, modified, or replaced. Use a DO LOOP to execute a SET statement with the KEY= option multiple times to update all duplicates with the transaction value.
- If there are duplicate, nonconsecutive values in the like-named variable in the data source, MODIFY applies each transaction cumulatively to the first observation in the master data set whose index value matches the values from the data source. Therefore, only the value in the last duplicate transaction is the result in the master observation unless you write an accumulation statement to accumulate each duplicate transaction value in the master observation.
- If there are duplicate, consecutive values in the variable in the data source, the values from the first observation in the data source are applied to the master data set, but the DATA step terminates with an error when it tries to locate an observation in the master data set for the second duplicate from the data source. To avoid this error, use
the UNIQUE option in the MODIFY statement. The UNIQUE option causes SAS to return to the top of the master data set before retrieving a match for the index value. You must write an accumulation statement to accumulate the values from all the duplicates. If you do not, only the last one applied is the result in the master observation.

“Example 6: Handling Duplicate Index Values” on page 324 shows how to handle duplicate index values.

- If there are duplicate index values in both data sets, you can use SQL to apply the duplicates in the transaction data set to the duplicates in the master data set in a one-to-one correspondence.

**Direct (Random) Access by Observation Number (Form 3)**

You can use the POINT= option in the MODIFY statement to name a variable from another data source (not the master data set), whose value is the number of an observation that you want to modify in the master data set. MODIFY uses the values of the POINT= variable to retrieve observations in the data set that you are modifying.

You can use POINT= on a compressed data set only if the data set was created with the POINTOBS= data set option.

It is good programming practice to validate the value of the POINT= variable and to check the status of the automatic variable _ERROR_.

“Example 4: Modifying Observations Located by Observation Number” on page 322 shows the direct (random) access by observation number method.

**CAUTION:**

POINT= can result in infinite looping. Be careful when you use POINT=, as failure to terminate the DATA step can cause the DATA step to go into a continuous loop. Use a STOP statement, programming logic that checks for an invalid value of the POINT= variable, or both.

**Sequential Access (Form 4)**

The sequential access method is the simplest form of the MODIFY statement, but it provides less control than the direct access methods. With the sequential access method, you can use the NOBS= and END= options to modify a data set; you do not use the POINT= or KEY= options.

**Preparing Your Data Sets Before Using MODIFY**

There are a number of things that you can do to improve performance and get the results that you want when using the MODIFY statement. For more information, see “Combining SAS Data Sets: Basic Concepts” in SAS Language Reference: Concepts.

**Automatic Variable _IORC_ and the SYSRC Autocall Macro**

The automatic variable _IORC_ contains the return code for each I/O operation that the MODIFY statement attempts to perform. The best way to test for values of _IORC_ is with the mnemonic codes that are provided by the SYSRC autocall macro. Each mnemonic code describes one condition. The mnemonics provide an easy method for testing problems in a DATA step program. These codes are useful:

_-DSENMR_

specifies that the transaction data set observation does not exist on the master data set (used only with MODIFY and BY statements). If consecutive observations with different BY values do not find a match in the master data set, both of them return _DSENMR_.

-
_DSEMTR

specifies that multiple transaction data set observations with a given BY value do not exist on the master data set (used only with MODIFY and BY statements). If consecutive observations with the same BY values do not find a match in the master data set, the first observation returns _DSENMR and the subsequent observations return _DSEMTR.

_DSENOM

specifies that the data set being modified does not contain the observation that is requested by the KEY= option or the POINT= option.

_SENOCHN

specifies that SAS is attempting to execute an OUTPUT or REPLACE statement on an observation that contains a key value which duplicates one already existing on an indexed data set that requires unique key values.

_SOK

specifies that the observation was located.

Note: The IORCMSG function returns a formatted error message associated with the current value of _IORC_.

“Example 7: Controlling I/O” on page 326 shows how to use the automatic variable _IORC_ and the SYSRC autocall macro.

Writing Observations When MODIFY Is Used in a DATA Step

The way SAS writes observations to a SAS data set when the DATA step contains a MODIFY statement depends on whether certain other statements are present. The possibilities are

no explicit statement

writes the current observation to its original place in the SAS data set. The action occurs as the last action in the step (as if a REPLACE statement were the last statement in the step).

OUTPUT statement

if no data set is specified in the OUTPUT statement, writes the current observation to the end of all data sets that are specified in the DATA step. If a data set is specified, the statement writes the current observation to the end of the data set that is indicated. The action occurs at the point in the DATA step where the OUTPUT statement appears.

REPLACE <data-set-name> statement

rewrites the current observation in the specified data set or data sets, or, if no argument is specified, rewrites the current observation in each data set specified in the DATA statement. The action occurs at the point of the REPLACE statement.

REMOVE <data-set-name> statement

deletes the current observation in the specified data set or data sets, or, if no argument is specified, deletes the current observation in each data set specified in the DATA statement. The deletion can be a physical one or a logical one, depending on the characteristics of the engine that maintains the data set.

Remember the following as you work with these statements:

• When no OUTPUT, REPLACE, or REMOVE statement is specified, the default action is REPLACE.

• The OUTPUT, REPLACE, and REMOVE statements are independent of each other. You can code multiple OUTPUT, REPLACE, and REMOVE statements to apply to one observation. However, once an OUTPUT, REPLACE, or REMOVE statement...
executes, the MODIFY statement must execute again before the next REPLACE or REMOVE statement executes.

You can use OUTPUT and REPLACE in the following example of conditional logic because only one of the REPLACE or OUTPUT statements executes per observation:

```sas
data master;
    modify master trans; by key;
    if _iorc_ = 0 then replace;
    else
        output;
run;
```

But you should not use multiple REPLACE operations on the same observation as in this example:

```sas
data master;
    modify master;
    x=1;
    replace;
    replace;
run;
```

You can code multiple OUTPUT statements per observation. However, be careful when you use multiple OUTPUT statements. It is possible to go into an infinite loop with just one OUTPUT statement.

```sas
data master;
    modify master;
    output;
run;
```

- Using OUTPUT, REPLACE, or REMOVE in a DATA step overrides the default replacement of observations. If you use any one of these statements in a DATA step, you must explicitly program each action that you want to take.

- If both an OUTPUT statement and a REPLACE or REMOVE statement execute on a given observation, perform the OUTPUT action last to keep the position of the observation pointer correct.

“Example 8: Replacing and Removing Observations and Writing Observations to Different SAS Data Sets” on page 328 shows how to use the OUTPUT, REMOVE, and REPLACE statements to write observations.

**Missing Values and the MODIFY Statement**

By default, the UPDATEMODE=MISSINGCHECK option is in effect, so missing values in the transaction data set do not replace existing values in the master data set. Therefore, if you want to update some but not all variables and if the variables that you want to update differ from one observation to the next, set to missing those variables that are not changing. If you want missing values in the transaction data set to replace existing values in the master data set, use UPDATEMODE=NOMISSINGCHECK.

Even when UPDATEMODE=MISSINGCHECK is in effect, you can replace existing values with missing values by using special missing value characters in the transaction data set. To create the transaction data set, use the MISSING statement in the DATA step. If you define one of the special missing values \( \text{A} \) through \( \text{Z} \) for the transaction data set, SAS updates numeric variables in the master data set to that value.

If you want the resulting value in the master data set to be a regular missing value, use a single underscore (\(_\)) to represent missing values in the transaction data set. The
resulting value in the master data set will be a period (.) for missing numeric values and a blank for missing character values.

For more information about defining and using special missing value characters, see the “MISSING Statement” on page 308.

**Using MODIFY with Data Set Options**

If you use data set options (such as \texttt{KEEP=}) in your program, then use the options in the MODIFY statement for the master data set. Using data set options in the DATA statement might produce unexpected results.

**Using MODIFY in a SAS/SHARE Environment**

In a SAS/SHARE environment, the MODIFY statement accesses an observation in Update mode. That is, the observation is locked from the time MODIFY reads it until a REPLACE or REMOVE statement executes. At that point the observation is unlocked. It cannot be accessed until it is re-read with the MODIFY statement. The MODIFY statement opens the data set in Update mode, but the control level is based on the statement used. For example, \texttt{KEY=} and \texttt{POINT=} are member-level locking. See the \textit{SAS/SHARE User's Guide} for more information.

**Comparisons**

- When you use a MERGE, SET, or UPDATE statement in a DATA step, SAS creates a new SAS data set. The data set descriptor of the new copy can be different from the old one (variables added or deleted, labels changed, and so on). When you use a MODIFY statement in a DATA step, however, SAS does not create a new copy of the data set. As a result, the data set descriptor cannot change.

  For information about DBMS replacement rules, see the SAS/ACCESS documentation.

- If you use a BY statement with a MODIFY statement, MODIFY works much like the UPDATE statement, except that
  - neither the master data set nor the transaction data set needs to be sorted or indexed. (The BY statement that is used with MODIFY triggers dynamic WHERE processing.)

    \textit{Note:} Dynamic WHERE processing can be costly if the MODIFY statement modifies a SAS data set that is not in sorted order or has not been indexed. Having the master data set in sorted order or indexed and having the transaction data set in sorted order reduces processing overhead, especially for large files.

    - both the master data set and the transaction data set can have observations with duplicate values of the BY variables. MODIFY treats the duplicates as described in “Duplicate BY Values (Form 1)” on page 314.

    - MODIFY cannot make any changes to the descriptor information of the data set as UPDATE can. Thus, it cannot add or delete variables, change variable labels, and so on.

**Examples**

**Example 1: Input Data Set for Examples**

The examples modify the INVTY.STOCK data set. INVTY.STOCK contains these variables:
PARTNO
is a character variable with a unique value identifying each tool number.

DESC
is a character variable with the text description of each tool.

INSTOCK
is a numeric variable with a value describing how many units of each tool the company has in stock.

RECDATE
is a numeric variable containing the SAS date value that is the day for which INSTOCK values are current.

PRICE
is a numeric variable with a value that describes the unit price for each tool.

In addition, INVTY.STOCK contains a simple index on PARTNO. This DATA step creates INVTY.STOCK:

libname invty 'SAS-library';
data invty.stock(index=(partno));
  input PARTNO $ DESC $ INSTOCK @17
    RECDATE date7. @25 PRICE;
  format  recdate date7.;
datalines;
  K89R seal   34  27jul95 245.00
  M4J7 sander 98  20jun95 45.88
  LK43 filter 121 19may96 10.99
  MN21 brace 43   10aug96 27.87
  BC85 clamp 80   16aug96 9.55
  NCF3 valve 198 20mar96 24.50
  KJ66 cutter 6   18jun96 19.77
  UYN7 rod  211   09sep96 11.55
  JD03 switch 383 09jan97 13.99
  BV1E timer 26   03jan97 34.50
;

Example 2: Modifying All Observations
This example replaces the date on all of the records in the data set INVTY.STOCK with the current date. It also replaces the value of the variable RECDATE with the current date for all observations in INVTY.STOCK:

data invty.stock;
  modify invty.stock;
    recdate=today();
run;
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
Output 2.19  Results of Updating the RECDATE Field

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>DESC</th>
<th>INSTOCK</th>
<th>RECDATE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>K89R</td>
<td>seal</td>
<td>34</td>
<td>08DEC10</td>
<td>245.00</td>
</tr>
<tr>
<td>M4J7</td>
<td>sander</td>
<td>98</td>
<td>08DEC10</td>
<td>45.88</td>
</tr>
<tr>
<td>LK43</td>
<td>filter</td>
<td>121</td>
<td>08DEC10</td>
<td>10.99</td>
</tr>
<tr>
<td>MN21</td>
<td>brace</td>
<td>43</td>
<td>08DEC10</td>
<td>27.87</td>
</tr>
<tr>
<td>BC85</td>
<td>clamp</td>
<td>80</td>
<td>08DEC10</td>
<td>9.55</td>
</tr>
<tr>
<td>NCF3</td>
<td>valve</td>
<td>198</td>
<td>08DEC10</td>
<td>24.50</td>
</tr>
<tr>
<td>KJ56</td>
<td>cutter</td>
<td>6</td>
<td>08DEC10</td>
<td>19.77</td>
</tr>
<tr>
<td>UYN7</td>
<td>rod</td>
<td>211</td>
<td>08DEC10</td>
<td>11.55</td>
</tr>
<tr>
<td>JD03</td>
<td>switch</td>
<td>363</td>
<td>08DEC10</td>
<td>13.99</td>
</tr>
<tr>
<td>BV1E</td>
<td>timer</td>
<td>26</td>
<td>08DEC10</td>
<td>34.50</td>
</tr>
</tbody>
</table>

The MODIFY statement opens INVTY.STOCK for update processing. SAS reads one observation of INVTY.STOCK for each iteration of the DATA step and performs any operations that the code specifies. In this case, the code replaces the value of RECDATE with the result of the TODAY function for every iteration of the DATA step. An implicit REPLACE statement at the end of the step writes each observation to its previous location in INVTY.STOCK.

Example 3: Modifying Observations Using a Transaction Data Set

This example adds the quantity of newly received stock to its data set INVTY.STOCK as well as updating the date on which stock was received. The transaction data set ADDINV in the Work library contains the new data.

The ADDINV data set is the data set that contains the updated information. ADDINV contains these variables:

PARTNO
    is a character variable that corresponds to the indexed variable PARTNO in INVTY.STOCK.

NWSTOCK
    is a numeric variable that represents quantities of newly received stock for each tool.

ADDINV is the second data set in the MODIFY statement. SAS uses it as the transaction data set and reads each observation from ADDINV sequentially. Because the BY statement specifies the common variable PARTNO, MODIFY finds the first occurrence of the value of PARTNO in INVTY.STOCK that matches the value of PARTNO in ADDINV. For each observation with a matching value, the DATA step changes the value of RECDATE to today's date and replaces the value of INSTOCK with the sum of INSTOCK and NWSTOCK (from ADDINV). MODIFY does not add NWSTOCK to the INVTY.STOCK data set because that would modify the data set descriptor. Thus, it is not necessary to put NWSTOCK in a DROP statement.
This example specifies ADDINV as the transaction data set that contains information to modify INVTY.STOCK. A BY statement specifies the shared variable whose values locate the observations in INVTY.STOCK.

This DATA step creates ADDINV:

```sas
data addinv;
  input PARTNO $ NWSTOCK;
datalines;
K89R 55
M4J7 21
LK43 43
MN21 73
BC85 57
NCF3 90
KJ66 2
UYN7 108
JD03 55
BV1E 27
;
```

This DATA step uses values from ADDINV to update INVTY.STOCK.

```sas
libname invty 'SAS-library';
data invty.stock;
  modify invty.stock addinv;
  by partno;
  RECDATE=today();
  INSTOCK=instock+nwstock;
  if _iorc_=0 then replace;
run;
```

```sas
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```
Output 2.20  Results of Updating the IN STOCK and RECE DATE Fields

Example 4: Modifying Observations Located by Observation Number
This example reads the data set NEWP, determines which observation number in IN VTY. STOCK to update based on the value of TOOL_OBS, and performs the update. This example explicitly specifies the update activity by using an assignment statement to replace the value of PRICE with the value of NEWP.

The data set NEWP contains two variables:

TOOL_OBS
contains the observation number of each tool in the tool company's master data set, IN VTY. STOCK.

NEWP
contains the new price for each tool.

This DATA step creates NEWP:

data newp;
  input TOOL_OBS NEWP;
datalines;
  1 251.00
  2 49.33
  3 12.32
  4 30.00
  5 15.00
  6 25.75
  7 22.00
  8 14.00
  9 14.32
 10 35.00
;
This DATA step updates INVTY.STOCK:

```
libname invty 'SAS-library';
data invty.stock;
  set newp;
  modify invty.stock point=tool_obs
     nobs=max_obs;
  if _error_=1 then
    do;
      put 'ERROR occurred for TOOL_OBS=' tool_obs /
         'during DATA step iteration' _n_ /
         'TOOL_OBS value might be out of range.';
      _error_=0;
      stop;
    end;
  PRICE=newp;
  RECDATE=today();
run;
```

```
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

**Output 2.21  Results of Updating the RECDATE and PRICE Fields**

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>DESC</th>
<th>INSTOCK</th>
<th>RECDATE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>K89R</td>
<td>seal</td>
<td>34</td>
<td>08DEC10</td>
<td>251.00</td>
</tr>
<tr>
<td>M4J7</td>
<td>sander</td>
<td>98</td>
<td>08DEC10</td>
<td>49.33</td>
</tr>
<tr>
<td>LK43</td>
<td>filter</td>
<td>121</td>
<td>08DEC10</td>
<td>12.32</td>
</tr>
<tr>
<td>MN21</td>
<td>brace</td>
<td>43</td>
<td>08DEC10</td>
<td>30.00</td>
</tr>
<tr>
<td>BC85</td>
<td>clamp</td>
<td>80</td>
<td>08DEC10</td>
<td>15.00</td>
</tr>
<tr>
<td>NCF3</td>
<td>valve</td>
<td>198</td>
<td>08DEC10</td>
<td>25.75</td>
</tr>
<tr>
<td>KJ56</td>
<td>cutter</td>
<td>6</td>
<td>08DEC10</td>
<td>22.00</td>
</tr>
<tr>
<td>UYN7</td>
<td>rod</td>
<td>211</td>
<td>08DEC10</td>
<td>14.00</td>
</tr>
<tr>
<td>JD03</td>
<td>switch</td>
<td>383</td>
<td>08DEC10</td>
<td>14.32</td>
</tr>
<tr>
<td>BV1E</td>
<td>timer</td>
<td>26</td>
<td>08DEC10</td>
<td>35.00</td>
</tr>
</tbody>
</table>

**Example 5: Modifying Observations Located by an Index**

This example uses the KEY= option to identify observations to retrieve by matching the values of PARTNO from ADDINV with the indexed values of PARTNO in INVTY.STOCK. ADDINV is created in “Example 3: Modifying Observations Using a Transaction Data Set” on page 320.

KEY= supplies index values that allow MODIFY to access directly the observations to update. No dynamic WHERE processing occurs. In this example, you specify that the
value of INSTOCK in the master data set INVTY.STOCK increases by the value of the variable NWSTOCK from the transaction data set ADDINV.

```
libname invty 'SAS-library';
data invty.stock;
  set addinv;
  modify invty.stock key=partno;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
run;
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

Output 2.22  Results of Updating the INSTOCK and RECDATE Fields By Using an Index

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>DESC</th>
<th>INSTOCK</th>
<th>RECDATE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>K89R</td>
<td>seal</td>
<td>89</td>
<td>08DEC10</td>
<td>245.00</td>
</tr>
<tr>
<td>M4J7</td>
<td>sander</td>
<td>119</td>
<td>08DEC10</td>
<td>45.88</td>
</tr>
<tr>
<td>LK43</td>
<td>filter</td>
<td>164</td>
<td>08DEC10</td>
<td>10.99</td>
</tr>
<tr>
<td>MN21</td>
<td>brace</td>
<td>116</td>
<td>08DEC10</td>
<td>27.87</td>
</tr>
<tr>
<td>BC85</td>
<td>clamp</td>
<td>137</td>
<td>08DEC10</td>
<td>9.55</td>
</tr>
<tr>
<td>NCF3</td>
<td>valve</td>
<td>288</td>
<td>08DEC10</td>
<td>24.50</td>
</tr>
<tr>
<td>KJ56</td>
<td>cutter</td>
<td>8</td>
<td>08DEC10</td>
<td>19.77</td>
</tr>
<tr>
<td>UYN7</td>
<td>rod</td>
<td>319</td>
<td>08DEC10</td>
<td>11.55</td>
</tr>
<tr>
<td>JD03</td>
<td>switch</td>
<td>438</td>
<td>08DEC10</td>
<td>13.99</td>
</tr>
<tr>
<td>BV1E</td>
<td>timer</td>
<td>53</td>
<td>08DEC10</td>
<td>34.50</td>
</tr>
</tbody>
</table>

Example 6: Handling Duplicate Index Values

This example shows how MODIFY handles duplicate values of the variable in the SET data set that is supplying values to the index on the master data set.

The NEWINV data set is the data set that contains the updated information. NEWINV contains these variables:

PARTNO
is a character variable that corresponds to the indexed variable PARTNO in INVTY.STOCK. The NEWINV data set contains duplicate values for PARTNO; M4J7 appears twice.

NWSTOCK
is a numeric variable that represents quantities of newly received stock for each tool.

This DATA step creates NEWINV:
data newinv;
   input PARTNO $ NWSTOCK;
datalines;
  K89R 55
  M4J7 21
  M4J7 26
  LK43 43
  MN21 73
  BC85 57
  NCF3 90
  KJ66 2
  UYN7 108
  JD03 55
  BV1E 27
;

This DATA step terminates with an error when it tries to locate an observation in INVTY.STOCK to match with the second occurrence of M4J7 in NEWINV:

libname invty 'SAS-library';
   /* This DATA step terminates with an error! */
data invty.stock;
   set newinv;
   modify invty.stock key=partno;
      INSTOCK=instock+nwstock;
      RECDATE=today();
run;

This message appears in the SAS log:

ERROR: No matching observation was found in MASTER data set.  
PARTNO=M4J7 NWSTOCK=26 DESC=sander INSTOCK=166 RECDATE=08DEC10 PRICE=45.88  
_ERROR_=1  
_IORC_=1230015 _N_=3  
NOTE: The SAS System stopped processing this step because of errors.  
NOTE: There were 3 observations read from the data set WORK.NEWINV.  
NOTE: The data set INVTY.STOCK has been updated.  There were 2 observations  
rewritten, 0  
   observations added and 0 observations deleted.

Adding the UNIQUE option to the MODIFY statement avoids the error in the previous DATA step. The UNIQUE option causes the DATA step to return to the top of the index each time it looks for a match for the value from the SET data set. Thus, it finds the M4J7 in the MASTER data set for each occurrence of M4J7 in the SET data set. The updated result for M4J7 in the output shows that both values of NWSTOCK from NEWINV for M4J7 are added to the value of INSTOCK for M4J7 in INVTY.STOCK. An accumulation statement sums the values; without it, only the value of the last instance of M4J7 would be the result in INVTY.STOCK.

data invty.stock;
   set newinv;
   modify invty.stock key=partno / unique;
      INSTOCK=instock+nwstock;
      RECDATE=today();
      if _iorc_=0 then replace;
run;
proc print data=invty.stock noobs;
   title 'Results of Using the UNIQUE Option';
run;

Output 2.23  Results of Updating the INSTOCK and RECDATE Fields By Using the UNIQUE Option

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>DESC</th>
<th>INSTOCK</th>
<th>RECDATE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>K89R</td>
<td>seal</td>
<td>89</td>
<td>08DEC10</td>
<td>245.00</td>
</tr>
<tr>
<td>M4J7</td>
<td>sander</td>
<td>145</td>
<td>08DEC10</td>
<td>45.88</td>
</tr>
<tr>
<td>LK43</td>
<td>filter</td>
<td>164</td>
<td>08DEC10</td>
<td>10.99</td>
</tr>
<tr>
<td>MN21</td>
<td>brace</td>
<td>116</td>
<td>08DEC10</td>
<td>27.87</td>
</tr>
<tr>
<td>BC85</td>
<td>clamp</td>
<td>137</td>
<td>08DEC10</td>
<td>9.55</td>
</tr>
<tr>
<td>NCF3</td>
<td>valve</td>
<td>288</td>
<td>08DEC10</td>
<td>24.50</td>
</tr>
<tr>
<td>KJ66</td>
<td>cutter</td>
<td>8</td>
<td>08DEC10</td>
<td>19.77</td>
</tr>
<tr>
<td>UYN7</td>
<td>rod</td>
<td>319</td>
<td>08DEC10</td>
<td>11.55</td>
</tr>
<tr>
<td>JD03</td>
<td>switch</td>
<td>438</td>
<td>08DEC10</td>
<td>13.99</td>
</tr>
<tr>
<td>BV1E</td>
<td>timer</td>
<td>53</td>
<td>08DEC10</td>
<td>34.50</td>
</tr>
</tbody>
</table>

Example 7: Controlling I/O

This example uses the SYSRC autocall macro and the _IORC_ automatic variable to control I/O condition. This technique helps prevent unexpected results that could go undetected. This example uses the direct access method with an index to update INVTY.STOCK. The data in the NEWSHIP data set updates INVTY.STOCK.

This DATA step creates NEWSHIP:

data newship;
  input PARTNO $ DESC $ NWSTOCK @17 SHPDTE date7. @25 NWPRICE;
datalines;
  K39R  seal 14   14nov96 245.00
  M4J7  sander 24  23aug96 47.98
  LK43  filter11 29jan97 14.99
  MN21  brace 9   09jan97 27.87
  BC85  clamp 12  09dec96 10.00
  ME34  cutter 8  14nov96 14.50
;

Each WHEN clause in the SELECT statement specifies actions for each input/output return code that is returned by the SYSRC autocall macro:

- _SOK_ indicates that the MODIFY statement executed successfully.
- _DSENOM_ indicates that no matching observation was found in INVTY.STOCK. The OUTPUT statement specifies that the observation be appended to INVTY.STOCK. See the last observation in the output.
• If any other code is returned by SYSRC, the DATA step terminates and the PUT statement writes the message to the log.

```sas
libname invty 'SAS-library';
data invty.stock;
  set newship;
  modify invty.stock key=partno;
  select (_iorc_);
  when (%sysrc(_sok)) do;
    INSTOCK=instock+ nwstock;
    RECDATE=shpdate;
    PRICE=nwprice;
    replace;
  end;
  when (%sysrc(_dsenom)) do;
    INSTOCK=nwstock;
    RECDATE=shpdate;
    PRICE=nwprice;
    output;
    _error_=0;
  end;
  otherwise do;
    put
    'An unexpected I/O error has occurred.'/
    'Check your data and your program';
    _error_=0;
    stop;
  end;
end;
run;
```

```sas
proc print data=invty.stock noobs;
  title 'INVTY.STOCK Data Set';
run;
```
Example 8: Replacing and Removing Observations and Writing Observations to Different SAS Data Sets

This example shows that you can replace and remove (delete) observations and write observations to different data sets. Further, this example shows that if an OUTPUT, REPLACE, or REMOVE statement is present, you must specify explicitly what action to take because no default statement is generated.

The parts that were received in 1997 are output to INVTY.STOCK97 and are removed from INVTY.STOCK. Likewise, the parts that were received in 1995 are output to INVTY.STOCK95 and are removed from INVTY.STOCK. Only the parts that were received in 1996 remain in INVTY.STOCK, and the PRICE is updated only in INVTY.STOCK.

```
libname invty 'SAS-library';

data invty.stock invty.stock95 invty.stock97;
modify invty.stock;
if recdate>'01jan97'd then do;
  output invty.stock97;
  remove invty.stock;
end;
else if recdate<'01jan96'd then do;
  output invty.stock95;
  remove invty.stock;
end;
else do;
  price=price*1.1;
  replace invty.stock;
end;
```
run;

proc print data=invty.stock noobs;
    title 'New Prices for Stock Received in 1996';
run;

Output 2.25  Output from Writing Observations to a Specific SAS Data Set

<table>
<thead>
<tr>
<th>PARTNO</th>
<th>DESC</th>
<th>INSTOCK</th>
<th>RECDATE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK43</td>
<td>filter</td>
<td>121</td>
<td>19MAY96</td>
<td>12.089</td>
</tr>
<tr>
<td>MN21</td>
<td>brace</td>
<td>43</td>
<td>10AUG96</td>
<td>30.657</td>
</tr>
<tr>
<td>BC85</td>
<td>clamp</td>
<td>80</td>
<td>16AUG96</td>
<td>10.505</td>
</tr>
<tr>
<td>NCF3</td>
<td>valve</td>
<td>198</td>
<td>20MAR96</td>
<td>26.950</td>
</tr>
<tr>
<td>KJ66</td>
<td>cutter</td>
<td>6</td>
<td>18JUN96</td>
<td>21.747</td>
</tr>
<tr>
<td>UYN7</td>
<td>rod</td>
<td>211</td>
<td>09SEP96</td>
<td>12.705</td>
</tr>
</tbody>
</table>

See Also

- “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts
- SAS SQL Procedure User's Guide

Statements:

- “MISSING Statement” on page 308
- “OUTPUT Statement” on page 332
- “REMOVE Statement” on page 370
- “REPLACE Statement” on page 374
- “UPDATE Statement” on page 422

Null Statement

Signals the end of data lines or acts as a placeholder.

Valid in: Anywhere
Category: Action
Type: Executable

Syntax

;
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.

Example: Marking the End of Data Lines

• The Null statement in this program marks the end of data lines and functions as a step boundary.
  ```sas
  data test;
     input score1 score2 score3;
     datalines;
      55 135 177
     44 132 169
    ;
  
• 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.
  ```sas
  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
    ;;:
  
• 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.
  ```sas
  data _null_;
     set dsn;
     file print header=header;
     put 'report text';
     ...more statements...
     return;
     header:;
  run;
  ```

See Also

Statements:

• “DATALINES Statement” on page 53
• “DATALINES4 Statement” on page 54
• “GO TO Statement” on page 185
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 option(s);

Arguments

option
  specifies one or more SAS system options to be changed.

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 details, 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:

```sas
options nodate linesize=72;
```

See Also

“Definition of System Options” in SAS System Options: Reference

OUTPUT Statement

Writes the current observation to a SAS data set.

Valid in: DATA step

Category: Action

Type: Executable

Syntax

```sas
OUTPUT <data-set-name(s)>;
```

Without Arguments

Using OUTPUT without arguments causes the current observation to be written to all data sets that are named in the DATA statement.

If a MODIFY statement is present, OUTPUT with no arguments writes the current observation to the end of the data set that is specified in the MODIFY statement.

Arguments

data-set-name

specifies the name of a data set to which SAS writes the observation.

Restriction

All names specified in the OUTPUT statement must also appear in the DATA statement.

Tips

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

You can specify up to as many data sets in the OUTPUT statement as you specified in the DATA statement for that DATA step.

Details

When and Where the OUTPUT Statement Writes Observations

The OUTPUT statement tells SAS to write the current observation to a SAS data set immediately, not at the end of the DATA step. If no data set name is specified in the OUTPUT statement, the observation is written to the data set or data sets that are listed in the DATA statement.
Implicit versus Explicit Output

By default, every DATA step contains an implicit OUTPUT statement at the end of each iteration that tells SAS to write observations to the data set or data sets that are being created. Placing an explicit OUTPUT statement in a DATA step overrides the automatic output, and SAS adds an observation to a data set only when an explicit OUTPUT statement is executed. Once you use an OUTPUT statement to write an observation to any one data set, however, there is no implicit OUTPUT statement at the end of the DATA step. In this situation, a DATA step writes an observation to a data set only when an explicit OUTPUT executes. You can use the OUTPUT statement alone or as part of an IF-THEN or SELECT statement or in DO-loop processing.

When Using the MODIFY Statement

When you use the MODIFY statement with the OUTPUT statement, the REMOVE and REPLACE statements override the implicit write action at the end of each DATA step iteration. See “Comparisons” on page 333 for more information. If both the OUTPUT statement and a REPLACE or REMOVE statement execute on a given observation, perform the output action last to keep the position of the observation pointer correct.

Comparisons

• OUTPUT writes observations to a SAS data set; PUT writes variable values or text strings to an external file or the SAS log.

• To control when an observation is written to a specified output data set, use the OUTPUT statement. To control which variables are written to a specified output data set, use the KEEP= or DROP= data set option in the DATA statement, or use the KEEP or DROP statement.

• When you use the OUTPUT statement with the MODIFY statement, the following items apply.
  • Using an OUTPUT, REPLACE, or REMOVE statement overrides the default write action at the end of a DATA step. (OUTPUT is the default action; REPLACE becomes the default action when a MODIFY statement is used.) If you use any of these statements in a DATA step, you must explicitly program output for the new observations that are added to the data set.
  • The OUTPUT, REPLACE, and REMOVE statements are independent of each other. More than one statement can apply to the same observation, as long as the sequence is logical.
  • If both an OUTPUT and a REPLACE or REMOVE statement execute on a given observation, perform the OUTPUT action last to keep the position of the observation pointer correct.

Examples

Example 1: Sample Uses of OUTPUT

These examples show how you can use an OUTPUT statement:

• This line of code writes the current observation to a SAS data set.
  
  output;

• This line of code writes the current observation to a SAS data set when a specified condition is true.
  
  if deptcode > 2000 then output;
• This line of code writes an observation to the data set MARKUP when the PHONE value is missing.

    if phone=. then output markup;

**Example 2: Creating Multiple Observations from Each Line of Input**
You can create two or more observations from each line of input data. This SAS program creates three observations in the data set RESPONSE for each observation in the data set SULFA:

    data response(drop=time1-time3);
    set sulfa;
    time=time1;
    output;
    time=time2;
    output;
    time=time3;
    output;
    run;

**Example 3: Creating Multiple Data Sets from a Single Input File**
You can create more than one SAS data set from one input file. In this example, OUTPUT writes observations to two data sets, OZONE and OXIDES:

    data ozone oxides;
    infile file-specification;
    input city $ 1-15 date date9.
    chemical $ 26-27 ppm 29-30;
    if chemical='O3' then output ozone;
    else output oxides;
    run;

**Example 4: Creating One Observation from Several Lines of Input**
You can combine several input observations into one observation. In this example, OUTPUT creates one observation that totals the values of DEFECTS in the first ten observations of the input data set:

    data discards;
    set gadgets;
    drop defects;
    reps+1;
    if reps=1 then total=0;
    total+defects;
    if reps=10 then do;
    output;
    stop;
    end;
    run;

**See Also**

**Statements:**

• “DATA Statement” on page 44
• “MODIFY Statement” on page 309
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” on page 295

System Options:
- “LINESIZE= System Option” in SAS System Options: Reference
- “PAGESIZE= System Option” in SAS System Options: Reference

PUT Statement

Writes lines to the SAS log, to the SAS output window, or to an external location that is specified in the most recent FILE statement.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

PUT <specification(s)> _ODS_> <@ | @@>;
Without Arguments
The PUT statement without arguments is called a null PUT statement. The null PUT statement

- writes the current output line to the current location, even if the current output line is blank
- releases an output line that is being held with a trailing @ by a previous PUT statement.

For an example, see “Example 5: Holding and Releasing Output Lines” on page 351. For more information, see “Using Line-Hold Specifiers” on page 344.

Arguments

specification(s) specifies what is written, how it is written, and where it is written. The specification can include

variable specifies the variable whose value is written.

Note: Beginning with Version 7, you can specify column-mapped Output Delivery System variables in the PUT statement. This functionality is described briefly here in _ODS_ on page 337. It is documented more completely in the “PUT Statement for ODS ” in SAS Output Delivery System: User’s Guide.

(variable-list) specifies a list of variables whose values are written.

Requirement The (format-list) must follow the (variable-list).

See “PUT Statement, Formatted” on page 356

'character-string' specifies a string of text, enclosed in quotation marks, to write.

Tips To write a hexadecimal string in EBCDIC or ASCII, follow the ending quotation mark with an x.

If you use single quotation marks (‘) or double quotation marks (") together (with no space in between them) as the string of text, SAS will output a single quotation mark (' ) or double quotation mark ("), respectively.

See “List Output” on page 342

Example This statement writes HELLO when the hexadecimal string is converted to ASCII characters:
put ‘68656C6C6F’x;

n* specifies to repeat n times the subsequent character string.

Example This statement writes a line of 132 underscores.
put 132*’_’;

Example “Example 4: Underlining Text” on page 350
**pointer-control**

moves the output pointer to a specified line or column in the output buffer.

See  “Column Pointer Controls” on page 338

“Line Pointer Controls” on page 340

**column-specifications**

specifies which columns of the output line the values are written.

See  “Column Output” on page 342

Example  “Example 2: Moving the Pointer within a Page” on page 347

**format.**

specifies a format to use when the variable values are written.

See  “Formatted Output” on page 342

Example  “Example 1: Using Multiple Output Styles in One PUT Statement” on page 347

**(format-list)**

specifies a list of formats to use when the values of the preceding list of variables are written.

Restriction  The (format-list) must follow the (variable-list).

See  “PUT Statement, Formatted” on page 356

**_INFILE_**

writes the last input data record that is read either from the current input file or from the data lines that follow a DATELINES statement.

Tips  _INFILE_ is an automatic variable that references the current INPUT buffer. You can use this automatic variable in other SAS statements.

Example  This PUT statement writes all the values of the first input data record:

```
input #3 score #1 name $ 6-23;
put _infile_;
```

Example  “Example 6: Writing the Current Input Record to the Log” on page 351

**_ALL_**

writes the values of all variables, which includes automatic variables, that are defined in the current DATA step by using named output.

See  “Named Output” on page 343

**_ODS_**

moves data values for all columns (as defined by the ODS option in the FILE statement) into a special buffer, from which it is eventually written to the data
component. The ODS option in the FILE statement defines the structure of the data component that holds the results of the DATA step.

**Restriction**
Use _ODS_ only if you have previously specified the ODS option in the FILE statement.

**Interaction**
_ODS_ writes data to a specific column only if a PUT statement has not already specified a variable for that column with a column pointer. That is, a variable specification for a column overrides the _ODS_ option.

**Tip**
You can use the _ODS_ specification in conjunction with variable specifications and column pointers, and it can appear anywhere in a PUT statement.

**See**

@ @@

holds an output line for the execution of the next PUT statement even across iterations of the DATA step. These line-hold specifiers are called trailing @ and double trailing @@.

**Restriction**
The trailing @ or double trailing @@ must be the last item in the PUT statement.

**Tip**
Use an @ or @@ to hold the pointer at its current location. The next PUT statement that executes writes to the same output line rather than to a new output line.

**See**
“Using Line-Hold Specifiers” on page 344

**Example**
“Example 5: Holding and Releasing Output Lines” on page 351

## Column Pointer Controls

@n
moves the pointer to column n.

**Range**
a positive integer

**Example**
@15 moves the pointer to column 15 before the value of NAME is written:
put @15 name $10.;

**Examples**
“Example 2: Moving the Pointer within a Page” on page 347 and
“Example 4: Underlining Text” on page 350

@numeric-variable
moves the pointer to the column given by the value of numeric-variable.

**Range**
a positive integer

**Tip**
If n is not an integer, SAS truncates the decimal portion and uses only the integer value. If n is zero or negative, the pointer moves to column 1.
### PUT Statement

**Example**  
The value of the variable A moves the pointer to column 15 before the value of NAME is written:  
```  
a=15;  
put @a name $10.;  
```

**Example**  
“Example 2: Moving the Pointer within a Page” on page 347

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| `@()`           | Moves the pointer to the column that is given by the value of `expression`.
| **Range**       | A positive integer                                                          |
| **Tip**         | If the value of `expression` is not an integer, SAS truncates the decimal value and uses only the integer value. If it is zero, the pointer moves to column 1. |

**Example**  
The result of the expression moves the pointer to column 15 before the value of NAME is written:  
```  
b=5;  
put @(b*3) name $10.;  
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+n</code></td>
<td>Moves the pointer <code>n</code> columns.</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>A positive integer or zero</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>If <code>n</code> is not an integer, SAS truncates the decimal portion and uses only the integer value.</td>
</tr>
</tbody>
</table>

**Example**  
This statement moves the pointer to column 23, writes a value of LENGTH in columns 23 through 26, advances the pointer five columns, and writes the value of WIDTH in columns 32 through 35:  
```  
put @23 length 4. +5 width 4.;  
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+numeric-variable</code></td>
<td>Moves the pointer the number of columns given by the value of <code>numeric-variable</code>.</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>A positive or negative integer or zero</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>If <code>numeric-variable</code> is not an integer, SAS truncates the decimal value and uses only the integer value. If <code>numeric-variable</code> is negative, the pointer moves backward. If the current column position becomes less than 1, the pointer moves to column 1. If the value is zero, the pointer does not move. If the value is greater than the length of the output buffer, the current line is written out and the pointer moves to column 1 on the next line.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+(expression)</code></td>
<td>Moves the pointer the number of columns given by <code>expression</code>.</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><code>expression</code> must result in an integer</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>If <code>expression</code> is not an integer, SAS truncates the decimal value and uses only the integer value. If <code>expression</code> is negative, the pointer moves backward. If the current column position becomes less than 1, the pointer moves to column 1. If the value is zero, the pointer does not move. If the value is greater than the length of the output buffer, the current line is written out and the pointer moves to column 1 on the next line.</td>
</tr>
</tbody>
</table>
Line Pointer Controls

\#n
moves the pointer to line \( n \) and column 1.

Range a positive integer

Example The \#2 moves the pointer to the second line before the value of ID is written in columns 3 and 4:
put @12 name $10. #2 id 3-4;

\#numeric-variable
moves the pointer to the line given by the value of numeric-variable and to column 1.

Range a positive integer

Tip If the value of numeric-variable is not an integer, SAS truncates the decimal value and uses only the integer value.

\#(expression)
moves the pointer to the line that is given by the value of expression and to column 1.

Range Expression must result in a positive integer.

Tip If the value of expression is not an integer, SAS truncates the decimal value and uses only the integer value.

/ advances the pointer to column 1 of the next line.

Note If you try to use one or more “/” line pointer controls to add blank lines to the SAS log, SAS suppresses the blank lines. For other forms of output, the blank lines are produced.

Example The values for NAME and AGE are written on one line, and then the pointer moves to the second line to write the value of ID in columns 3 and 4:
put name age / id 3-4;

Example “Example 3: Moving the Pointer to a New Page” on page 349

OVERPRINT
causes the values that follow the keyword OVERPRINT to print on the most recently written output line.

Requirement You must direct the output to a file. Set the N= option in the FILE statement to 1 and direct the PUT statements to a file.

Tips OVERPRINT has no effect on lines that are written to a display.

Use OVERPRINT in combination with column pointer and line pointer controls to overprint text.
Example

This statement overprints underscores, starting in column 15, which underlines the title:

```plaintext
put @15 'Report Title' overprint
@15 '____________';
```

Example

“Example 4: Underlining Text” on page 350

__BLANKPAGE__ advances the pointer to the first line of a new page, even when the pointer is positioned on the first line and the first column of a new page.

Tip

If the current output file contains carriage-control characters, _BLANKPAGE_ produces output lines that contain the appropriate carriage-control character.

Example

“Example 3: Moving the Pointer to a New Page” on page 349

__PAGE__ advances the pointer to the first line of a new page. SAS automatically begins a new page when a line exceeds the current PAGESIZE= value.

Tips

If the current output file is printed, _PAGE_ produces an output line that contains the appropriate carriage-control character. _PAGE_ has no effect on a file that is not printed.

If you specify FILE PRINT in an interactive SAS session, then the Output window interprets the form-feed control characters as page breaks, and they are removed from the output. The resulting file is a flat file without page break characters. If a file needs to contain the form-feed characters, then the FILE statement should include a physical file location and the PRINT option.

Example

“Example 3: Moving the Pointer to a New Page” on page 349

Details

When to Use PUT

Use the PUT statement to write lines to the SAS log, to the SAS output window, or to an external location. If you do not execute a FILE statement before the PUT statement in the current iteration of a DATA step, SAS writes the lines to the SAS log. If you specify the PRINT option in the FILE statement, SAS writes the lines to the SAS output window.

The PUT statement can write lines that contain variable values, character strings, and hexadecimal character constants. With specifications in the PUT statement, you specify what to write, where to write it, and how to format it.

Output Styles

Overview of Output Styles

There are four ways to write variable values with the PUT statement:

- column
- list (simple and modified)
- formatted
A single PUT statement might contain any or all of the available output styles, depending on how you want to write lines.

**Column Output**
With *column output*, the column numbers follow the variable in the PUT statement. These numbers indicate where in the line to write the following value:

```
put name 6-15 age 17-19;
```

These lines are written to the SAS log.

*Note:* The ruled line is for illustrative purposes only; the PUT statement does not generate it.

```
---------1---------2---------
 Peterson  21
 Morgan    17
```

The PUT statement writes values for NAME and AGE in the specified columns. See “PUT Statement, Column” on page 354 for more information.

**List Output**
With *list output*, list the variables and character strings in the PUT statement in the order in which you want to write them. For example, this PUT statement writes the values for NAME and AGE to the SAS log.

```
put name age;
```

Here is the SAS log.

```
---------1---------2---------
 Peterson 21
 Morgan 17
```

*Note:* The ruled line is for illustrative purposes only; the PUT statement does not generate it.

See “PUT Statement, List” on page 360 for more information.

**Formatted Output**
With *formatted output*, specify a SAS format or a user-written format after the variable name. The format gives instructions on how to write the variable value. Formats enable you to write in a nonstandard form, such as packed decimal, or numbers that contain special characters such as commas. You can also override the default alignment of the formatted output by using -L, -C, or -R.

For example, this PUT statement writes the values for NAME, AGE, and DATE to the SAS log.

```
put name $char10. age 2. +1 date mmddyy10.;
```

Here is the SAS log.

```
---------1---------2---------
 Peterson  21 07/18/1999
 Morgan    17 11/12/1999
```

*Note:* The ruled line is for illustrative purposes only; the PUT statement does not generate it.

Using a pointer control of +1 inserts a blank space between the values of AGE and DATE. For more information, see “PUT Statement, Formatted” on page 356.
Named Output
With named output, list the variable name followed by an equal sign. For example, this PUT statement writes the values for NAME and AGE to the SAS log.

```plaintext
call name= age=;
```

Here is the SAS log.

```
-----+----1----+----2----+
name=Peterson age=21
name=Morgan age=17
```

*Note:* The ruled line is for illustrative purposes only; the PUT statement does not generate it.

See “PUT Statement, Named” on page 365 for more information.

Using Multiple Output Styles in a Single PUT Statement
A PUT statement can combine any or all of the different output styles. Here is an example.

```plaintext
put name 'on ' date mmddyy8. ' weighs ' startwght +(-1) '.' idno= 40-45;
```

See “Example 1: Using Multiple Output Styles in One PUT Statement” on page 347 for an explanation of the lines written to the SAS log.

When you combine different output styles, it is important to understand the location of the output pointer after each value is written. For more information about the pointer location, see “Pointer Location After a Value Is Written” on page 345.

Avoiding a Common Error When Writing Both a Character Constant and a Variable
When using a PUT statement to write a character constant that is followed by a variable name, always put a blank space between the closing quotation mark and the variable name:

```plaintext
put 'Player:' name1 'Player:' name2 'Player:' name3;
```

Otherwise, SAS might interpret a character constant that is followed by a variable name as a special SAS constant as illustrated in this table.

<table>
<thead>
<tr>
<th>Starting Letter of Variable</th>
<th>Represents</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>bit testing constant</td>
<td>'00100000'b</td>
</tr>
<tr>
<td>d</td>
<td>date constant</td>
<td>'01jan04'd</td>
</tr>
<tr>
<td>dt</td>
<td>datetime constant</td>
<td>'18jan2003:9:27:05am'</td>
</tr>
<tr>
<td>n</td>
<td>name literal</td>
<td>'My Table'n</td>
</tr>
<tr>
<td>t</td>
<td>time constant</td>
<td>'9:25:19pm't</td>
</tr>
<tr>
<td>x</td>
<td>hexadecimal notation</td>
<td>'534153'x</td>
</tr>
</tbody>
</table>
“Example 7: Avoiding a Common Error When Writing a Character Constant Followed by a Variable” on page 352 shows how to use character constants followed by variables. For more information about SAS name literals and SAS constants in expressions, see SAS Language Reference: Concepts.

**Pointer Controls**
As SAS writes values with the PUT statement, it keeps track of its position with a pointer. The PUT statement provides three ways to control the movement of the pointer:

- **column pointer controls**
  - reset the pointer's column position when the PUT statement starts to write the value to the output line.

- **line pointer controls**
  - reset the pointer's line position when the PUT statement writes the value to the output line.

- **line-hold specifiers**
  - hold a line in the output buffer so that another PUT statement can write to it. By default, the PUT statement releases the previous line and writes to a new line.

With column and line pointer controls, you can specify an absolute line number or column number to move the pointer or you can specify a column or line location that is relative to the current pointer position. The following table lists all pointer controls that are available in the PUT statement.

<table>
<thead>
<tr>
<th>Pointer Controls</th>
<th>Relative</th>
<th>Absolute</th>
</tr>
</thead>
<tbody>
<tr>
<td>column pointer controls</td>
<td>( +n )</td>
<td>@n )</td>
</tr>
<tr>
<td></td>
<td>(+\text{numeric-variable})</td>
<td>@\text{numeric-variable}</td>
</tr>
<tr>
<td></td>
<td>(+\text{expression})</td>
<td>@\text{expression}</td>
</tr>
<tr>
<td>line pointer controls</td>
<td>/<em>PAGE</em>/</td>
<td>#n )</td>
</tr>
<tr>
<td></td>
<td>_BLANKPAGE_</td>
<td>#\text{numeric-variable} #\text{expression}</td>
</tr>
<tr>
<td></td>
<td>OVERPRINT</td>
<td>none )</td>
</tr>
<tr>
<td>line-hold specifiers</td>
<td>@</td>
<td>(not applicable)</td>
</tr>
<tr>
<td></td>
<td>@@</td>
<td>(not applicable)</td>
</tr>
</tbody>
</table>

*Note:* Always specify pointer controls before the variable for which they apply.

See “Pointer Location After a Value Is Written” on page 345 for more information about how SAS determines the pointer position.

**Using Line-Hold Specifiers**
Line-hold specifiers keep the pointer on the current output line when
• more than one PUT statement writes to the same output line
• a PUT statement writes values from more than one observation to the same output line.

Without line-hold specifiers, each PUT statement in a DATA step writes a new output line.

In the PUT statement, trailing @ and double trailing @@ produce the same effect. Unlike the INPUT statement, the PUT statement does not automatically release a line that is held by a trailing @ when the DATA step begins a new iteration. SAS releases the current output line that is held by a trailing @ or double trailing @ when it encounters

• a PUT statement without a trailing @
• a PUT statement that uses _BLANKPAGE_ or _PAGE_
• the end of the current line (determined by the current value of the LRECL= or LINESIZE= option in the FILE statement, if specified, or the LINESIZE= system option)
• the end of the last iteration of the DATA step.

Using a trailing @ or double trailing @ can cause SAS to attempt to write past the current line length because the pointer value is unchanged when the next PUT statement executes. See “When the Pointer Goes Past the End of a Line” on page 345.

**Pointer Location After a Value Is Written**
Understanding the location of the output pointer after a value is written is important, especially if you combine output styles in a single PUT statement. The pointer location after a value is written depends on which output style you use and whether a character string or a variable is written. With column or formatted output, the pointer is located in the first column after the end of the field that is specified in the PUT statement. These two styles write only variable values.

With list output or named output, the pointer is located in the second column after a variable value because PUT skips a column automatically after each value is written. However, when a PUT statement uses list output to write a character string, the pointer is located in the first column after the string. If you do not use a line pointer control or column output after a character string is written, add a blank space to the end of the character string to separate it from the next value.

After an _INFILE_ specification, the pointer is located in the first column after the record is written from the current input file.

When the output pointer is in the upper left corner of a page,

• PUT _BLANKPAGE_ writes a blank page and moves the pointer to the top of the next page.
• PUT _PAGE_ leaves the pointer in the same location.

You can determine the current location of the pointer by examining the variables that are specified with the COLUMN= option and the LINE= option in the FILE statement.

**When the Pointer Goes Past the End of a Line**
SAS does not write an output line that is longer than the current output line length. The line length of the current output file is determined by

• the value of the LINESIZE= option in the current FILE statement
• the value of the LINESIZE= system option (for the SAS output window)
the LRECL= option in the current FILE statement (for external files).
You can inadvertently position the pointer beyond the current line length with one or more of these specifications:

- a + pointer control with a value that moves the pointer to a column beyond the current line length
- a column range that exceeds the current line length (for example, PUT X 90–100 when the current line length is 80)
- a variable value or character string that does not fit in the space that remains on the current output line.

By default, when PUT attempts to write past the end of the current line, SAS withholds the entire item that overflows the current line, writes the current line, and then writes the overflow item on a new line, starting in column 1. See the FLOWOVER, DROPOVER, and STOPOVER options in the “FILE Statement” on page 73.

Arrays
You can use the PUT statement to write an array element. The subscript is any SAS expression that results in an integer when the PUT statement executes. You can use an array reference in a numeric-variable construction with a pointer control if you enclose the reference in parentheses, as shown here:

- @(array-name{i})
- +(array-name{i})
- #(array-name{i})

Use the array subscript asterisk (*) to write all elements of a previously defined array to an external location. SAS allows one-dimensional or multidimensional arrays, but it does not allow a _TEMPORARY_ array. Enclose the subscript in braces, brackets, or parentheses, and print the array using list, formatted, column, or named output. With list output, the form of this statement is

PUT array-name{*};

With formatted output, the form of this statement is

PUT array-name{*}(format|format.list)

The format in parentheses follows the array reference.

Comparisons
- The PUT statement writes variable values and character strings to the SAS log or to an external location while the INPUT statement reads raw data in external files or data lines entered instream.
- Both the INPUT and the PUT statements use the trailing @ and double trailing @ line-hold specifiers to hold the current line in the input or output buffer, respectively. In an INPUT statement, a double trailing @ holds a line in the input buffer from one iteration of the DATA step to the next. In a PUT statement, however, a trailing @ has the same effect as a double trailing @; both hold a line across iterations of the DATA step.
- Both the PUT and OUTPUT statements create output in a DATA step. The PUT statement uses an output buffer and writes output lines to an external location, the SAS log, or your monitor. The OUTPUT statement uses the program data vector and writes observations to a SAS data set.
Examples

Example 1: Using Multiple Output Styles in One PUT Statement
This example uses several output styles in a single PUT statement:

data club1;
  input idno name $ startwght date : date7.;
  put name 'on ' date mmddyy8. ' weighs '
     startwght +(-1) '.' idno= 32-40;
datalines;
032 David 180 25nov99
049 Amelia 145 25nov99
219 Alan 210 12nov99
;

The following table shows the output style used for each variable in the example:

<table>
<thead>
<tr>
<th>Variables</th>
<th>Output Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME, STARTWGHT</td>
<td>list output</td>
</tr>
<tr>
<td>DATE</td>
<td>formatted output</td>
</tr>
<tr>
<td>IDNO</td>
<td>named output</td>
</tr>
</tbody>
</table>

The PUT statement also uses pointer controls and specifies both character strings and variable names.

The program writes the following lines to the SAS log:

----+----1----+----2----+----3----+----4
David on 11/25/99 weighs 180.  idno=1032
Amelia on 11/25/99 weighs 145. idno=1049
Alan on 11/12/99 weighs 210.   idno=1219

Note: The ruled line is for illustrative purposes only; the PUT statement does not generate it.

Blank spaces are inserted at the beginning and the end of the character strings to change the pointer position. These spaces separate the value of a variable from the character string. The +(-1) pointer control moves the pointer backward to remove the unwanted blank that occurs between the value of STARTWGHT and the period. For more information about how to position the pointer, see “Pointer Location After a Value Is Written” on page 345.

Example 2: Moving the Pointer within a Page
These PUT statements show how to use column and line pointer controls to position the output pointer.

• To move the pointer to a specific column, use @ followed by the column number, variable, or expression whose value is that column number. For example, this statement moves the pointer to column 15 and writes the value of TOTAL SALES using list output:
  put @15 totalsales;

This PUT statement moves the pointer to the value that is specified in COLUMN and writes the value of TOTALSALES with the COMMA6 format:
This program shows two techniques to move the pointer backward:

data carsales;
  input item $10. jan : comma5.  
    feb : comma5. mar : comma5. ;
  saleqtr1=sum(jan,feb,mar);
  /* an expression moves pointer backward */
  put '1st qtr sales for ' item 
'is ' saleqtr1 : comma6. +(-1) '.'; 
  /* a numeric variable with a negative 
  value moves pointer backward. */
  x=-1;
  put '1st qtr sales for ' item 
'is ' saleqtr1 : comma5. +x '.'; 
datalines;
trucks      1,382     2,789     3,556
vans        1,265     2,543     3,987
sedans      2,391     3,011     3,658 ;

Because the value of SALEQTR1 is written with modified list output, the pointer 
move automatically two spaces. For more information, see “How Modified List 
Output and Formatted Output Differ” on page 362. To remove the unwanted blank 
that occurs between the value and the period, move the pointer backward by one 
space.

The program writes the following lines to the SAS log:

----+----1----+----2----+----3----+----4
st qtr sales for trucks is 7,727.
st qtr sales for trucks is 7,727.
st qtr sales for vans is 7,795.
st qtr sales for vans is 7,795.
st qtr sales for sedans is 9,060.
st qtr sales for sedans is 9,060.

Note: The ruled line is for illustrative purposes only; the PUT statement does not 
generate it.

This program uses a PUT statement with the / line pointer control to advance to the 
next output line:

data _null_; 
  set carsales end=lastrec;
  totalsales+saleqtr1;
  if lastrec then 
    put @2 'Total Sales for 1st Qtr' 
      / totalsales 10-15;
run;

After the DATA step calculates TOTALSALES using all the observations in the 
CARSALES data set, the PUT statement executes. It writes a character string 
begining in column 2 and moves to the next line to write the value of 
TOTALSALES in columns 10 through 15:
Example 3: Moving the Pointer to a New Page

This example creates a data set called STATEPOP, which contains information from the 1990 U.S. census about the population of metropolitan and non-metropolitan areas. It executes the FORMAT procedure to group the 50 states and the District of Columbia into four regions. It then uses the IF and PUT statements to control the printed output.

```sas
title1;
proc format;
value regfmt 1='Northeast'
2='South'
3='Midwest'
4='West';
run;
proc format;
value regfmt 1='Northeast'
2='South'
3='Midwest'
4='West';
run;
data _null_;
set statepop;
by region;
pop90=sum(cityp90,ncityp90);
file print;
put state 1-2 @5 pop90 7.3 ' million';
if first.region then
  regioncitypop=0;   /* new region */
regioncitypop+cityp90;
if last.region then
do;
  put // '1990 US CENSUS for ' region regfmt.
    / 'Total Urban Population: ' regioncitypop' million' _page_;
end;
run;

Output 2.26   PUT Statement Output for the Northeast Region

<table>
<thead>
<tr>
<th>State</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME</td>
<td>1.228 million</td>
</tr>
<tr>
<td>NH</td>
<td>1.109 million</td>
</tr>
<tr>
<td>VT</td>
<td>0.563 million</td>
</tr>
<tr>
<td>MA</td>
<td>6.017 million</td>
</tr>
<tr>
<td>RI</td>
<td>1.003 million</td>
</tr>
<tr>
<td>CT</td>
<td>3.288 million</td>
</tr>
<tr>
<td>NY</td>
<td>17.990 million</td>
</tr>
<tr>
<td>NJ</td>
<td>7.730 million</td>
</tr>
<tr>
<td>PA</td>
<td>11.882 million</td>
</tr>
</tbody>
</table>

1990 US CENSUS for Northeast
Total Urban Population: 45.456 million

PUT _PAGE_ advances the pointer to line 1 of the new page when the value of LAST.REGION is 1. The example prints a footer message before exiting the page.

Example 4: Underlining Text
This example uses OVERPRINT to underscore a value written by a previous PUT statement:

data _null_;  
  input idno name $ startwght;
  file file-specification print;
  put name 1-10 @15 startwght 3.;
  if startwght > 200 then
    put overprint @15 '___';
  datalines;
  032 David 180
  049 Amelia 145
  219 Alan 210
;  
The second PUT statement underlines weights above 200 on the output line the first PUT statement prints.

This PUT statement uses OVERPRINT with both a column pointer control and a line pointer control:

put @5 name $8. overprint @5 8*'_'
  / @20 address;
The PUT statement writes a NAME value, underlines it by overprinting eight underscores, and moves the output pointer to the next line to write an ADDRESS value.

**Example 5: Holding and Releasing Output Lines**

This DATA step demonstrates how to hold and release an output line with a PUT statement:

```sas
data _null_;  
input idno name $ startwght 3.;  
put name @;  
if startwght ne . then  
   put @15 startwght;  
else put;  
datalines;  
032 David 180  
049 Amelia 145  
126 Monica  
219 Alan 210  
;
```

In this example,

- the trailing @ in the first PUT statement holds the current output line after the value of NAME is written
- if the condition is met in the IF-THEN statement, the second PUT statement writes the value of STARTWGHT and releases the current output line
- if the condition is not met, the second PUT never executes. Instead, the ELSE PUT statement executes. The ELSE PUT statement releases the output line and positions the output pointer at column 1 in the output buffer.

The program writes the following lines to the SAS log:

```
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Startwght</th>
</tr>
</thead>
<tbody>
<tr>
<td>032</td>
<td>David</td>
<td>180</td>
</tr>
<tr>
<td>049</td>
<td>Amelia</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Monica</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>Alan</td>
<td>210</td>
</tr>
</tbody>
</table>
```

*Note:* The ruled line is for illustrative purposes only; the PUT statement does not generate it.

**Example 6: Writing the Current Input Record to the Log**

When a value for ID is less than 1000, PUT _INFILE_ executes and writes the current input record to the SAS log. The DELETE statement prevents the DATA step from writing the observation to the TEAM data set.

```sas
data team;  
input id team $ score1 score2;  
if id le 1000 then  
do;  
   put _infile_;  
delete;  
end;  
datalines;  
032 red 180 165  
049 yellow 145 124  
219 red 210 192  
;```
The program writes the following line to the SAS log:

219 red 210 192

Note: The ruled line is for illustrative purposes only; the PUT statement does not generate it.

**Example 7: Avoiding a Common Error When Writing a Character Constant Followed by a Variable**

This example illustrates how to use a PUT statement to write character constants and variable values without causing them to be misinterpreted as SAS name literals. A SAS name literal is a name token that is expressed as a string within quotation marks, followed by the letter n. For more information about SAS name literals, see *SAS Language Reference: Concepts*.

In the program below, the PUT statement writes the constant 'n' followed by the value of the variable NVAR1, and then writes another constant 'n':

```sas
data _null_;
  n=5;
  nvar1=1;
  var1=7;
  put @1 'n' nvar1 'n';
run;
```

This program writes the following line to the SAS log:

219 red 210 192

Note: The ruled line is for illustrative purposes only; the PUT statement does not generate it.

If all the spaces between the constants and the variables are removed from the previous PUT statement, SAS interprets 'n' as a name literal instead of reading 'n' as a constant. The next variable is read as VAR1 instead of NVAR1. The final 'n' constant is interpreted correctly.

```sas
put @1 'n' nvar1 'n';
```

This PUT statement writes the following line to the SAS log:

219 red 210 192

To print character constants and variable values without intervening spaces, and without potential misinterpretation, you can add spaces between them and use pointer controls where necessary. For example, the following PUT statement uses a pointer control to write the correct character constants and variable values but does not insert blank spaces. Note that +(-1) moves the PUT statement pointer backward by one space.

```sas
put @1 'n' nvar1 +(-1) 'n';
```

This PUT statement writes the following line to the SAS log:

219 red 210 192

Chapter 2 • Dictionary of SAS Statements
Example 8: Creating Multi-Column Output
This example uses the #n and @n column and pointer controls to create multi-column output.

`/*
   * Use #i and @j to position name and weight information into
   * four columns in column-major order. That is print down column 1
   * first, then print down column 2, etc.
   * This example highlights the need to specify # before @ because
   * # sets the column pointer to 1.
   */
data _null_;    file print n=ps notitles header=hd;
  do i = 1 to 80 by 20;
    do j = 1 to ceil(num_students/4);
      set sashelp.class nobs=num_students;
      put #(j+3) @i name $8. '-' +1 weight 5.1;
    end;
  end;
  stop;
hd:
  put @26 'Student Weight in Pounds' / @26 24*'-.';   return;
run;

The program creates the following output:

<table>
<thead>
<tr>
<th>Student Weight in Pounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Alfred   - 112.5      James - 83.0      Joyce - 50.5      Robert - 128.0</td>
</tr>
<tr>
<td>Alice    - 84.0        Jane  - 84.5      Judy  - 90.0       Ronald - 133.0</td>
</tr>
<tr>
<td>Barbara  - 98.0        Janet - 112.5     Louise - 77.0      Thomas - 85.0</td>
</tr>
<tr>
<td>Carol    - 102.5       Jeffrey - 84.0    Mary   - 112.0      William - 112.0</td>
</tr>
<tr>
<td>Henry    - 102.5       John   - 99.5     Philip  - 150.0</td>
</tr>
</tbody>
</table>

See Also

Statements:
- “FILE Statement” on page 73
- “PUT Statement, Column” on page 354
- “PUT Statement, Formatted” on page 356
- “PUT Statement, List” on page 360
- “PUT Statement, Named” on page 365

System Options:
- “LINESIZE= System Option” in SAS System Options: Reference
- “PAGESIZE= System Option” in SAS System Options: Reference
PUT Statement, Column

Writes variable values in the specified columns in the output line.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>DATA step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>File-Handling</td>
</tr>
<tr>
<td>Type</td>
<td>Executable</td>
</tr>
</tbody>
</table>

**Syntax**

```
PUT variable start-column <- end-column>
<.decimal-places> <@ | @@>
```

**Arguments**

- **variable**
  - specifies the variable whose value is written.

- **start-column**
  - specifies the first column of the field where the value is written in the output line.

- **end-column**
  - specifies the last column of the field for the value.

**Tip**

If the value occupies only one column in the output line, omit `end-column`.

**Example**

Because `end-column` is omitted, the values for the character variable GENDER occupy only column 16:

```
put name 1-10 gender 16;
```

- **.decimal-places**
  - specifies the number of digits to the right of the decimal point in a numeric value.

**Range**

positive integer

**Tip**

If you specify 0 for `d` or omit `d`, the value is written without a decimal point.

**Example**

“Example: Using Column Output in the PUT Statement” on page 355

- **@ | @@**
  - holds an output line for the execution of the next PUT statement even across iterations of the DATA step. These line-hold specifiers are called `trailing @` and `double trailing @`.

**Requirement**

The trailing `@` or double trailing `@` must be the last item in the PUT statement.

**See**

“Using Line-Hold Specifiers” on page 344
Details

With column output, the column numbers indicate the position that each variable value will occupy in the output line. If a value requires fewer columns than specified, a character variable is left-aligned in the specified columns, and a numeric variable is right-aligned in the specified columns.

There is no limit to the number of column specifications that you can make in a single PUT statement. You can write anywhere in the output line, even if a value overwrites columns that were written earlier in the same statement. You can combine column output with any of the other output styles in a single PUT statement. For more information, see “Using Multiple Output Styles in a Single PUT Statement” on page 343.

Example: Using Column Output in the PUT Statement

Use column output in the PUT statement as shown here.

- This PUT statement uses column output:

```sas
data _null_
  input name $ 1-18 score1 score2 score3;
  put name 1-20 score1 23-25 score2 28-30 score3 33-35;
  datalines;
Joseph                  11   32   76
Mitchel                 13   29   82
Sue Ellen               14   27   74
;
```

The program writes the following lines to the SAS log:

```
Joseph                 11   32   76
Mitchel                13   29   82
Sue Ellen               14   27   74
```

Note: The ruled line is for illustrative purposes only; the PUT statement does not generate it.

The values for the character variable NAME begin in column 1, the left boundary of the specified field (columns 1 through 20). The values for the numeric variables SCORE1 through SCORE3 appear flush with the right boundary of their field.

- This statement produces the same output lines, but writes the SCORE1 value first and the NAME value last:

```sas
put score1 23-25 score2 28-30 score3 33-35 name $ 1-20;
```

- This DATA step specifies decimal points with column output:

```sas
data _null_
  x=11;
  y=15;
  put x 10-18 .1 y 20-28 .1;
run;
```

This program writes the following line to the SAS log:

```
11.0      15.0
```

PUT Statement, Column 355
PUT Statement, Formatted

Writes variable values with the specified format in the output line.

Valid in: DATA step
Category: File-Handling
Type: Executable

Syntax

PUT <pointer-control> variable format. <*> | @@ >

PUT <pointer-control> (variable-list) (format-list) <*> | @@ >

Arguments

pointer-control
moves the output pointer to a specified line or column.

variable
specifies the variable whose value is written.

(variable-list)
specifies a list of variables whose values are written.

(format)
specifies a format to use when the variable values are written. To override the default alignment, you can add an alignment specification to a format:

- L left aligns the value.
- C centers the value.
- R right aligns the value.

See Also

Statements:
- “PUT Statement” on page 335

Example 1: Writing a Character between Formatted Values” on page 359

“How to Group Variables and Formats” on page 358
Tip  Ensure that the format width provides enough space to write the value and any commas, dollar signs, decimal points, or other special characters that the format includes.

Examples  This PUT statement uses the format dollar7.2 to write the value of X:

```sas
put x dollar7.2;
```

When X is 100, the formatted value uses seven columns:

```
$100.00
```

Example  “Example 2: Overriding the Default Alignment of Formatted Values” on page 359

(format-list)

specifies a list of formats to use when the values of the preceding list of variables are written. In a PUT statement, a format-list can include

format.

specifies the format to use to write the variable values.

Tip  You can specify either a SAS format or a user-written format.

See  SAS Formats and Informats: Reference

pointer-control

specifies one of these pointer controls to use to position a value: @, #, /, +, and OVERPRINT.

Example  “Example 1: Writing a Character between Formatted Values” on page 359

character-string

specifies one or more characters to place between formatted values.

Example  This statement places a hyphen between the formatted values of CODE1, CODE2, and CODE3:

```sas
put bldg $ (code1 code2 code3) (3. '-');
```

Example  “Example 1: Writing a Character between Formatted Values” on page 359

n*

specifies to repeat n times the next format in a format list.

Restriction  The (format-list) must follow (variable-list).

See  “How to Group Variables and Formats” on page 358

Example  This statement uses the 7.2 format to write GRADES1, GRADES2, and GRADES3 and the 5.2 format to write GRADES4 and GRADES5:

```sas
put (grades1-grades5) (3*7.2, 2*5.2);
```

@| @@

holds an output line for the execution of the next PUT statement even across iterations of the DATA step. These line-hold specifiers are called trailing @ and double trailing @.
Restriction  The trailing @ or double trailing @ must be the last item in the PUT statement.

See  “Using Line-Hold Specifiers” on page 344

Details

Using Formatted Output

The Formatted output describes the output lines by listing the variable names and the formats to use to write the values. You can use a SAS format or a user-written format to control how SAS prints the variable values. For a complete description of the SAS formats, see “Definition of Formats” in SAS Formats and Informats: Reference.

With formatted output, the PUT statement uses the format that follows the variable name to write each value. SAS does not automatically add blanks between values. If the value uses fewer columns than specified, character values are left-aligned and numeric values are right-aligned in the field that is specified by the format width.

Formatted output, combined with pointer controls, makes it possible to specify the exact line and column location to write each variable. For example, this PUT statement uses the dollar7.2 format and centers the value of X starting at column 12:

```
put @12 x dollar7.2-c;
```

How to Group Variables and Formats

When you want to write values in a pattern on the output lines, use format lists to shorten your coding time. A format list consists of the corresponding formats separated by either blanks or commas and enclosed in parentheses. It must follow the names of the variables enclosed in parentheses.

For example, this statement uses a format list to write the five variables SCORE1 through SCORE5, one after another, using four columns for each value with no blanks in between:

```
put (score1-score5) {4. 4. 4. 4. 4.};
```

A shorter version of the previous statement is

```
put (score1-score5) {4.};
```

You can include any of the pointer controls (@, #, /, +, and OVERPRINT) in the list of formats, as well as n*, and a character string. You can use as many format lists as necessary in a PUT statement, but do not nest the format lists. After all the values in the variable list are written, the PUT statement ignores any directions that remain in the format list. For an example, see “Example 3: Including More Format Specifications Than Necessary” on page 359.

You can also specify a reference to all elements in an array as (array-name {*}), followed by a list of formats. However, you cannot specify the elements in a _TEMPORARY_ array in this way. This PUT statement specifies an array name and a format list:

```
put (array1{*}) {4.};
```

For more information about how to reference an array, see “Arrays” on page 346.
Examples

Example 1: Writing a Character between Formatted Values
This example formats some values and writes a - (hyphen) between the values of variables BLDG and ROOM:

```sas
data _null_;  
  input name $ & $15. bldg $ room;  
  put name @20 (bldg room) ($1. '-' 3.);  
  datalines;  
Bill Perkins  J 126  
Sydney Riley  C 219  
;  
These lines are written to the SAS log:

Bill Perkins       J-126
Sydney Riley       C-219
```

Example 2: Overriding the Default Alignment of Formatted Values
This example includes an alignment specification in the format:

```sas
data _null_;  
  input name $ 1-12 score1 score2 score3;  
  put name $12.-r +3 score1 3. score2 3. score3 4.;  
  datalines;  
Joseph                  11   32   76  
Mitchel                  13   29   82  
Sue Ellen                14   27   74  
;  
These lines are written to the SAS log:  

1  -----------1---+----2---+----3---+----4
Joseph          11 32  76
Mitchel          13 29  82
Sue Ellen        14 27  74

The value of the character variable NAME is right-aligned in the formatted field. (Left alignment is the default for character variables.)
```

Example 3: Including More Format Specifications Than Necessary
This format list includes more specifications than are necessary when the PUT statement executes:

```sas
data _null_;  
  input x y z;  
  put (x y z) (2.,+1);  
  datalines;  
2  24 36  
0  20 30  
;  
The PUT statement writes the value of X using the 2. format. Then, the +1 column pointer control moves the pointer forward one column. Next, the value of Y is written with the 2. format. Again, the +1 column pointer moves the pointer forward one column.
```

1 The ruled line is for illustrative purposes only; the PUT statement does not generate it.
Then, the value of Z is written with the 2. format. For the third iteration, the PUT statement ignores the +1 pointer control.

These lines are written to the SAS log:  

```
+-----+-----+-----+
| 2 24 36 |
| 0 20  30 |
```

**See Also**

**Statements:**

- “PUT Statement” on page 335

---

### PUT Statement, List

Writes variable values and the specified character strings in the output line.

**Valid in:**  
DATA step

**Category:**  
File-Handling

**Type:**  
Executable

---

**Syntax**

```plaintext
PUT <pointer-control> variable <@ | @@>;
PUT <pointer-control> <n*> 'character-string' <@ | @@>;
PUT <pointer-control> variable <: | ~> format.<@ | @@>;
```

**Arguments**

- **pointer-control**  
  moves the output pointer to a specified line or column.  
  See  
  “Column Pointer Controls ” on page 338
  “Line Pointer Controls ” on page 340
  **Example**  
  “Example 2: Writing Character Strings and Variable Values” on page 363

- **variable**  
  specifies the variable whose value is written.  
  **Example**  
  “Example 1: Writing Values with List Output” on page 363

- **n**  
  specifies to repeat n times the subsequent character string.  
  **Example**  
  This statement writes a line of 132 underscores:  
  ```plaintext
  put 132*'_' ;
  ```

---

1 The ruled line is for illustrative purposes only; the PUT statement does not generate it.
### `character-string`

specifies a string of text, enclosed in quotation marks, to write.

**Interaction**

When insufficient space remains on the current line to write the entire text string, SAS withholding the entire string and writes the current line. Then it writes the text string on a new line, starting in column 1. For more information, see “When the Pointer Goes Past the End of a Line” on page 345.

**Tips**

To avoid misinterpretation, always put a space after a closing quotation mark in a PUT statement.

If you follow a quotation mark with X, SAS interprets the text string as a hexadecimal constant.

If you use single quotation (’) or double quotation marks (“) together (with no space in between them) as the string of text, SAS will output a single quotation mark (‘) or double quotation mark (“), respectively.

**See**

“How List Output Is Spaced” on page 362

**Example**

“Example 2: Writing Character Strings and Variable Values” on page 363

`:`

enables you to specify a format that the PUT statement uses to write the variable value. All leading and trailing blanks are deleted, and each value is followed by a single blank.

**Requirement**

You must specify a format.

**See**

“How Modified List Output and Formatted Output Differ” on page 362

**Example**

“Example 3: Writing Values with Modified List Output (:)” on page 364

`~`

enables you to specify a format that the PUT statement uses to write the variable value. SAS displays the formatted value in quotation marks even if the formatted value does not contain the delimiter. SAS deletes all leading and trailing blanks, and each value is followed by a single blank. Missing values for character variables are written as a blank (" ") and, by default, missing values for numeric variables are written as a period (".").

**Requirement**

You must specify the DSD option in the FILE statement.

**Example**

“Example 4: Writing Values with Modified List Output and ~” on page 364

**format.**

specifies a format to use when the data values are written.

**Tip**

You can specify either a SAS format or a user-written format. See SAS Formats and Informats: Reference

**Example**

“Example 3: Writing Values with Modified List Output (:)” on page 364
@ | @@

holds an output line for the execution of the next PUT statement even across iterations of the DATA step. These line-hold specifiers are called trailing @ and double trailing @.

Restriction

The trailing @ or double-trailing @ must be the last item in the PUT statement.

See

“Using Line-Hold Specifiers” on page 344

Details

Using List Output

With list output, you list the names of the variables whose values you want written, or you specify a character string in quotation marks. The PUT statement writes a variable value, inserts a single blank, and then writes the next value. Missing values for numeric variables are written as a single period. Character values are left-aligned in the field; leading and trailing blanks are removed. To include blanks (in addition to the blank inserted after each value), use formatted or column output instead of list output.

There are two types of list output:

• simple list output
• modified list output.

Modified list output increases the versatility of the PUT statement because you can specify a format to control how the variable values are written. See “Example 3: Writing Values with Modified List Output (:)” on page 364.

How List Output Is Spaced

List output uses different spacing methods when it writes variable values and character strings. When a variable is written with list output, SAS automatically inserts a blank space. The output pointer stops at the second column that follows the variable value. However, when a character string is written, SAS does not automatically insert a blank space. The output pointer stops at the column that immediately follows the last character in the string.

To avoid spacing problems when both character strings and variable values are written, you might want to use a blank space as the last character in a character string. When a character string that provides punctuation follows a variable value, you need to move the output pointer backward. Moving the output pointer backward prevents an unwanted space from appearing in the output line. See “Example 2: Writing Character Strings and Variable Values” on page 363.

How Modified List Output and Formatted Output Differ

List output and formatted output use different methods to determine how far to move the pointer after a variable value is written. Therefore, modified list output, which uses formats, and formatted output produce different results in the output lines. Modified list output writes the value, inserts a blank space, and moves the pointer to the next column. Formatted output moves the pointer the length of the format, even if the value does not fill that length. The pointer moves to the next column; an intervening blank is not inserted.

The following DATA step uses modified list output to write each output line:

data _null_;
input x y;
put x : comma10.2 y : 7.2;
datalines;
2353.20 7.10
6231.00 121
;
These lines are written to the SAS log:

```
-------1-------2
2,353.20 7.10
6,231.00 121.00
```

In comparison, the following example uses formatted output:

put x comma10.2 y 7.2;

These lines are written to the SAS log, with the values aligned in columns:

```
-------1-------2
2,353.20   7.10
6,231.00 121.00
```

Examples

**Example 1: Writing Values with List Output**

This DATA step uses a PUT statement with list output to write variable values to the SAS log:

```sas
data _null_;  
input name $ 1-10 sex $ 12 age 15-16;  
put name sex age;  
datalines;  
Joseph     M  13  
Mitchel    M  14  
Sue Ellen  F  11  
;  
```

These lines are written to the SAS log:

```
-------1-------2-------3-------4
Joseph M 13  
Mitchel M 14  
Sue Ellen F 11
```

By default, the values of the character variable NAME are left-aligned in the field.

**Example 2: Writing Character Strings and Variable Values**

This PUT statement adds a space to the end of a character string and moves the output pointer backward to prevent an unwanted space from appearing in the output line after the variable STARTWGHT:

```sas
data _null_;  
input idno name $ startwght;  
put name 'weighs ' startwght +(-1) '.';  
datalines;  
032 David 180  
049 Amelia 145  
219 Alan 210
```

```
```
These lines are written to the SAS log:

David weighs 180.
Amelia weighs 145.
Alan weighs 210.

The blank space at the end of the character string changes the pointer position. This space separates the character string from the value of the variable that follows. The +(-1) pointer control moves the pointer backward to remove the unwanted blank that occurs between the value of STARTWGHT and the period.

**Example 3: Writing Values with Modified List Output (:)**

This DATA step uses modified list output to write several variable values in the output line using the : argument:

```sas
data _null_
    input salesrep : $10. tot : comma6. date : date9.;
    put 'Week of ' date : worddate15.
        salesrep : $12. 'sales were '
        tot : dollar9. + (-1) '.
    datalines;
Wong 15,300 12OCT2004
Hoffman 9,600 12OCT2004
```

These lines are written to the SAS log:

Week of Oct 12, 2004 Wong sales were $15,300.
Week of Oct 12, 2004 Hoffman sales were $9,600.

**Example 4: Writing Values with Modified List Output and ~**

This DATA step uses modified list output to write several variable values in the output line using the ~ argument:

```sas
data _null_
    input salesrep : $10. tot : comma6. date : date9.;
    file log delimiter=" " dsd;
    put 'Week of ' date - worddate15.
        salesrep : $12. 'sales were '
        tot - dollar9. + (-1) '.
    datalines;
Wong 15,300 12OCT2004
Hoffman 9,600 12OCT2004
```

These lines are written to the SAS log:

Week of "Oct 12, 2004" "Wong" sales were "$15,300".
Week of "Oct 12, 2004" "Hoffman" sales were "$9,600".

**See Also**

**Statements:**

- “PUT Statement” on page 335
- “PUT Statement, Formatted” on page 356
**PUT Statement, Named**

Writes variable values after the variable name and an equal sign.

- **Valid in:** DATA step
- **Category:** File-Handling
- **Type:** Executable

### Syntax

PUT `<pointer-control>` `variable`=`<format>` `<@ | @@>`;

PUT `variable`=`start-column `<end-column>` `<decimal-places>` `<@ | @@>`;

### Arguments

**pointer-control**

- moves the output pointer to a specified line or column in the output buffer.

See “Column Pointer Controls” on page 338

“Line Pointer Controls” on page 340

**variable**

- specifies the variable whose value is written by the PUT statement in the form `variable=value`

**format.**

- specifies a format to use when the variable values are written.

**Tip** Ensure that the format width provides enough space to write the value and any commas, dollar signs, decimal points, or other special characters that the format includes.

See “Formatting Named Output” on page 366

**Examples**

This PUT statement uses the format DOLLAR7.2 to write the value of X:

```
put x= dollar7.2;
```

When X=100, the formatted value uses seven columns:

```
X=$100.00
```

**start-column**

- specifies the first column of the field where the variable name, equal sign, and value are to be written in the output line.

**end-column**

- determines the last column of the field for the value.

**Tip** If the variable name, equal sign, and value require more space than the columns specified, PUT will write past the end column rather than truncate the value. You must leave enough space before beginning the next value.
.decimal-places
specifies the number of digits to the right of the decimal point in a numeric value. If you specify 0 for \( d \) or omit \( d \), the value is written without a decimal point.

Range positive integer

@ | @@
holds an output line for the execution of the next PUT statement even across iterations of the DATA step. These line-hold specifiers are called trailing @ and double trailing @.

Restriction The trailing @ or double trailing @ must be the last item in the PUT statement.

See “Using Line-Hold Specifiers” on page 344

Details

Using Named Output
With named output, follow the variable name with an equal sign in the PUT statement. You can use either list output, column output, or formatted output specifications to indicate how to position the variable name and values. To insert a blank space between each variable value automatically, use list output. To align the output in columns, use pointer controls or column specifications.

Formatting Named Output
You can specify either a SAS format or a user-written format to control how SAS prints the variable values. The width of the format does not include the columns required by the variable name and equal sign. To align a formatted value, SAS deletes leading blanks and writes the variable value immediately after the equal sign. SAS does not align on the right side of the formatted length, as in unnamed formatted output.

For a complete description of the SAS formats, see “Definition of Formats” in SAS Formats and Informats: Reference.

Example: Using Named Output in the PUT Statement
Use named output in the PUT statement as shown here.

- This PUT combines named output with column pointer controls to align the output:

```sas
   data _null_;   
      input name $ 1-18 score1 score2 score3; 
      put name = @20 score1= score3= ; 
   datalines; 
   Joseph                  11   32   76 
   Mitchel                 13   29   82 
   Sue Ellen               14   27   74 
   ;
```

The program writes the following lines to the SAS log:

```
----+----1----+----2----+----3----+----4
NAME=Joseph        SCORE1=11 SCORE3=76
NAME=Mitchel       SCORE1=13 SCORE3=82
NAME=Sue Ellen     SCORE1=14 SCORE3=74
```

366 Chapter 2 • Dictionary of SAS Statements
This example specifies an output format for the variable AMOUNT:

```
put item= @25 amount= dollar12.2;
```

When the value of ITEM is binders and the value of AMOUNT is 153.25, this output line is produced:

```
---------1---------2---------3---------4
ITEM=binders            AMOUNT=$153.25
```

See Also

Statements:

- “PUT Statement” on page 335

---

**PUTLOG Statement**

Writes a message to the SAS log.

- **Valid in:** DATA step
- **Category:** Action
- **Type:** Executable

**Syntax**

```
PUTLOG 'message';
```

**Arguments**

- **message**

  Specifies the message that you want to write to the SAS log. *Message* can include character literals (enclosed in quotation marks), variable names, formats, and pointer controls.

  **Tip** You can precede your message text with WARNING, MESSAGE, or NOTE to better identify the output in the log.

**Details**

The PUTLOG statement writes a message that you specify to the SAS log. The PUTLOG statement is also helpful when you use macro-generated code because you can send output to the SAS log without affecting the current file destination.

**Comparisons**

The PUTLOG statement is similar to the ERROR statement except that PUTLOG does not set _ERROR_ to 1.
Example: Writing Messages to the SAS Log Using the PUTLOG Statement

The following program creates the computeAverage92 macro, which computes the average score, validates input data, and uses the PUTLOG statement to write error messages to the SAS log. The DATA step uses the PUTLOG statement to write a warning message to the log.

data ExamScores;
  input Name $ 1-16 Score1 Score2 Score3;
datalines;
Sullivan, James 86 92 88
Martinez, Maria 95 91 92
Guzik, Eugene 99 98 .
Schultz, John 90 87 93
van Dyke, Sylvia 98 . 91
Tan, Carol 93 85 85;

filename outfile 'path-to-your-output-file';
/* Create a macro that computes the average score, validates */
/* input data, and uses PUTLOG to write error messages to the */
/* SAS log. */
%macro computeAverage92(s1, s2, s3, avg);
if &s1 < 0 or &s2 < 0 or &s3 < 0 then
  do;
    putlog 'ERROR: Invalid score data ' &s1= &s2= &s3=;
    &avg = .;
  end;
else
  &avg = mean(&s1, &s2, &s3);
%mend;

data _null_;
set ExamScores;
file outfile;
%computeAverage92(Score1, Score2, Score3, AverageScore);
put name Score1 Score2 Score3 AverageScore;
/* Use PUTLOG to write a warning message to the SAS log. */
if AverageScore < 92 then
  putlog 'WARNING: Score below the minimum ' name= AverageScore= 5.2;
run;

proc print;
run;

The following lines are written to the SAS log.

<table>
<thead>
<tr>
<th>Warning Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARNING: Score below the minimum Name=Sullivan, James AverageScore=88.67</td>
</tr>
<tr>
<td>ERROR: Invalid score data Score1=99 Score2=98 Score3=.</td>
</tr>
<tr>
<td>WARNING: Score below the minimum Name=Guzik, Eugene AverageScore=.</td>
</tr>
<tr>
<td>WARNING: Score below the minimum Name=Schultz, John AverageScore=90.00</td>
</tr>
<tr>
<td>ERROR: Invalid score data Score1=98 Score2=. Score3=91</td>
</tr>
<tr>
<td>WARNING: Score below the minimum Name=van Dyke, Sylvia AverageScore=.</td>
</tr>
<tr>
<td>WARNING: Score below the minimum Name=Tan, Carol AverageScore=87.67</td>
</tr>
</tbody>
</table>

SAS creates the following output file.
The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Score1</th>
<th>Score2</th>
<th>Score3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sullivan, James</td>
<td>86</td>
<td>92</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>Martinez, Maria</td>
<td>95</td>
<td>91</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>Guzik, Eugene</td>
<td>99</td>
<td>98</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>Schultz, John</td>
<td>90</td>
<td>87</td>
<td>93</td>
</tr>
<tr>
<td>5</td>
<td>van Dyke, Sylvia</td>
<td>98</td>
<td>.</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>Tan, Carol</td>
<td>93</td>
<td>85</td>
<td>85</td>
</tr>
</tbody>
</table>

See Also

Statements:

- “ERROR Statement” on page 71

REDIRECT Statement

Points to different input or output SAS data sets when you execute a stored program.

Valid in: DATA step
Category: Action
Type: Executable
Requirement: You must specify the PGM= option in the DATA statement.

Syntax

REDIRECT INPUT | OUTPUT old-name-1=new-name-1 <…old-name-n=new-name-n>;

Arguments

INPUT | OUTPUT

specifies whether to redirect input or output data sets. When you specify INPUT, the REDIRECT statement associates the name of the input data set in the source program with the name of another SAS data set. When you specify OUTPUT, the REDIRECT statement associates the name of the output data set with the name of another SAS data set.

old-name

specifies the name of the input or output data set in the source program.

new-name

specifies the name of the input or output data set that you want SAS to process for the current execution.
Details

The REDIRECT statement is available only when you execute a stored program. For more information about stored programs, see “Stored Compiled DATA Step Programs” in *SAS Language Reference: Concepts*.

**CAUTION:**

*Use care when you redirect input data sets.* The number and attributes of variables in the input data sets that you read with the REDIRECT statement should match the number and attributes of variables in the input data sets in the MERGE, SET, MODIFY, or UPDATE statements of the source code. If the variable type attributes differ, the stored program stops processing and an appropriate error message is written to the SAS log. If the variable length attributes differ, the length of the variable in the source code data set determines the length of the variable in the redirected data set. Extra variables in the redirected data sets cause the stored program to stop processing and an error message is written to the SAS log.

**TIP**  The DROP or KEEP data set options can be added in the stored program if the input data set that you read with the REDIRECT statement has more variables than are in the data set used to compile the program.

Comparisons

The REDIRECT statement applies only to SAS data sets. To redirect input and output stored in external files, include a FILENAME statement to associate the fileref in the source program with different external files.

Example: Executing a Stored Program

This example executes the stored program called STORED.SAMPLE. The REDIRECT statement specifies the source of the input data as BASE.SAMPLE. The output data set from this execution of the program is redirected and stored in a data set named SUMS.SAMPLE.

```sas
libname stored 'SAS-library';
libname base 'SAS-library';
libname sums 'SAS-library';
data pgm=stored.sample;
  redirect input in.sample=base.sample;
  redirect output out.sample=sums.sample;
run;
```

See Also

• “Stored Compiled DATA Step Programs” in *SAS Language Reference: Concepts*

Statements:

• “DATA Statement” on page 44

---

**REMOVE Statement**

Deletes an observation from a SAS data set.

**Valid in:**  DATA step
Category: Action
Type: Executable
Restriction: Use only with a MODIFY statement.

## Syntax

```plaintext
REMOVE <data-set-name(s)>;
```

### Without Arguments

If you specify no argument, the REMOVE statement deletes the current observation from all data sets that are named in the DATA statement.

### Arguments

- **data-set-name**
  
  Specifies the data set in which the observation is deleted.

### Restriction

The data set name must also appear in the DATA statement and in one or more MODIFY statements.

### Tip

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

## Details

The deletion of an observation can be physical or logical, depending on the engine that maintains the data set. Using REMOVE overrides the default replacement of observations. If a DATA step contains a REMOVE statement, you must explicitly program all output for the step.

## Comparisons

- Using an OUTPUT, REPLACE, or REMOVE statement overrides the default write action at the end of a DATA step. (OUTPUT is the default action; REPLACE becomes the default action when a MODIFY statement is used.) If you use any of these statements in a DATA step, you must explicitly program all output for new observations.

- The OUTPUT, REPLACE, and REMOVE statements are independent of each other. More than one statement can apply to the same observation, as long as the sequence is logical.

- If both an OUTPUT and a REPLACE or REMOVE statement execute on a given observation, perform the OUTPUT action last to keep the position of the observation pointer correct.

- Because the REMOVE statement can perform a physical or a logical deletion, REMOVE is available with the MODIFY statement for all SAS data set engines. Both the DELETE and subsetting IF statements perform only physical deletions. Therefore, they are not available with the MODIFY statement for certain engines.

## Example: Removing an Observation from a Data Set

This example removes one observation from a SAS data set.
libname perm 'SAS-library';

data perm.accounts;
  input AcctNumber Credit;
  datalines;
  1001 1500
  1002 4900
  1003 3000
;  
  data perm.accounts;
    modify perm.accounts;
    if AcctNumber=1002 then remove;
  run;
  proc print data=perm.accounts;
    title 'Edited Data Set';
  run;

Here are the results of the PROC PRINT statement:

*Output 2.28*   *Edited Data Set*

<table>
<thead>
<tr>
<th>Obs</th>
<th>AcctNumber</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>1500</td>
</tr>
<tr>
<td>3</td>
<td>1003</td>
<td>3000</td>
</tr>
</tbody>
</table>

**See Also**

**Statements:**

- “DELETE Statement” on page 55
- “IF Statement, Subsetting” on page 187
- “MODIFY Statement” on page 309
- “OUTPUT Statement” on page 332
- “REPLACE Statement” on page 374

**RENAME Statement**

Specifies new names for variables in output SAS data sets.

Valid in:  DATA step

Category: Information

Type: Declarative
Syntax

`RENAME old-name-1=new-name-1 <...old-name-n=new-name-n>;`

Arguments

old-name

specifies the name of a variable or variable list as it appears in the input data set, or in the current DATA step for newly created variables.

new-name

specifies the name or list to use in the output data set.

Details

The RENAME statement enables you to change the names of one or more variables, variables in a list, or a combination of variables and variable lists. The new variable names are written to the output data set only. Use the old variable names in programming statements for the current DATA step. RENAME applies to all output data sets.

Note: The RENAME statement has an effect on data sets opened in output mode only.

Comparisons

- RENAME cannot be used in PROC steps, but the RENAME= data set option can.
- The RENAME= data set option enables you to specify the variables that you want to rename for each input or output data set. Use it in input data sets to rename variables before processing.
- If you use the RENAME= data set option in an output data set, you must continue to use the old variable names in programming statements for the current DATA step. After your output data is created, you can use the new variable names.
- The RENAME= data set option in the SET statement renames variables in the input data set. You can use the new names in programming statements for the current DATA step.
- To rename variables as a file management task, use the DATASETS procedure or access the variables through the SAS windowing interface. These methods are simpler and do not require DATA step processing.

Example: Renaming Data Set Variables

- These examples show the correct syntax for renaming variables using the RENAME statement:
  
  ```
  rename street=address;
  rename time1=temp1 time2=temp2 time3=temp3;
  rename name=Firstname score1-score3=Newscore1-Newscore3;
  ```

- This example uses the old name of the variable in program statements. The variable Olddept is named Newdept in the output data set, and the variable Oldaccount is named Newaccount.

  ```
  rename Olddept=Newdept Oldaccount=Newaccount;
  if Oldaccount>5000;
  keep Olddept Oldaccount items volume;
  ```
This example uses the old name OLDACCNT in the program statements. However, the new name NEWACCNT is used in the DATA statement because SAS applies the RENAME statement before it applies the KEEP= data set option.

```sas
data market(keep=newdept newacct items volume);
   rename olddept=newdept oldacct=newacct;
  set sales;
   if oldacct>5000;
run;
```

The following example uses both a variable and a variable list to rename variables. New variable names appear in the output data set.

```sas
data temp;
   input (score1-score3) (2.,+1) name $;
   rename name=Firstname
score1-score3=Newscore1-Newscore3;
  datalines;
12 24 36 Lisa
22 44 66 Fran
;```

See Also

Data Set Options:

- “RENAME= Data Set Option” in *SAS Data Set Options: Reference*

---

**REPLACE Statement**

Replaces an observation in the same location.

- **Valid in:** DATA step
- **Category:** Action
- **Type:** Executable
- **Restriction:** Use only with a MODIFY statement.

**Syntax**

```sas
REPLACE <data-set-name-1> <...data-set-name-n>;
```

**Without Arguments**

If you specify no argument, the REPLACE statement writes the current observation to the same physical location from which it was read in all data sets that are named in the DATA statement.

**Arguments**

- `data-set-name`
  
  specifies the data set to which the observation is written.

  **Requirement** The data set name must also appear in the DATA statement and in one or more MODIFY statements.
Tip Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

Details

Using an explicit REPLACE statement overrides the default replacement of observations. If a DATA step contains a REPLACE statement, explicitly program all output for the step.

Comparisons

- Using an OUTPUT, REPLACE, or REMOVE statement overrides the default write action at the end of a DATA step. (OUTPUT is the default action; REPLACE becomes the default action when a MODIFY statement is used.) If you use any of these statements in a DATA step, you must explicitly program output of a new observation for the step.

- The OUTPUT, REPLACE, and REMOVE statements are independent of each other. More than one statement can apply to the same observation, as long as the sequence is logical.

- If both an OUTPUT and a REPLACE or REMOVE statement execute on a given observation, perform the OUTPUT action last to keep the position of the observation pointer correct.

- REPLACE writes the observation to the same physical location. OUTPUT writes a new observation to the end of the data set.

- REPLACE can appear only in a DATA step that contains a MODIFY statement. You can use OUTPUT with or without MODIFY.

Example: Replacing Observations

This example updates phone numbers in data set MASTER with values in data set TRANS. It also adds one new observation at the end of data set MASTER. The SYSRC autocall macro tests the value of _IORC_ for each attempted retrieval from MASTER. (SYSRC is part of the SAS autocall macro library.) The resulting SAS data set appears after the code:

```sas
data master;
  input FirstName $ id $ PhoneNumber;
  datalines;
  Kevin ABCjkh 904
  Sandi defsns 905
  Terry ghitDP 951
  Jason jklJWM 962
; data trans;
  input FirstName $ id $ PhoneNumber;
  datalines;
  . ABCjkh 2904
  . defsns 2905
  Madeline mnombt 2983
; data master;
```
modify master trans;
by id;
/* obs found in master */
/* change info, replace */
if _iorc_ = %sysrc(_sok) then replace;
/* obs not in master */
else if _iorc_ = %sysrc(_dsenmr) then
do;
/* reset _error_ */
_error_=0;
/* reset _iorc_ */
_iorc_=0;
/* output obs to master */
output;
end;
run;
proc print data=master;
  title 'MASTER with New Phone Numbers';
run;

Output 2.29  Data Set with Replaced Observations

<table>
<thead>
<tr>
<th>Obs</th>
<th>FirstName</th>
<th>id</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kevin</td>
<td>ABCjkh</td>
<td>2904</td>
</tr>
<tr>
<td>2</td>
<td>Sandi</td>
<td>defsns</td>
<td>2905</td>
</tr>
<tr>
<td>3</td>
<td>Terry</td>
<td>ghtDP</td>
<td>951</td>
</tr>
<tr>
<td>4</td>
<td>Jason</td>
<td>jkUWM</td>
<td>962</td>
</tr>
<tr>
<td>5</td>
<td>Madeline</td>
<td>nnombt</td>
<td>2983</td>
</tr>
</tbody>
</table>

See Also

Statements:
- “MODIFY Statement” on page 309
- “OUTPUT Statement” on page 332
- “REMOVE Statement” on page 370

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

The following example resets the program line numbers between DATA steps.

```sas
data a;
  a=1;
run;
resetline;
data b;
  b=2;
run;
```

The following lines are written 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
```
RETAIN Statement

Causes a variable that is created by an INPUT or assignment statement to retain its value from one iteration of the DATA step to the next.

Valid in:     DATA step
Category:    Information
Type:        Declarative

Syntax

RETAIN <element-list(s) <initial-value(s) | (initial-value-list-1)> <... element-list-n <initial-value-n | (initial-value-list-n)> >> ;

Without Arguments

If you do not specify an argument, the RETAIN statement causes the values of all variables that are created with INPUT or assignment statements to be retained from one iteration of the DATA step to the next.

Arguments

element-list

specifies variable names, variable lists, or array names whose values you want retained.

Tips

If you specify _ALL_, _CHAR_, or _NUMERIC_, only the variables that are defined before the RETAIN statement are affected.

If a variable name is specified only in the RETAIN statement and you do not specify an initial value, the variable is not written to the data set, and a note stating that the variable is uninitialized is written to the SAS log. If you specify an initial value, the variable is written to the data set.

initial-value

specifies an initial value, numeric or character, for one or more of the preceding elements.

Tip

If you omit initial-value, the initial value is missing. Initial-value is assigned to all the elements that precede it in the list. All members of a variable list, therefore, are given the same initial value.

See

(initial-value) and (initial-value-list)

(initial-value)

specifies an initial value, numeric or character, for a single preceding element or for the first in a list of preceding elements.

(initial-value-list)

specifies an initial value, numeric or character, for individual elements in the preceding list. SAS matches the first value in the list with the first variable in the list of elements, the second value with the second variable, and so on.
Element values are enclosed in quotation marks. To specify one or more initial values directly, use the following format:

\[(initial-value(s))\]

To specify an iteration factor and nested sublists for the initial values, use the following format:

\[<constant-iter-value*> \langle constant value | constant-sublist\rangle\]

**Restriction** If you specify both an initial-value-list and an element-list, then element-list must be listed before initial-value-list in the RETAIN statement.

**Tips**

You can separate initial values by blank spaces or commas.

You can also use a shorthand notation for specifying a range of sequential integers. The increment is always +1.

You can assign initial values to both variables and temporary data elements.

If there are more variables than initial values, the remaining variables are assigned an initial value of missing and SAS issues a warning message.

### Details

**Default DATA Step Behavior**

Without a RETAIN statement, SAS automatically sets variables that are assigned values by an INPUT or assignment statement to missing before each iteration of the DATA step.

**Assigning Initial Values**

Use a RETAIN statement to specify initial values for individual variables, a list of variables, or members of an array. If a value appears in a RETAIN statement, variables that appear before it in the list are set to that value initially. (If you assign different initial values to the same variable by naming it more than once in a RETAIN statement, SAS uses the last value.) You can also use RETAIN to assign an initial value other than the default value of 0 to a variable whose value is assigned by a sum statement.

**Redundancy**

It is redundant to name any of these items in a RETAIN statement, because their values are automatically retained from one iteration of the DATA step to the next:

- variables that are read with a SET, MERGE, MODIFY or UPDATE statement
- a variable whose value is assigned in a sum statement
- the automatic variables _N_, _ERROR_, _I_, _CMD_, and _MSG_
- variables that are created by the END= or IN= option in the SET, MERGE, MODIFY, or UPDATE statement or by options that create variables in the FILE and INFILE statements
- data elements that are specified in a temporary array
- array elements that are initialized in the ARRAY statement
elements of an array that have assigned initial values to any or all of the elements in
the ARRAY statement.

However, you can use a RETAIN statement to assign an initial value to any of the
previous items, with the exception of _N_ and _ERROR_.

**Comparisons**

The RETAIN statement specifies variables whose values are *not set to missing* at the
beginning of each iteration of the DATA step. The KEEP statement specifies variables
that are to be included in any data set that is being created.

**Examples**

**Example 1: Basic Usage**

- This RETAIN statement retains the values of variables MONTH1 through MONTH5
  from one iteration of the DATA step to the next:

  ```sas
  retain month1-month5;
  ```

- This RETAIN statement retains the values of nine variables and sets their initial
  values:

  ```sas
  retain month1-month5 1 year 0 a b c 'XYZ';
  ```

  The values of MONTH1 through MONTH5 are set initially to 1; YEAR is set to 0;
  variables A, B, and C are each set to the character value *XYZ*.

- This RETAIN statement assigns the initial value 1 to the variable MONTH1 only:

  ```sas
  retain month1-month5 (1);
  ```

  Variables MONTH2 through MONTH5 are set to missing initially.

- This RETAIN statement retains the values of all variables that are defined earlier in
  the DATA step but not the values that are defined *afterwards*:

  ```sas
  retain _all_;
  ```

- All of these statements assign initial values of 1 through 4 to VAR1 through VAR4:

  ```sas
  retain var1-var4 (1 2 3 4);
  ```

  ```sas
  retain var1-var4 (1,2,3,4);
  ```

  ```sas
  retain var1-var4(1:4);
  ```

**Example 2: Overview of the RETAIN Operation**

This example shows how to use variable names and array names as elements in the
RETAIN statement and shows assignment of initial values with and without parentheses:

```sas
data _null_;
  array City{3} $ City1-City3;
  array cp{3} Citypop1-Citypop3;
  retain Year Taxyear 1999 City ' '
    cp (10000,50000,100000);
  file file-specification print;
  put 'Values at beginning of DATA step:'
    / @3 _all_ /
  / @3 _all_ /
  input Gain;
  do i=1 to 3;
    cp{i}=cp{i}+Gain;
```
end;
put 'Values after adding Gain to city populations:'
   / @3 _all_
;datalines;
5000
10000
;
Here are the initial values assigned by RETAIN:
• Year and Taxyear are assigned the initial value 1999.
• City1, City2, and City3 are assigned missing values.
• Citypop1 is assigned the value 10000.
• Citypop2 is assigned 50000.
• Citypop3 is assigned 100000.

Here are the lines written by the PUT statements:

| Values at beginning of DATA step: |
| City1= City2= City3= Citypop1=10000 |
| Citypop2=50000 Citypop3=100000 |
| Year=1999 Taxyear=1999 Gain=. i=. |
| _ERROR_=0 _N_=1 |
| Values after adding GAIN to city populations: |
| City1= City2= City3= Citypop1=15000 |
| Citypop2=55000 Citypop3=105000 |
| Year=1999 Taxyear=1999 Gain=5000 i=4 |
| _ERROR_=0 _N_=1 |
| Values at beginning of DATA step: |
| City1= City2= City3= Citypop1=15000 |
| Citypop2=55000 Citypop3=105000 |
| Year=1999 Taxyear=1999 Gain=. i=. |
| _ERROR_=0 _N_=2 |
| Values after adding GAIN to city populations: |
| City1= City2= City3= Citypop1=25000 |
| Citypop2=65000 Citypop3=115000 |
| Year=1999 Taxyear=1999 Gain=10000 i=4 |
| _ERROR_=0 _N_=2 |
| Values at beginning of DATA step: |
| City1= City2= City3= Citypop1=25000 |
| Citypop2=65000 Citypop3=115000 |
| Year=1999 Taxyear=1999 Gain=. i=. |
| _ERROR_=0 _N_=3 |

The first PUT statement is executed three times, whereas the second PUT statement is executed only twice. The DATA step ceases execution when the INPUT statement executes for the third time and reaches the end of the file.

Example 3: Selecting One Value from a Series of Observations
In this example, the data set ALLSCORES contains several observations for each identification number and variable ID. Different observations for a particular ID value might have different values of the variable GRADE. This example creates a new data set, CLASS.BESTSCORES, which contains one observation for each ID value. The observation must have the highest GRADE value of all observations for that ID in BESTSCORES.

libname class 'SAS-library';
proc sort data=class.allscores;
   by id;
run;
data class.bestscores;
    drop grade;
set class.allscores;
by id;
    /* Prevents HIGHEST from being reset*/
    /* to missing for each iteration. */
retain highest;
    /* Sets HIGHEST to missing for each */
    /* different ID value. */
if first.id then highest=.;
    /* Compares HIGHEST to GRADE in */
    /* current iteration and resets */
    /* value if GRADE is higher. */
    highest=max(highest,grade);
    if last.id then output;
run;

See Also

Statements:

• “Assignment Statement” on page 26
• “BY Statement” on page 31
• “INPUT Statement” on page 232

RETURN Statement

Stops executing statements at the current point in the DATA step and returns to a predetermined point in the step.

Valid in: DATA step
Category: Control
Type: Executable

Syntax

RETURN;

Without Arguments

The RETURN statement causes execution to stop at the current point in the DATA step, and returns control to a previous DATA step statement.

Details

The point to which SAS returns depends on the order in which statements are executed in the DATA step.

The RETURN statement is often used with the

• GO TO statement
• HEADER= option in the FILE statement
• LINK statement.

When RETURN causes a return to the beginning of the DATA step, an implicit OUTPUT statement writes the current observation to any new data sets (unless the DATA step contains an explicit OUTPUT statement, or REMOVE or REPLACE statements with MODIFY statements). Every DATA step has an implied RETURN as its last executable statement.

Example: Basic Usage

In this example, when the values of X and Y are the same, SAS executes the RETURN statement and adds the observation to the data set. When the values of X and Y are not equal, SAS executes the remaining statements and then adds the observation to the data set.

```sas
data survey;
  input x y;
  if x=y then return;
  put x= y=;
datalines;
21 25
20 20
7 17
; 
```

See Also

Statements:

• “FILE Statement” on page 73
• “GO TO Statement” on page 185
• “LINK Statement” on page 293

---

**RUN Statement**

Executes the previously entered SAS statements.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anywhere</td>
<td>Program Control</td>
</tr>
</tbody>
</table>

**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:

```sas
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.

```sas
data circle;
  infile file-specification;
  input radius;
  c=2*4.13*radius;
run cancel;
```

The following 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

```sas
%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:

```sas
%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 the following statement to return processing to the program that contains the %INCLUDE statement.

```sas
%run;
```

See Also

Statements:

- “%INCLUDE Statement” on page 191
- “RUN Statement” on page 383

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
Syntax

SASFILE <libref> member-name<member-type> <(password-option(s))> 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).

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 that 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. Valid value is DATA, which is the default.

password-option(s)
specifies one or more of the following password options:

ENCRIPTKEY=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 Language Reference: Concepts

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 390.

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. Once 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, which automatically closes the file and frees the buffers.

Using the SASFILE statement can improve performance by reducing the following:

• 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.

The following 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 2.9  Processing Requests for a File Opened by SASFILE

<table>
<thead>
<tr>
<th>Processing Request</th>
<th>Open Mode</th>
<th>Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND procedure</td>
<td>update</td>
<td>Yes</td>
</tr>
<tr>
<td>DATA step that creates or replaces the file</td>
<td>output</td>
<td>No</td>
</tr>
<tr>
<td>DATASETS procedure to rename or add a variable, add or change a label, or add or remove integrity constraints or indexes</td>
<td>utility</td>
<td>No</td>
</tr>
<tr>
<td>DATASETS procedure with AGE, CHANGE, or DELETE statements</td>
<td>does not open the file but requires exclusive access</td>
<td>No</td>
</tr>
<tr>
<td>FSEDIT procedure</td>
<td>update</td>
<td>Yes</td>
</tr>
<tr>
<td>PRINT procedure</td>
<td>input</td>
<td>Yes</td>
</tr>
<tr>
<td>SORT procedure that replaces original data set with sorted one</td>
<td>output</td>
<td>No</td>
</tr>
<tr>
<td>SQL procedure to modify, add, or delete observations</td>
<td>update</td>
<td>Yes</td>
</tr>
<tr>
<td>SQL procedure with CREATE TABLE or CREATE VIEW statement</td>
<td>output</td>
<td>No</td>
</tr>
<tr>
<td>SQL procedure to create or remove integrity constraints or indexes</td>
<td>utility</td>
<td>No</td>
</tr>
</tbody>
</table>

**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 not an 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
The following 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.

• Once the SASFILE statement is executed, all users who subsequently open the file will access the data held in memory instead of data that is stored on the disk.

• Once 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
The following 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;
   set mydata.census;
run;
data test2;
   set mydata.census;
run;
proc summary data=mydata.census print;
run;
data mydata.census;
   modify mydata.census;
   .
   . (statements to modify data)
   .
run;
sasfile mydata.census close;
```

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
The following 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;
proc print data=mydata.census (read=gizmo);
run;
data mydata.census;
   modify mydata.census (alter=luke);
   .
   . (statements to modify data)
```

The SASFILE statement specifies the READ password, which is sufficient to open the file.

In the PRINT procedure, the READ password must be specified again.

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

SELECT Statement

Executes one of several statements or groups of statements.

Valid in: DATA step
Category: Control
Type: Executable

Syntax

SELECT <(select-expression)> ;

WHEN-1 (when-expression-1 <..., when-expression-n>) statement;
<... WHEN-n (when-expression-1 <...,when-expression-n>) statement;>
< OTHERWISE statement;>

END;

Arguments

(select-expression)

specifies any SAS expression that evaluates to a single value.

See “Evaluating the when-expression When a select-expression Is Included” on page 392

(when-expression)

specifies any SAS expression, including a compound expression. SELECT requires you to specify at least one when-expression.
Tips

Separating multiple when-expressions with a comma is equivalent to separating them with the logical operator OR.

The way a when-expression is used depends on whether a select-expression is present.

See “Evaluating the when-expression When a select-expression Is Not Included” on page 392

statement

can be any executable SAS statement, including DO, SELECT, and null statements. You must specify the statement argument.

Details

Using WHEN Statements in a SELECT Group

The SELECT statement begins a SELECT group. SELECT groups contain WHEN statements that identify SAS statements that are executed when a particular condition is true. Use at least one WHEN statement in a SELECT group. An optional OTHERWISE statement specifies a statement to be executed if no WHEN condition is met. An END statement ends a SELECT group.

Null statements that are used in WHEN statements cause SAS to recognize a condition as true without taking further action. Null statements that are used in OTHERWISE statements prevent SAS from issuing an error message when all WHEN conditions are false.

Evaluating the when-expression When a select-expression Is Included

If the select-expression is present, SAS evaluates the select-expression and when-expression. SAS compares the two for equality and returns a value of true or false. If the comparison is true, statement is executed. If the comparison is false, execution proceeds either to the next when-expression in the current WHEN statement, or to the next WHEN statement if no more expressions are present. If no WHEN statements remain, execution proceeds to the OTHERWISE statement, if one is present. If the result of all SELECT-WHEN comparisons is false and no OTHERWISE statement is present, SAS issues an error message and stops executing the DATA step.

Evaluating the when-expression When a select-expression Is Not Included

If no select-expression is present, the when-expression is evaluated to produce a result of true or false. If the result is true, statement is executed. If the result is false, SAS proceeds to the next when-expression in the current WHEN statement, or to the next WHEN statement if no more expressions are present, or to the OTHERWISE statement if one is present. (That is, SAS performs the action that is indicated in the first true WHEN statement.) If the result of all when-expressions is false and no OTHERWISE statement is present, SAS issues an error message. If more than one WHEN statement has a true when-expression, only the first WHEN statement is used. Once a when-expression is true, no other when-expressions are evaluated.

Processing Large Amounts of Data with %INCLUDE Files

One way to process large amounts of data is to use %INCLUDE statements in your DATA step. Using %INCLUDE statements enables you to perform complex processing while keeping your main program manageable. The %INCLUDE files that you use in
your main program can contain WHEN statements and other SAS statements to process your data. See “Example 5: Processing Large Amounts of Data” on page 394 for an example.

Comparisons

Use IF-THEN/ELSE statements for programs with few statements. Use subsetting IF statements without a THEN clause to continue processing only those observations or records that meet the condition that is specified in the IF clause.

The SELECT statement works much like the CASE statement in the SQL procedure.

Examples

Example 1: Using Statements

```sas
select (a);
when (1) x=x*10;
when (2);
when (3,4,5) x=x*100;
otherwise;
end;
```

Example 2: Using DO Groups

```sas
select (payclass);
when ('monthly') amt=salary;
when ('hourly')
do;
    amt=hlywage*min(hrs,40);
    if hrs>40 then put 'CHECK TIMECARD';
end; /* end of do */
otherwise put 'PROBLEM OBSERVATION';
end; /* end of select */
```

Example 3: Using a Compound Expression

```sas
select;
when (mon in ('JUN', 'JUL', 'AUG')
    and temp>70) put 'SUMMER ' mon=;
when (mon in ('MAR', 'APR', 'MAY'))
    put 'SPRING ' mon=;
otherwise put 'FALL OR WINTER ' mon=;
end;
```

Example 4: Making Comparisons for Equality

```sas
/* INCORRECT usage to select value of 2 */
select (x);
/* evaluates T/F and compares for */
/* equality with x */
when (x=2) put 'two';
end;
/* correct usage */
select(x);
/* compares 2 to x for equality */
when (2) put 'two';
```
Example 5: Processing Large Amounts of Data

In the following example, the %INCLUDE statements contain code that includes WHEN statements to process new and old items in the inventory. The main program shows the overall logic of the DATA step.

```sas
data test (keep=ItemNumber);
  set ItemList;
  select;
    %include NewItems;
    %include OldItems;
    otherwise put 'Item ' ItemNumber ' is not in the inventory.';
  end;
run;
```

See Also

Statements:
- “DO Statement” on page 61
- “IF Statement, Subsetting” on page 187
- “IF-THEN/ELSE Statement” on page 189

SET Statement

Reads an observation from one or more SAS data sets.

Valid in: DATA step
Category: File-Handling
Type: Executable

Note: The variables read using the SET statement are retained in the PDV. For more information, see “Overview of DATA Step Processing” in SAS Language Reference: Concepts and the “RETAIN Statement” on page 378.

Tip: The SAS Tutorial video Reading a SAS Data Set shows you how to use the SET statement to read data.

Syntax

```
SET <SAS-data-set(s) <(data-set-options(s))> > <options>;
```

Without Arguments

When you do not specify an argument, the SET statement reads an observation from the most recently created data set.
**Arguments**

*SAS-data-set (s)*

specifies a one-level name, a two-level name, or one of the special SAS data set names.

**Tips**

You can specify data set lists. For more information, see “Using Data Set Lists with SET” on page 400.

Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

**See**

See “SAS Data Sets” in *SAS Language Reference: Concepts* for a description of the levels of SAS data set names and when to use each level.

**Example**

“Example 13: Using Data Set Lists” on page 405

*(data-set-options)*

specifies actions SAS is to take when it reads variables or observations into the program data vector for processing.

**Tip**

Data set options that apply to a data set list apply to all of the data sets in the list.

**See**

For more information, see “Definition of Data Set Options” in *SAS Data Set Options: Reference* for a list of the data set options to use with input data sets.

**SET Options**

**CUROBS=variable**

creates and names a variable that contains the observation number that was just read from the data set.

**Example**

“Example 14: Finding the Current Observation Number” on page 407

**END=variable**

creates and names a temporary variable that contains an end-of-file indicator. The variable, which is initialized to zero, is set to 1 when SET reads the last observation of the last data set listed. This variable is not added to any new data set.

**Restriction**

END= cannot be used with POINT=. When random access is used, the END= variable is never set to 1.

**Interaction**

If you use a BY statement, END= is set to 1 when the SET statement reads the last observation of the interleaved data set. For more information, see “BY-Group Processing with SET” on page 401.

**Example**

“Example 11: Writing an Observation Only After All Observations Have Been Read” on page 404

**KEY=index</UNIQUE>**

provides nonsequential access to observations in a SAS data set, which are based on the value of an index variable or a key.
Range

Specify the name of a simple or a composite index of the data set that is being read.

Restriction

KEY= cannot be used with POINT=.

Tips

Using the _IORC_ automatic variable in conjunction with the SYSRC autocall macro provides you with more error-handling information than was previously available. When you use the SET statement with the KEY= option, the new automatic variable _IORC_ is created. This automatic variable is set to a return code that shows the status of the most recent I/O operation that is performed on an observation in a SAS data set. If the KEY= value is not found, the _IORC_ variable returns a value that corresponds to the SYSRC autocall macro's mnemonic _DSENOM and the automatic variable _ERROR_ is set to 1.

When using the SET statement with the KEY= option and a non-unique index, it is often desirable to force the SET statement to start reading again with the first observation that matches the key value. Use the KEYRESET= option to control whether a KEY= search should begin at the top of the index for the data set that is being read.

See

For more information, see the description of the autocall macro SYSRC in *SAS Macro Language: Reference*.

“KEYRESET=variable” on page 396

Examples

“Example 7: Performing a Table Lookup” on page 403

“Example 8: Performing a Table Lookup When the Master File Contains Duplicate Observations” on page 403

CAUTION

Continuous loops can occur when you use the KEY= option. If you use the KEY= option without specifying the primary data set, you must include either a STOP statement to stop DATA step processing, or programming logic that uses the _IORC_ automatic variable in conjunction with the SYSRC autocall macro and checks for an invalid value of the _IORC_ variable, or both.

KEYRESET=variable

does not control whether a KEY= search should begin at the top of the index for the data set that is being read. When the value of the KEYRESET variable is 1, the index lookup begins at the top of the index. When the value of the KEYRESET variable is 0, the index lookup is not reset and the lookup continues where the prior ended.

Interaction

The KEYRESET= option is similar to the UNIQUE option, except the KEYRESET= option enables you to determine when the KEY= search should begin at the top of the index again.

See

“KEY=index</UNIQUE>” on page 395

“UNIQUE” on page 399

Example

“Example 15: Using the KEYRESET Option” on page 407
INDSNAME=variable
creates and names a variable that stores the name of the SAS data set from which the current observation is read. The stored name can be a data set name or a physical name. The physical name is the name by which the operating environment recognizes the file.

Tips
For data set names, SAS adds the library name to the variable value (for example, WORK.PRICE) and converts the two-level name to uppercase.
Unless previously defined, the length of the variable is set to 41 bytes. Use a LENGTH statement to make the variable length long enough to contain the value of the physical filename if the filename is longer than 41 bytes.
If the variable is previously defined as a character variable with a specific length, that length is not changed. If the value placed into the INDSNAME variable is longer than that length, then the value is truncated.
If the variable is previously defined as a numeric variable, an error will occur.
The variable is available in the DATA step, but the variable is not added to any output data set.

Example
“Example 12: Retrieving the Name of the Data Set from Which the Current Observation Is Read” on page 404

NOBS=variable
creates and names a temporary variable whose value is usually the total number of observations in the input data set or data sets. If more than one data set is listed in the SET statement, NOBS= the total number of observations in the data sets that are listed. The number of observations includes those observations that are marked for deletion but are not yet deleted.

Restriction
For certain SAS views and sequential engines such as the TAPE and XML engines, SAS cannot determine the number of observations. In these cases, SAS sets the value of the NOBS= variable to the largest positive integer value that is available in your operating environment.

Interaction
The NOBS= and POINT= options are independent of each other.

Tip
At compilation time, SAS reads the descriptor portion of each data set and assigns the value of the NOBS= variable automatically. Thus, you can refer to the NOBS= variable before the SET statement. The variable is available in the DATA step but is not added to any output data set.

Example
“Example 10: Performing a Function until the Last Observation Is Reached” on page 404

OPEN=(IMMEDIATE | DEFER)
enables you to delay the opening of any concatenated SAS data sets until they are ready to be processed.

IMMEDIATE
during the compilation phase, opens all data sets that are listed in the SET statement.
Restriction When you use the IMMEDIATE option KEY=, POINT=, and BY statement processing are mutually exclusive.

Tip If a variable on a subsequent data set is of a different type (character versus numeric, for example) than the type of the same-named variable on the first data set, the DATA step will stop processing and produce an error message.

DEFER
opens the first data set during the compilation phase, and opens subsequent data sets during the execution phase. When the DATA step reads and processes all observations in a data set, it closes the data set and opens the next data set in the list.

Restriction When you specify the DEFER option, you cannot use the KEY= statement option, the POINT= statement option, or the BY statement. These constructs imply either random processing or interleaving of observations from the data sets, which is not possible unless all data sets are open.

Requirement You can use the DROP=, KEEP=, or RENAME= data set options to process a set of variables, but the set of variables that are processed for each data set must be identical. In most cases, if the set of variables defined by any subsequent data set differs from the variables defined by the first data set, SAS prints a warning message to the log but does not stop execution.

• If a variable on a subsequent data set is of a different type (character versus numeric, for example) than the type of the same-named variable on the first data set, the DATA step will stop processing and produce an error message.

• If a variable on a subsequent data set was not defined by the first data set in the SET statement, but was defined previously in the DATA step program, the DATA step will stop processing and produce an error message. In this case, the value of the variable in previous iterations might be incorrect because the semantic behavior of SET requires this variable to be set to missing when processing the first observation of the first data set.

Default IMMEDIATE

POINT=variable
 specifies a temporary variable whose numeric value determines which observation is read. POINT= causes the SET statement to use random (direct) access to read a SAS data set.

Restrictions You cannot use POINT= with a BY statement, a WHERE statement, or a WHERE= data set option. In addition, you cannot use it with transport format data sets, data sets in sequential format on tape or disk, and SAS/ACCESS views or the SQL procedure views that read data from external files.

You cannot use POINT= with KEY=.

Requirement a STOP statement
<table>
<thead>
<tr>
<th>Note</th>
<th>Remember that <em>N</em> is an iteration count and not the observation number of the last observation that was read.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tips</td>
<td>You must supply the values of the POINT= variable. For example, you can use the POINT= variable as the index variable in some form of the DO statement.</td>
</tr>
<tr>
<td></td>
<td>The POINT= variable is available anywhere in the DATA step, but it is not added to any new SAS data set.</td>
</tr>
</tbody>
</table>
| Examples | “Example 6: Combining One Observation with Many” on page 403  
“Example 9: Reading a Subset By Using Direct Access” on page 404 |
| CAUTION | **Continuous loops can occur when you use the POINT= option.**  
When you use the POINT= option, you must include a STOP statement to stop DATA step processing, programming logic that checks for an invalid value of the POINT= variable, or both. Because POINT= reads only those observations that are specified in the DO statement, SAS cannot read an end-of-file indicator as it would if the file were being read sequentially. Because reading an end-of-file indicator ends a DATA step automatically, failure to substitute another means of ending the DATA step when you use POINT= can cause the DATA step to go into a continuous loop. If SAS reads an invalid value of the POINT= variable, it sets the automatic variable _ERROR_ to 1. Use this information to check for conditions that cause continuous DO-loop processing, or include a STOP statement at the end of the DATA step, or both. |
| UNIQUE | causes a KEY= search always to begin at the top of the index for the data set that is being read. |
| Restriction | UNIQUE can appear only with the KEY= argument and must be preceded by a slash. |
| Notes | By default, SET begins searching at the top of the index only when the KEY= value changes.  
If the KEY= value does not change on successive executions of the SET statement, the search begins by following the most recently retrieved observation. In other words, when consecutive duplicate KEY= values appear, the SET statement attempts a one-to-one match with duplicate indexed values in the data set that is being read. If more consecutive duplicate KEY= values are specified than exist in the data set that is being read, the extra duplicates are treated as not found.  
When KEY= is a unique value, only the first attempt to read an observation with that key value succeeds; subsequent attempts to read the observation with that value of the key will fail. The _IORC_ variable returns a value that corresponds to the SYSRC autocall macro's mnemonic _DSENOM. If you add the /UNIQUE option, subsequent attempts to read the observation with the unique KEY= value will succeed. The _IORC_ variable returns a 0. |
See
For extensive examples, see *Combining and Modifying SAS Data Sets: Examples*

“KEYRESET=variable” on page 396

Example
“Example 8: Performing a Table Lookup When the Master File Contains Duplicate Observations” on page 403

Details

**What SET Does**
Each time the SET statement is executed, SAS reads one observation into the program data vector. SET reads all variables and all observations from the input data sets unless you tell SAS to do otherwise. A SET statement can contain multiple data sets; a DATA step can contain multiple SET statements. See *Combining and Modifying SAS Data Sets: Examples*.

*Note:* When the DATA step comes to an end-of-file marker or the end of all open data sets, it will perform an orderly shutdown. For example, if you use SET with FIRSTOBS, a file with only a header record in a series of files will trigger a normal shutdown of the DATA step. The shutdown occurs because SAS reads beyond the end-of-file marker and the DATA step terminates. You can use the END= option to avoid the shutdown.

**Uses**
The SET statement is flexible and has a variety of uses in SAS programming. These uses are determined by the options and statements that you use with the SET statement:

- reading observations and variables from existing SAS data sets for further processing in the DATA step
- concatenating and interleaving data sets, and performing one-to-one reading of data sets
- reading SAS data sets by using direct access methods.

**Using Data Set Lists with SET**
You can use data set lists with the SET statement. Data set lists provide a quick way to reference existing groups of data sets. These data set lists must be either name prefix lists or numbered range lists.

*Name prefix lists* refer to all data sets that begin with a specified character string. For example, `set SALES1:;` tells SAS to read all data sets starting with "SALES1" such as SALES1, SALES10, SALES11, and SALES12.

*Numbered range lists* require you to have a series of data sets with the same name, except for the last character or characters, which are consecutive numbers. In a numbered range list, you can begin with any number and end with any number. For example, these lists refer to the same data sets:

```
sales1 sales2 sales3 sales4
sales1-sales4
```

*Note:* If the numeric suffix of the first data set name contains leading zeros, the number of digits in the numeric suffix of the last data set name must be greater than or equal to the number of digits in the first data set name. Otherwise, an error will occur. For example, the data set lists sales001–sales99 and sales01–sales9 will cause an error. The data set list sales001–sales999 is valid. If the numeric suffix of the first data set
name does not contain leading zeros, the number of digits in the numeric suffix of the first and last data set names do not have to be equal. For example, the data set list sales1–sales999 is valid.

Some other rules to consider when using numbered data set lists are as follows:

- You can specify groups of ranges.
  
  ```
  set cost1-cost4 cost11-cost14 cost21-cost24;
  ```

- You can mix numbered range lists with name prefix lists.

  ```
  set cost1-cost4 cost2: cost33-37;
  ```

- You can mix single data sets with data set lists.

  ```
  set cost1 cost10-cost20 cost30;
  ```

- Quotation marks around data set lists are ignored.

  ```
  /* these two lines are the same */
  set sales1 - sales4;
  set 'sales1'n - 'sales4'n;
  ```

- Spaces in data set names are invalid. If quotation marks are used, trailing blanks are ignored.

  ```
  /* blanks in these statements will cause errors */
  set sales 1 - sales 4;
  set 'sales 1'n - 'sales 4'n;
  /* trailing blanks in this statement will be ignored */
  set 'sales1   'n - 'sales4   'n;
  ```

- The maximum numeric suffix is 2147483647.

  ```
  /* this suffix will cause an error */
  set prod2000000000-prod2934850239;
  ```

**BY-Group Processing with SET**

Only one BY statement can accompany each SET statement in a DATA step. The BY statement should immediately follow the SET statement to which it applies. The data sets that are listed in the SET statement must be sorted by the values of the variables that are listed in the BY statement, or they must have an appropriate index. SET, when it is used with a BY statement, interleaves data sets. The observations in the new data set are arranged by the values of the BY variable or variables, and within each BY group, by the order of the data sets in which they occur. See “Example 2: Interleaving SAS Data Sets” on page 402 for an example of BY-group processing with the SET statement.

**Combining SAS Data Sets**

Use a single SET statement with multiple data sets to concatenate the specified data sets. That is, the number of observations in the new data set is the sum of the number of observations in the original data sets, and the order of the observations is all the observations from the first data set followed by all the observations from the second data set, and so on. See “Example 1: Concatenating SAS Data Sets” on page 402 for an example of concatenating data sets.

Use a single SET statement with a BY statement to interleave the specified data sets. The observations in the new data set are arranged by the values of the BY variable or variables, and within each BY group, by the order of the data sets in which they occur. See “Example 2: Interleaving SAS Data Sets” on page 402 for an example of interleaving data sets.
Use multiple SET statements to perform one-to-one reading (also called one-to-one matching) of the specified data sets. The new data set contains all the variables from all the input data sets. The number of observations in the new data set is the number of observations in the smallest original data set. If the data sets contain common variables, the values that are read in from the last data set replace the values that were read in from earlier ones. For examples of one-to-one reading of data sets, see

- “Example 6: Combining One Observation with Many” on page 403
- “Example 7: Performing a Table Lookup” on page 403
- “Example 8: Performing a Table Lookup When the Master File Contains Duplicate Observations” on page 403

For extensive examples, see *Combining and Modifying SAS Data Sets: Examples*.

For more information about how to prepare your data sets, see “Combining SAS Data Sets: Basic Concepts” in *SAS Language Reference: Concepts*.

**Comparisons**

- SET reads an observation from an existing SAS data set. INPUT reads raw data from an external file or from in-stream data lines in order to create SAS variables and observations.
- Using the KEY= option with SET enables you to access observations nonsequentially in a SAS data set according to a value. Using the POINT= option with SET enables you to access observations nonsequentially in a SAS data set according to the observation number.

**Examples**

**Example 1: Concatenating SAS Data Sets**

If more than one data set name appears in the SET statement, the resulting output data set is a concatenation of all the data sets that are listed. SAS reads all observations from the first data set, then all from the second data set, and so on, until all observations from all the data sets have been read. This example concatenates the three SAS data sets into one output data set named FITNESS:

```sas
data fitness;
  set health exercise well;
run;
```

**Example 2: Interleaving SAS Data Sets**

To interleave two or more SAS data sets, use a BY statement after the SET statement:

```sas
data april;
  set payable recvable;
  by account;
run;
```

**Example 3: Reading a SAS Data Set**

In this DATA step, each observation in the data set NC.MEMBERS is read into the program data vector. Only those observations whose value of CITY is Raleigh are output to the new data set RALEIGH.MEMBERS:

```sas
data raleigh.members;
  set nc.members;
```
if city='Raleigh';
run;

Example 4: Merging a Single Observation with All Observations in a SAS Data Set
An observation to be merged into an existing data set can be one that is created by a SAS procedure or another DATA step. In this example, the data set AVGSALES has only one observation:

data national;
  if _n_=1 then set avgsales;
  set totsales;
run;

Example 5: Reading from the Same Data Set More Than Once
In this example, SAS treats each SET statement independently. That is, it reads from one data set as if it were reading from two separate data sets:

data drugxyz;
  set trial5(keep=sample);
  if sample>2;
  set trial5;
run;

For each iteration of the DATA step, the first SET statement reads one observation. The next time the first SET statement is executed, it reads the next observation. Each SET statement can read different observations with the same iteration of the DATA step.

Example 6: Combining One Observation with Many
You can subset observations from one data set and combine them with observations from another data set by using direct access methods, as follows:

data south;
  set revenue;
  if region=4;
  set expense point=_n_;
run;

Example 7: Performing a Table Lookup
This example illustrates using the KEY= option to perform a table lookup. The DATA step reads a primary data set that is named INVTORY and a lookup data set that is named PARTCODE. It uses the index PARTNO to read PARTCODE nonsequentially, by looking for a match between the PARTNO value in each data set. The purpose is to obtain the appropriate description, which is available only in the variable DESC in the lookup data set, for each part that is listed in the primary data set:

data combine;
  set invtory(keep=partno instock price);
  set partcode(keep=partno desc) key=partno;
run;

Example 8: Performing a Table Lookup When the Master File Contains Duplicate Observations
This example uses the KEY= option to perform a table lookup. The DATA step reads a primary data set that is named INVTORY, which is indexed on PARTNO, and a lookup data set named PARTCODE. PARTCODE contains quantities of new stock (variable
NEW_STK). The UNIQUE option ensures that, if there are any duplicate observations in INVTORY, values of NEW_STK are added only to the first observation of the group:

```sas
data combine;
  set partcode(keep=partno new_stk);
  set invtory(keep=partno instock price)
    key=partno/unique;
  instock=instock+new_stk;
run;
```

**Example 9: Reading a Subset By Using Direct Access**

These statements select a subset of 50 observations from the data set DRUGTEST by using the POINT= option to access observations directly by number:

```sas
data sample;
  do obsnum=1 to 100 by 2;
    set drugtest point=obsnum;
    if _error_ then abort;
    output;
  end;
  stop;
run;
```

**Example 10: Performing a Function until the Last Observation Is Reached**

These statements use NOBS= to set the termination value for DO-loop processing. The value of the temporary variable LAST is the sum of the observations in SURVEY1 and SURVEY2:

```sas
do obsnum=1 to last by 100;
  set survey1 survey2 point=obsnum nobs=last;
  output;
end;
stop;
```

**Example 11: Writing an Observation Only After All Observations Have Been Read**

This example uses the END= variable LAST to tell SAS to assign a value to the variable REVENUE and write an observation only after the last observation of RENTAL has been read:

```sas
set rental end=last;
totdays + days;
if last then
do;
  revenue=totdays*65.78;
  output;
end;
```

**Example 12: Retrieving the Name of the Data Set from Which the Current Observation Is Read**

This example creates three data sets and stores the data set name in a variable named dsn. The name is split into three parts and the example prints out the results.

```sas
/* Create some data sets to read */
data gas_price_option; value=395; run;
```
data gas_rbid_option; value=840; run;
data gas_price_forward; value=275; run;
/* Create a data set D */
data d;
   set gas_price_option gas_rbid_option gas_price_forward indsn=dsn;
   /* split the data set names into 3 parts */
   commodity = scan (dsn, 2, "._");
   type = scan (dsn, 3, "._");
   instrument = scan (dsn, 4, "._");
   run;
proc print data=d;
run;

Output 2.30  Data Set Name Split into Three Parts

<table>
<thead>
<tr>
<th>Obs</th>
<th>value</th>
<th>commodity</th>
<th>type</th>
<th>instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>395</td>
<td>GAS</td>
<td>PRICE</td>
<td>OPTION</td>
</tr>
<tr>
<td>2</td>
<td>840</td>
<td>GAS</td>
<td>RBID</td>
<td>OPTION</td>
</tr>
<tr>
<td>3</td>
<td>275</td>
<td>GAS</td>
<td>PRICE</td>
<td>FORWARD</td>
</tr>
</tbody>
</table>

Example 13: Using Data Set Lists
This example uses a numbered range list to input the data sets.

data dept008; emp=13; run;
data dept009; emp=9; run;
data dept010; emp=4; run;
data dept011; emp=33; run;
data _null_
   set dept008-dept010;
   put _all_
run;

The following lines are written to the SAS log.
Using a Data Set List with the SET Statement

In addition, you could use data set lists to find missing data sets. This example uses a numbered range list to locate the missing data sets. An error occurs for each data set that does not exist. Once you know which data sets are missing, you can correct the SET statement to reflect the data sets that actually exist.

data dept008; emp=13; run;
data dept009; emp=9; run;
data dept011; emp=4; run;
data dept014; emp=33; run;
data _null_
   set dept008-dept014;
   put _all_
run;

The following lines are written to the SAS log.
Example 14: Finding the Current Observation Number

The following example uses the CUROBS option to return the number of the current observation.

data women;
  set sashelp.class curobs=cobs;
  where sex = 'F';
  orig_obs = cobs;
run;

Example 15: Using the KEYRESET Option

This example uses the KEYRESET= option to look up all the values when I=3 two times.

data a(index=(i));
  do i = 1,2,3,3,3,4,5;
    j=ranuni(4);
    output;
  end;
run;
data _null_;  
  input i;
  reset = 1;
do while (_iorc_ = 0);
    set a key=i keyreset=reset;
    put _all_;
end;
_error_ = 0; _iorc_ = 0;
datalines;
3
3
;

See Also

•  Combining and Modifying SAS Data Sets: Examples
•  “Definition of Data Set Options” in SAS Data Set Options: Reference
•  “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts
•  “Rules for Words and Names in the SAS Language” in SAS Language Reference: Concepts
•  SAS Macro Language: Reference

Statements:

•  “BY Statement” on page 31
•  “DO Statement” on page 61
•  “INPUT Statement” on page 232
•  “MERGE Statement” on page 304
•  “STOP Statement” on page 409
•  “UPDATE Statement” on page 422

---

**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 335

System Options:
•  “LINESIZE= System Option” in SAS System Options: Reference
•  “PAGESIZE= System Option” in SAS System Options: Reference

STOP Statement
Stops execution of the current DATA step.

Valid in:  DATA step
Category:  Action
Type:  Executable

Syntax
STOP;

Without Arguments
The STOP statement causes SAS to stop processing the current DATA step immediately and resume processing statements after the end of the current DATA step.

Details
SAS outputs a data set for the current DATA step. However, the observation being processed when STOP executes is not added. The STOP statement can be used alone or in an IF-THEN statement or SELECT group.

Use STOP with any features that read SAS data sets using random access methods, such as the POINT= option in the SET statement. Because SAS does not detect an end-of-file with this access method, you must include program statements to prevent continuous processing of the DATA step.

Comparisons
•  When you use a windowing environment or other interactive methods of operation, the ABORT statement and the STOP statement both stop processing. The ABORT statement sets the value of the automatic variable _ERROR_ to 1, but the STOP statement does not.
• In batch or noninteractive mode, the two statements also have different effects. Use the STOP statement in batch or noninteractive mode to continue processing with the next DATA or PROC step.

Examples

Example 1: Basic Usage
• stop;
• if idcode=9999 then stop;
• select (a);
  when (0) output;
  otherwise stop;
end;

Example 2: Avoiding an Infinite Loop
This example shows how to use STOP to avoid an infinite loop within a DATA step when you are using random access methods:

data sample;
  do sampleobs=1 to totalobs by 10;
    set master.research point=sampleobs nobs=totalobs;
    output;
  end;
  stop;
run;

See Also

Statements:
• “ABORT Statement” on page 14
• POINT= option in the SET statement on page 398

Sum Statement

Adds the result of an expression to an accumulator variable.

Valid in: DATA step
Category: Action
Type: Executable

Syntax

```
variable+expression;
```

Arguments

variable
  specifies the name of the accumulator variable, which contains a numeric value.
Tips
The variable is automatically set to 0 before SAS reads the first observation. The variable's value is retained from one iteration to the next, as if it had appeared in a RETAIN statement.

To initialize a sum variable to a value other than 0, include it in a RETAIN statement with an initial value.

expression
is any SAS expression.

Tips
The expression is evaluated and the result added to the accumulator variable.

SAS treats an expression that produces a missing value as zero.

Comparisons
The sum statement is equivalent to using the SUM function and the RETAIN statement, as shown here:

```
retain variable 0;
variable=sum(variable,expression);
```

Example: Using the Sum Statement
Here are examples of sum statements that illustrate various expressions:

- `balance+(-debit);`
- `sumxsq+x*x;`
- `nx+(x ne .);`
- `if status='ready' then OK+1;`

See Also

Functions:
- “SUM Function” in SAS Functions and CALL Routines: Reference

Statements:
- “RETAIN Statement” on page 378

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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>ODS destination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTML, RTF, PRINTER</td>
</tr>
</tbody>
</table>

**COLOR=color**
specifies the title text color.

<table>
<thead>
<tr>
<th>Alias</th>
<th>ODS destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>HTML, RTF, PRINTER</td>
</tr>
</tbody>
</table>

**Example**
“Example 3: Customizing Titles and Footnotes By Using the Output Delivery System” on page 417

**BCOLOR=color**
specifies the background color of the title block.

<table>
<thead>
<tr>
<th>Alias</th>
<th>ODS destination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTML, RTF, PRINTER</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>ODS destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>HTML, RTF, PRINTER</td>
</tr>
</tbody>
</table>

**HEIGHT=dimension | size**
specifies 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.
<table>
<thead>
<tr>
<th>Range</th>
<th>1 to 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>H</td>
</tr>
<tr>
<td>ODS destination</td>
<td>HTML, RTF, PRINTER</td>
</tr>
<tr>
<td>Example</td>
<td>“Example 3: Customizing Titles and Footnotes By Using the Output Delivery System” on page 417</td>
</tr>
</tbody>
</table>

**ITALIC**

specifies that the title 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 417 |

**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 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 “Concepts: Styles and the TEMPLATE Procedure” 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 the text by the option.

For example, this code makes the title “Red, White, and Blue” appear in different colors.

```plaintext
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 (#BYVALn), BY variable names (#BYVARn), 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.

#BYVALn | #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 the following:

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\( (\text{YEAR}) \) specifies the BY variable, YEAR.

\#BYVAR\( n \) | \#BYVAR\( \text{variable-name} \)

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 the following:

\( n \)

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  \#BYVAR\( 2 \) specifies the second variable in the BY statement.

\text{variable-name}

names the BY variable.

Tip  \text{variable-name} is not case sensitive.

Example  \#BYVAR\( (\text{SITES}) \) 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 it 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.
Details
A TITLE statement takes effect when the step or RUN group with which it is associated executes. Once 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
The following examples show how you can use the TITLE statement:

• This statement suppresses a title on line n and all lines after it:
  title n;
• 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. The following examples show how to use #BYVALn, #BYVARn, 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. The following example shows you how to use PROC TEMPLATE to change the color, justification, and size of the text for the title and footnote.

/*********************************************
*The following program creates the data set*
*grain_production and the $cntry format.    *
*********************************************/

See For more information about including quotation marks as part of the title, see “Expressions” in SAS Language Reference: Concepts.
data grain_production;
    length Year $ 4 Country $ 3 Type $ 5;
    input Year country $ type $ Kilotons;
    datalines;
    1995 BRZ Wheat 1516
    1995 BRZ Rice 11236
    1995 BRZ Corn 36276
    1995 CHN Wheat 102207
    1995 CHN Rice 185226
    1995 CHN Corn 112331
    1995 IND Wheat 63007
    1995 IND Rice 122372
    1995 IND Corn 9800
    1995 INS Wheat .
    1995 INS Rice 49860
    1995 INS Corn 8223
    1995 USA Wheat 59494
    1995 USA Rice 7888
    1995 USA Corn 187300
    2010 BRZ Wheat 3302
    2010 BRZ Rice 10035
    2010 BRZ Corn 31975
    2010 CHN Wheat 109000
    2010 CHN Rice 190100
    2010 CHN Corn 119350
    2010 IND Wheat 62620
    2010 IND Rice 120012
    2010 IND Corn 8660
    2010 INS Wheat .
    2010 INS Rice 51165
    2010 INS Corn 8925
    2010 USA Wheat 62099
    2010 USA Rice 7771
    2010 USA Corn 236064
    run;
    proc format;
        value $cntry 'BRZ'='Brazil'
              'CHN'='China'
              'IND'='India'
              'INS'='Indonesia'
              'USA'='United States';
    run;
/*****************************************************/
*This PROC TEMPLATE step creates the  *
*table definition TABLE1 that is used  *
in the DATA step.  *
*****************************************************/
proc template;
    define table table1;
        mvar sysdate9;
        dynamic colhd;
        classlevels=on;
        define column char_var;
            generic=on;
        run;
    run;
    proc format;
        value $cntry 'BRZ'='Brazil'
              'CHN'='China'
              'IND'='India'
              'INS'='Indonesia'
              'USA'='United States';
        run;
blank_dups=on;
header=colhd;
styled=cellcontents;
end;

define column num_var;
generic=on;
header=colhd;
styled=cellcontents;
end;

define footer table_footer;
end;
end;
run;

/******************************************************************************
*The ODS HTML statement creates HTML output created with                 *
*the style definition D3D.                                               *
*                                                                     *
*The TITLE statement specifies the text for the first title          *
*and the attributes that ODS uses to modify it.                      *
*The J= style attribute left-justifies the title.                     *
*The COLOR= style attributes change the color of the title text       *
"Leading Grain" to blue and "Producers in" to green.                  *
*                                                                     *
*The TITLE2 statement specifies the text for the second title         *
*and the attributes that ODS uses to modify it.                       *
*The J= style attribute center justifies the title.                   *
*The COLOR= attribute changes the color of the title text "2010"      *
*to red.                                                            *
* The HEIGHT= attributes change the size of each                      *
*individual number in "2010".                                       *
*                                                                     *
*The FOOTNOTE statement specifies the text for the first footnote     *
*and the attributes that ODS uses to modify it.                      *
*The J=left style attribute left-justifies the footnote.             *
*The HEIGHT=20 style attribute changes the font size to 20pt.         *
*The COLOR= style attributes change the color of the footnote text    *
"Prepared" to red and "on" to green.                                 *
*                                                                     *
*The FOOTNOTE2 statement specifies the text for the second footnote   *
*and the attributes that ODS uses to modify it.                       *
*The J= style attribute centers the footnote.                        *
*The COLOR= attribute changes the color of the date                  *
*to blue,                                                            *
*The HEIGHT= attribute changes the font size                         *
*of the date specified by the sysdate9 macro.                        *
***************************************************************************/
ods html body='newstyle-body.htm'
    style=d3d;

    title j=left
        font= 'Times New Roman' color=blue bcolor=red "Leading Grain "
color=green bold italic "Producers in";
title2 j=center color=red underlin=1
/*This step uses the DATA step and ODS to produce an HTML report. It uses the default table definition (template) for the DATA step and writes an output object to the HTML destination.  
 ***********************************************************/
data _null_
  set grain_production;
  where type  in ('Rice', 'Corn') and year=1996;
  file print ods=(
    template='table1'
    columns=(
      char_var=country(generic=on format=$cntry.
        dynamic=(colhd='Country'))
      char_var=type(generic dynamic=(colhd='Year'))
      num_var=kilotons(generic=on format=comma12.
        dynamic=(colhd='Kilotons'))
    )
  );
  put _ods_;
run;
**Leading Grain Producers in 1996**

<table>
<thead>
<tr>
<th>Country</th>
<th>Year</th>
<th>Kilotons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>Rice</td>
<td>10,035</td>
</tr>
<tr>
<td></td>
<td>Corn</td>
<td>31,975</td>
</tr>
<tr>
<td>China</td>
<td>Rice</td>
<td>190,100</td>
</tr>
<tr>
<td></td>
<td>Corn</td>
<td>119,350</td>
</tr>
<tr>
<td>India</td>
<td>Rice</td>
<td>120,012</td>
</tr>
<tr>
<td></td>
<td>Corn</td>
<td>8,660</td>
</tr>
<tr>
<td>Indonesia</td>
<td>Rice</td>
<td>51,165</td>
</tr>
<tr>
<td></td>
<td>Corn</td>
<td>8,925</td>
</tr>
<tr>
<td>United States</td>
<td>Rice</td>
<td>7,771</td>
</tr>
<tr>
<td></td>
<td>Corn</td>
<td>236,064</td>
</tr>
</tbody>
</table>

**Prepared on 07DEC2010**

**See Also**
- “TEMPLATE Procedure: Overview” in *SAS Output Delivery System: Procedures Guide*

**Statements:**
- “FOOTNOTE Statement” on page 178

**System Options:**
- “LINESIZE= System Option” in *SAS System Options: Reference*
UPDATE Statement

Updates a master file by applying transactions.

**Valid in:** DATA step

**Category:** File-Handling

**Type:** Executable

**Note:** The values read using the UPDATE statement are retained in the PDV. For more information, see “Overview of DATA Step Processing” in SAS Language Reference: Concepts and the “RETAIN Statement” on page 378.

**CAUTION:** If you add an OUTPUT statement when using an UPDATE statement, the results that are generated are predictable but can be undesired.

### Syntax

```
  <END=variable>
  <UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;
BY by-variable;
```

### Arguments

- **master-data-set**
  
  specifies the SAS data set used as the master file.

  **Range**
  The name can be a one-level name (for example, FITNESS), a two-level name (for example, IN.FITNESS), or one of the special SAS data set names.

  **Tip**
  Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

  **See**
  “Rules for Words and Names in the SAS Language” in SAS Language Reference: Concepts

- **(data-set-options)**
  
  specifies actions SAS is to take when it reads variables into the DATA step for processing.

  **Requirement**
  Data-set-options must appear within parentheses and follow a SAS data set name.

  **Tip**
  Dropping, keeping, and renaming variables is often useful when you update a data set. Renaming like-named variables prevents the second value that is read from over-writing the first one. By renaming one variable, you make the values of both of them available for processing, such as comparing.

  **See**
  A list of data set options to use with input data sets in SAS Data Set Options: Reference
Example 2: Updating By Renaming Variables

`transaction-data-set` specifies the SAS data set that contains the changes to be applied to the master data set.

**Range**
The name can be a one-level name (for example, `HEALTH`), a two-level name (for example, `IN.HEALTH`), or one of the special SAS data set names.

**Tip** Instead of using a data set name, you can specify the physical pathname to the file, using syntax that your operating system understands. The pathname must be enclosed in single or double quotation marks.

`END=variable` creates and names a temporary variable that contains an end-of-file indicator. This variable is initialized to 0 and is set to 1 when UPDATE processes the last observation. This variable is not added to any data set.

`UPDATEMODE=MISSINGCHECK` `UPDATEMODE=NOMISSINGCHECK` specifies whether missing variable values in a transaction data set are to be allowed to replace existing variable values in a master data set.

**MISSINGCHECK**
prevents missing variable values in a transaction data set from replacing values in a master data set.

**NOMISSINGCHECK**
allows missing variable values in a transaction data set to replace values in a master data set.

**Default** `MISSINGCHECK`

**Tip** Special missing values, however, are the exception and will replace values in the master data set even when `MISSINGCHECK` (the default) is in effect.

**Details**

**Requirements**

- The UPDATE statement must be accompanied by a BY statement that specifies the variables by which observations are matched.

- The BY statement should immediately follow the UPDATE statement to which it applies.

- The data sets listed in the UPDATE statement must be sorted by the values of the variables listed in the BY statement, or they must have an appropriate index.

- Each observation in the master data set should have a unique value of the BY variable or BY variables. If there are multiple values for the BY variable, only the first observation with that value is updated. The transaction data set can contain more than one observation with the same BY value. (Multiple transaction observations are all applied to the master observation before it is written to the output file.)

For more information, see “How to Prepare Your Data Sets” in SAS Language Reference: Concepts.
**Transaction Data Sets**

Usually, the master data set and the transaction data set contain the same variables. However, to reduce processing time, you can create a transaction data set that contains only those variables that are being updated. The transaction data set can also contain new variables to be added to the output data set.

The output data set contains one observation for each observation in the master data set. If any transaction observations do not match master observations, they become new observations in the output data set. Observations that are not to be updated can be omitted from the transaction data set. See “Reading, Combining, and Modifying SAS Data Sets” in *SAS Language Reference: Concepts*.

**Missing Values**

By default the UPDATEMODE=MISSINGCHECK option is in effect, so missing values in the transaction data set do not replace existing values in the master data set. Therefore, if you want to update some but not all variables and if the variables that you want to update differ from one observation to the next, set to missing those variables that are not changing. If you want missing values in the transaction data set to replace existing values in the master data set, use UPDATEMODE=NOMISSINGCHECK.

Even when UPDATEMODE=MISSINGCHECK is in effect, you can replace existing values with missing values by using special missing value characters in the transaction data set. To create the transaction data set, use the MISSING statement in the DATA step. If you define one of the special missing values A through Z for the transaction data set, SAS updates numeric variables in the master data set to that value.

If you want the resulting value in the master data set to be a regular missing value, use a single underscore (_) to represent missing values in the transaction data set. The resulting value in the master data set will be a period (.) for missing numeric values and a blank for missing character values.

For more information about defining and using special missing value characters, see the “MISSING Statement” on page 308.

**Comparisons**

- Both UPDATE and MERGE can update observations in a SAS data set.
- MERGE automatically replaces existing values in the first data set with missing values in the second data set. UPDATE, however, does not do so by default. To cause UPDATE to overwrite existing values in the master data set with missing ones in the transaction data set, you must use UPDATEMODE=NOMISSINGCHECK.
- UPDATE changes or updates the values of selected observations in a master file by applying transactions. UPDATE can also add new observations.

**Examples**

**Example 1: Basic Updating**

These program statements create a new data set (OHIO.QTR1) by applying transactions to a master data set (OHIO.JAN). The BY variable STORE must appear in both OHIO.JAN and OHIO.WEEK4, and its values in the master data set should be unique:

```sas
data ohio.qtr1;
  update ohio.jan ohio.week4;
  by store;
run;
```

Example 2: Updating By Renaming Variables

This example shows renaming a variable in the FITNESS data set so that it will not overwrite the value of the same variable in the program data vector. Also, the WEIGHT variable is renamed in each data set and a new WEIGHT variable is calculated. The master data set and the transaction data set are listed before the code that performs the update:

Master Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>ID</th>
<th>NAME</th>
<th>TEAM</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1114</td>
<td>sally</td>
<td>blue</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>1441</td>
<td>sue</td>
<td>green</td>
<td>145</td>
</tr>
<tr>
<td>3</td>
<td>1750</td>
<td>joey</td>
<td>red</td>
<td>189</td>
</tr>
<tr>
<td>4</td>
<td>1994</td>
<td>mark</td>
<td>yellow</td>
<td>165</td>
</tr>
<tr>
<td>5</td>
<td>2304</td>
<td>joe</td>
<td>red</td>
<td>170</td>
</tr>
</tbody>
</table>

Transaction Data Set

<table>
<thead>
<tr>
<th>OBS</th>
<th>ID</th>
<th>NAME</th>
<th>TEAM</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1114</td>
<td>sally</td>
<td>blue</td>
<td>119</td>
</tr>
<tr>
<td>2</td>
<td>1994</td>
<td>mark</td>
<td>yellow</td>
<td>174</td>
</tr>
<tr>
<td>3</td>
<td>2304</td>
<td>joe</td>
<td>red</td>
<td>170</td>
</tr>
</tbody>
</table>

```plaintext
/* Sort both data sets by ID */
proc sort data=health;
   by id;
run;
proc sort data=fitness;
   by id;
run;
/* Update Master with Transaction */
data health2;
   length STATUS $11;
   update health(rename=(weight=ORIG) in=a)
      fitness(drop=name team in=b);
   by id ;
   if a and b then
```
do;
    CHANGE=abs(orig - weight);
    if weight<orig then status='loss';
    else if weight>orig then status='gain';
    else status='same';
end;
    else status='no weigh in';
run;
proc print data=health2;
    title 'Weekly Weigh-in Report';
run;

Output 2.32 Updating By Renaming Variables

### Weekly Weigh-in Report

<table>
<thead>
<tr>
<th>Obs</th>
<th>STATUS</th>
<th>ID</th>
<th>NAME</th>
<th>TEAM</th>
<th>ORIG</th>
<th>WEIGHT</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>loss</td>
<td>1114</td>
<td>sally</td>
<td>blue</td>
<td>125</td>
<td>119</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>no weigh in</td>
<td>1441</td>
<td>sue</td>
<td>green</td>
<td>145</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>no weigh in</td>
<td>1750</td>
<td>joey</td>
<td>red</td>
<td>189</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>gain</td>
<td>1994</td>
<td>mark</td>
<td>yellow</td>
<td>165</td>
<td>174</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>same</td>
<td>2304</td>
<td>jie</td>
<td>red</td>
<td>170</td>
<td>170</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example 3: Updating with Missing Values**

This example illustrates the DATA steps used to create a master data set PAYROLL and a transaction data set INCREASE that contains regular and special missing values. Note the following after the update is made:

- The salary for ID 1026 remains the same.
- The salary for ID 1034 is a special missing value.
- The salary for ID 1057 is a regular missing value.

```sas
/* Create the Master Data Set */
data payroll;
    input ID SALARY;
    datalines;
1011 245
1026 269
1028 374
1034 333
1057 582
;
/* Create the Transaction Data Set */
data increase;
    input ID SALARY;
    missing A _;
    datalines;
1011 376
1026 .
```
WHERE Statement

Selects observations from SAS data sets that meet a particular condition.

Valid in: DATA step and PROC step

/* Update Master with Transaction */
data newpay;
  update payroll increase;
  by id;
run;
proc print data=newpay;
  title 'Updating with Missing Values';
run;

Output 2.33  Updating with Missing Values

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1011</td>
<td>376</td>
</tr>
<tr>
<td>2</td>
<td>1026</td>
<td>269</td>
</tr>
<tr>
<td>3</td>
<td>1028</td>
<td>374</td>
</tr>
<tr>
<td>4</td>
<td>1034</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>1057</td>
<td>.</td>
</tr>
</tbody>
</table>

See Also

- “Definition of Data Set Options” in SAS Data Set Options: Reference
- “Reading, Combining, and Modifying SAS Data Sets” in SAS Language Reference: Concepts

Statements:

- “BY Statement” on page 31
- “MERGE Statement” on page 304
- “MISSING Statement” on page 308
- “MODIFY Statement” on page 309
- “SET Statement” on page 394

System Options:

- “MISSING= System Option” in SAS System Options: Reference
**Category:** Action  
**Type:** Declarative  
**Tip:** The SAS Tutorial video Filtering Data shows you how to filter data using the WHERE statement.

---

**Syntax**

```
WHERE where-expression-1
< logical-operator where-expression-n>
```

**Arguments**

*where-expression*

is an arithmetic or logical expression that generally consists of a sequence of operands and operators.

**Tips**

The operands and operators described in the next several sections are also valid for the WHERE= data set option.

You can specify multiple where-expressions.

*logical-operator*

can be AND, AND NOT, OR, or OR NOT.

---

**Details**

**The Basics**

Using the WHERE statement might improve the efficiency of your SAS programs because SAS is not required to read all observations from the input data set.

The WHERE statement cannot be executed conditionally. That is, you cannot use it as part of an IF-THEN statement.

WHERE statements can contain multiple WHERE expressions that are joined by logical operators.

**Note:** Using indexed SAS data sets can significantly improve performance when you use WHERE expressions to access a subset of the observations in a SAS data set. See “Understanding SAS Indexes” in SAS Language Reference: Concepts for a complete discussion of WHERE-expression processing with indexed data sets and a list of guidelines to consider before you index your SAS data sets.

**In DATA Steps**

The WHERE statement applies to all data sets in the preceding SET, MERGE, MODIFY, or UPDATE statement, and variables that are used in the WHERE statement must appear in all of those data sets. You cannot use the WHERE statement with the POINT= option in the SET and MODIFY statements.

You can apply OBS= and FIRSTOBS= processing to WHERE processing. For more information, see “Processing a Segment of Data That Is Conditionally Selected” in SAS Language Reference: Concepts.

You cannot use the WHERE statement to select records from an external file that contains raw data, nor can you use the WHERE statement within the same DATA step in which you read in-stream data with a DATALINES statement.
For each iteration of the DATA step, the first operation SAS performs in each execution of a SET, MERGE, MODIFY, or UPDATE statement is to determine whether the observation in the input data set meets the condition of the WHERE statement. The WHERE statement takes effect immediately after the input data set options are applied and before any other statement in the DATA step is executed. If a DATA step combines observations using a WHERE statement with a MERGE, MODIFY, or UPDATE statement, SAS selects observations from each input data set before it combines them.

**WHERE and BY in a DATA Step**

If a DATA step contains both a WHERE statement and a BY statement, the WHERE statement executes before BY groups are created. Therefore, BY groups reflect groups of observations in the subset of observations that are selected by the WHERE statement, not the actual BY groups of observations in the original input data set.

For a complete discussion of BY-group processing, see “By-Group Processing in SAS Programs” in *SAS Language Reference: Concepts*.

**In PROC Steps**

You can use the WHERE statement with any SAS procedure that reads a SAS data set. The WHERE statement is useful in order to subset the original data set for processing by the procedure. The *Base SAS Procedures Guide* documents the action of the WHERE statement only in those procedures for which you can specify more than one data set. In all other cases, the WHERE statement performs as documented here.

**Use of Indexes**

A DATA or PROC step attempts to use an available index to optimize the selection of data when an indexed variable is used in combination with one of the following operators and functions:

- the BETWEEN-AND operator
- the comparison operators, with or without the colon modifier
- the CONTAINS operator
- the IS NULL and IS NOT NULL operators
- the LIKE operator
- the TRIM function
- the SUBSTR function, in some cases.

SUBSTR requires the following arguments:

```plaintext
where substr(variable, position, length) = 'character-string';
```

An index is used in processing when the arguments of the SUBSTR function meet all of the following conditions:

- `position` is equal to 1
- `length` is less than or equal to the length of `variable`
- `length` is equal to the length of `character-string`.

**Operands Used in WHERE Expressions**

Operands in WHERE expressions can contain the following values:

- constants
• time and date values
• values of variables that are obtained from the SAS data sets
• values created within the WHERE expression itself.

You cannot use variables that are created within the DATA step (for example, FIRST:variable, LAST:variable, _N_, or variables that are created in assignment statements) in a WHERE expression because the WHERE statement is executed before the SAS System brings observations into the DATA or PROC step. When WHERE expressions contain comparisons, the unformatted values of variables are compared.

The following are examples of using operands in WHERE expressions:

- where score>50;
- where date>='01jan1999'd and time>='9:00't;
- where state='Mississippi';

As in other SAS expressions, the names of numeric variables can stand alone. SAS treats values of 0 or missing as false; other values are true. These examples are WHERE expressions that contain the numeric variables EMPNUM and SSN:

- where empnum;
- where empnum and ssn;

Character literals or the names of character variables can also stand alone in WHERE expressions. If you use the name of a character variable by itself as a WHERE expression, SAS selects observations where the value of the character variable is not blank.

**Operators Used in the WHERE Expression**

You can include both SAS operators and special WHERE-expression operators in the WHERE statement. For a complete list of the operators, see Table 2.10 on page 430. For the rules that SAS follows when it evaluates WHERE expressions, see “WHERE-Expression Processing” in SAS Language Reference: Concepts.

The following table lists the operators for the WHERE statement.

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Symbol or Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td></td>
<td>−</td>
<td>subtraction</td>
</tr>
<tr>
<td></td>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>Comparison</td>
<td>= or EQ</td>
<td>equal to</td>
</tr>
<tr>
<td>Operator Type</td>
<td>Symbol or Mnemonic</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>^=, -=, ~=, or NE*</td>
<td></td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt; or GT</td>
<td></td>
<td>greater than</td>
</tr>
<tr>
<td>&lt; or LT</td>
<td></td>
<td>less than</td>
</tr>
<tr>
<td>&gt;= or GE</td>
<td></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;= or LE</td>
<td></td>
<td>less than or equal to</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>equal to one of a list</td>
</tr>
</tbody>
</table>

Logical (Boolean)

| & or AND | rational and |
| | logical and |
| | logical or |
| | logical not |

Other

| ||*** | concatenation of character variables |
| | )    | indicate order of evaluation |
| | + prefix | positive number |
| | - prefix | negative number |

WHERE Expression

Only

| BETWEEN–AND | an inclusive range |
| ? or CONTAINS | a character string |
| IS NULL or IS MISSING | missing values |
| LIKE | match patterns |
| *= | sounds-like |
### Operator Types

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Symbol or Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAME-AND</td>
<td></td>
<td>add clauses to an existing WHERE statement without retyping original one</td>
</tr>
</tbody>
</table>

* The caret (^), tilde (~), and the not sign (¬) all indicate a logical not. Use the character available on your keyboard, or use the mnemonic equivalent.

** The OR symbol (|), broken vertical bar (| |), and exclamation point (!) all indicate a logical or. Use the character available on your keyboard, or use the mnemonic equivalent.

*** Two OR symbols (||), two broken vertical bars (| | |), or two exclamation points (!!!) indicate concatenation. Use the character available on your keyboard.

† You can use the colon modifier (:) with any of the comparison operators in order to compare only a specified prefix of a character string.

### Comparisons

- You can use the WHERE command in SAS/FSP software to subset data for editing and browsing. You can use both the WHERE statement and WHERE= data set option in windowing procedures and in conjunction with the WHERE command.

- To select observations from individual data sets when a SET, MERGE, MODIFY, or UPDATE statement specifies more than one data set, apply a WHERE= data set option to each data set. In the DATA step, if a WHERE statement and a WHERE= data set option apply to the same data set, SAS uses the data set option and ignores the statement for that data set. Other data sets without a WHERE data set option use the statement.

- The most important differences between the WHERE statement in the DATA step and the subsetting IF statement are as follows:
  - The WHERE statement selects observations before they are brought into the program data vector, making it a more efficient programming technique. The subsetting IF statement works on observations after they are read into the program data vector.
  - The WHERE statement can produce a different data set from the subsetting IF when a BY statement accompanies a SET, MERGE, or UPDATE statement. The different data set occurs because SAS creates BY groups before the subsetting IF statement selects but after the WHERE statement selects.
  - The WHERE statement cannot be executed conditionally as part of an IF statement, but the subsetting IF statement can.
  - The WHERE statement selects observations in SAS data sets only, whereas the subsetting IF statement selects observations from an existing SAS data set or from observations that are created with an INPUT statement.
  - The subsetting IF statement cannot be used in SAS windowing procedures to subset observations for browsing or editing.
  - Do not confuse the WHERE statement with the DROP or KEEP statement. The DROP and KEEP statements select variables for processing. The WHERE statement selects observations.
Examples

Example 1: Basic WHERE Statement Usage
This DATA step produces a SAS data set that contains only observations from data set CUSTOMER in which the value for NAME begins with Mac and the value for CITY is Charleston or Atlanta.

```
data testmacs;
  set customer;
  where substr(name,1,3)='Mac' and 
      (city='Charleston' or city='Atlanta');
run;
```

Example 2: Using Operators Available Only in the WHERE Statement
- Using BETWEEN-AND:
  ```
  where empnum between 500 and 1000;
  ```
- Using CONTAINS:
  ```
  where company ? 'bay';
  where company contains 'bay';
  ```
- Using IS NULL and IS MISSING:
  ```
  where name is null;
  where name is missing;
  ```
- Using LIKE to select all names that start with the letter D:
  ```
  where name like 'D%';
  ```
- Using LIKE to match patterns from a list of the following names:
  Diana
  Diane
  Dianna
  Dianthus
  Dyan

<table>
<thead>
<tr>
<th>WHERE Statement</th>
<th>Name Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>where name like 'D_an';</td>
<td>Dyan</td>
</tr>
<tr>
<td>where name like 'D_an_';</td>
<td>Diana, Diane</td>
</tr>
<tr>
<td>where name like 'D_an__';</td>
<td>Dianna</td>
</tr>
<tr>
<td>where name like 'D_an%';</td>
<td>all names from list</td>
</tr>
</tbody>
</table>

- Using the Sounds-like Operator to select names that sound like “Smith”:
  ```
  where lastname=*'Smith';
  ```
- Using SAME-AND:
  ```
  where year>1991;
  ```
...more SAS statements...
where same and year<1999;

In this example, the second WHERE statement is equivalent to the following WHERE statement:
where year>1991 and year<1999;

See Also
• Base SAS Procedures Guide
• “By-Group Processing in SAS Programs” in SAS Language Reference: Concepts
• SAS SQL Query Window User’s Guide
• SAS/IML User’s Guide
• “Understanding SAS Indexes” in SAS Language Reference: Concepts
• “WHERE-Expression Processing” in SAS Language Reference: Concepts

Data Set Options:
• “WHERE= Data Set Option” in SAS Data Set Options: Reference

Statements:
• “IF Statement, Subsetting” on page 187

WINDOW Statement
Creates customized windows for your applications.

Valid in: DATA step
Category: Window Display
Type: Declarative

Syntax
WINDOW window <window-options> field-definition(s);
WINDOW window <window-options> group-definition(s);

Arguments
window
specifies the window name.

Restriction Window names must conform to SAS naming conventions.

window-options
specifies characteristics of the window as a whole. Specify these window-options before any field or GROUP= specifications:
COLOR=color
specifies the color of the window background for operating environments that have this capability. In other operating environments, this option affects the color of the window border. The following colors are available:

BLACK   MAGENTA
BLUE     ORANGE
BROWN    PINK
CYAN     RED
GRAY     WHITE
GREEN    YELLOW

Default If you do not specify a color with the COLOR= option, the window’s background color is device-dependent instead of black, and the color of a field is device-dependent instead of white.

Tip The representation of colors might vary, depending on the monitor being used. COLOR= has no effect on monochrome monitors.

COLUMNS=columns
specifies the number of columns in the window.

Default The window fills all remaining columns on the monitor; the number of columns that are available depends on the type of monitor that is being used.

ICOLUMN=column
specifies the initial column within the monitor at which the window is displayed.

Default SAS displays the window at column 1.

IROW=row
specifies the initial row (or line) within the monitor at which the window is displayed.

Default SAS displays the window at row 1.

KEYS=<<libref.>catalog.>keys-entry
specifies the name of a KEYS entry that contains the function key definitions for the window.

Default SAS uses the current function key settings that are defined in the KEYS window.

Tips If you specify only an entry name, SAS looks in the SASUSER.PROFILE catalog for a KEYS entry of the name that is specified. You can also specify the three-level name of a KEYS entry, in the form

libref.catalog.keys-entry

To create a set of function key definitions for a window, use the KEYS window. Define the keys as you want, and use the SAVE command to save the definitions in the SASUSER.PROFILE catalog or in a SAS library and catalog that you specify.
MENU=<libref.>catalog.>pmenu-entry
  specifies the name of a menu (pmenu) you have built with the PMENU
  procedure.

  Tip  If you specify only an entry name, SAS looks in the SASUSER.PROFILE
       catalog for a PMENU entry of the name specified. You can also specify the
       three-level name of a PMENU entry in the form
       libref.catalog.pmenu-entry

ROWS=rows
  specifies the number of rows (or lines) in the window.

  Default  The window fills all remaining rows on the monitor.

  Tip  The number of rows that are available depends on the type of monitor
       that is being used.

field-definition(s)
  specifies and describes a variable or character string to be displayed in a window or
  within a group of related fields.

  Tips  A window or group can contain any number of fields, and you can define the
        same field in several groups or windows.

        You can specify multiple field-definitions.

        See  “Field Definitions” on page 437

group-definition(s)
  specifies a group and defines all fields within a group. A group definition consists of
  two parts: the GROUP= option and one or more field definitions.

GROUP=group
  specifies a group of related fields.

  Default  A window contains one unnamed group of fields.

  Restriction  group must be a SAS name.

  Tips  When you refer to a group in a DISPLAY statement, write the name
        as window:group.

        A group contains all fields in a window that you want to display at
        the same time. Display various groups of fields within the same
        window at different times by naming each group. Choose the group
        to appear by specifying window:group in the DISPLAY statement.

        Specifying several groups within a window prevents repetition of
        window options that do not change and helps you keep track of
        related displays. For example, if you are defining a window to
        check data values, arrange the display of variables and messages for
        most data values in the data set in a group that is named
        STANDARD. Arrange the display of different messages in a group
        that is named CHECKIT that appears when data values meet the
        conditions that you want to check.
Details

The Basics

Operating Environment Information

The WINDOW statement has some functionality that might be specific to your operating environment. For details, see the SAS documentation for your operating environment.

You can use the WINDOW statement in the SAS windowing environment, in interactive line mode, or in noninteractive mode to create customized windows for your applications. Windows that you create can display text and accept input; they have command and message lines. The window name appears at the top of the window. Use commands and function keys with windows that you create. A window definition remains in effect only for the DATA step that contains the WINDOW statement.

Define a window before you display it. Use the DISPLAY statement to display windows that are created with the WINDOW statement. For more information, see the “DISPLAY Statement” on page 57.

Field Definitions

Use a field definition to identify a variable or a character string to be displayed, its position, and its attributes. Enclose character strings in quotation marks. The position of an item is its beginning row (or line) and column. Attributes include color, whether you can enter a value into the field, and characteristics such as highlighting.

You can define a field to contain a variable value or a character string, but not both. The form of a field definition for a variable value is

\[<row\ column>\ variable\ <format>\ options\]

The form for a character string is

\[<row\ column>\ 'character-string'\ options\]

The elements of a field definition are described here.

row column

specifies the position of the variable or character string.

SAS keeps track of its position in the window with a pointer. For example, when you tell SAS to write a variable's value in the third column of the second row of a window, the pointer moves to row 2, column 3 to write the value. Use the pointer controls that are listed here to move the pointer to the appropriate position for a field.

In a field definition, row can be one of these row pointer controls:

\[#n\]

specifies row \( n \) within the window.

\[Range\ n\]

must be a positive integer.

\[#numeric-variable\]

specifies the row within the window that is given by the value of \( numeric-variable \).

---

1 You cannot use the WINDOW statement in batch mode because no computer is connected to a batch executing process.
Restriction  #numeric-variable must be a positive integer. If the value is not an integer, the decimal portion is truncated and only the integer is used.

#(expression)
specifies the row within the window that is given by the value of expression.

Restrictions  expression can contain array references and must evaluate to a positive integer.

Enclose expression in parentheses.

/  
moves the pointer to column 1 of the next row.

In a field definition, column can be one of these column pointer controls:

@n  
specifies column n within the window.

Restriction  n must be a positive integer.

@numeric-variable  
specifies the column within the window that is given by the value of numeric-variable.

Restriction  numeric-variable must be a positive integer. If the value is not an integer, the decimal portion is truncated and only the integer is used.

@(#expression)
specifies the column within the window that is given by the value of expression.

Restrictions  expression can contain array references and must evaluate to a positive integer.

Enclose expression in parentheses.

+n  
moves the pointer n columns.

Range  n must be a positive integer.

+n numeric-variable  
moves the pointer the number of columns that is given by the numeric-variable.

Restriction  +numeric-variable must be a positive or negative integer. If the value is not an integer, the decimal portion is truncated and only the integer is used.

Default  If you omit row in the first field of a window or group, SAS uses the first row of the window. If you omit row in a later field specification, SAS continues on the row that contains the previous field. If you omit column, SAS uses column 1 (the left border of the window).

Tip  Although you can specify either row or column first, the examples in this documentation show the row first.
**variable**

specifies a variable to be displayed or to be assigned the value that you enter at that position when the window is displayed.

**Tips**  
variable can be the name of a variable or of an array reference.

---

To allow a variable value in a field to be displayed but not changed by the user, use the PROTECT= option (described later in this section). You can also protect an entire window or group for the current execution of the DISPLAY statement by specifying the NOINPUT option in the DISPLAY statement.

If a field definition contains the name of a new variable, that variable is added to the data set that is being created (unless you use a KEEP or DROP specification).

**format**

gives the format for the variable.

**Default**

If you omit format, SAS uses an informat and format that are specified elsewhere (for example, in an ATTRIB, INFORMAT, or FORMAT statement or permanently stored with the data set) or a SAS default informat and format.

**Tips**

If a field displays a variable that cannot be changed (that is, you use the PROTECT=YES option), format can be any SAS format or a format that you define with the FORMAT procedure.

If a field can both display a variable and accept input, you must either specify the informat in an INFORMAT or ATTRIB statement or use a SAS format such as $CHAR. or TIME. that has a corresponding informat.

If a format is specified, the corresponding informat is assigned automatically to fields that can accept input.

A format and an informat in a WINDOW statement override an informat and a format that are specified elsewhere.

---

'character-string'

contains the text of a character string to be displayed.

**Restrictions**  
The character string must be enclosed in quotation marks.

You cannot enter a value in a field that contains a character string.

---

**options**

Specify field definition attributes:

**ATTR=highlighting-attribute**

controls these highlighting attributes of the field:

- **BLINK**  
  causes the field to blink.
- **HIGHLIGHT**  
  displays the field at high intensity.
- **REV_VIDEO**  
  displays the field in reverse video.
- **UNDERLINE**  
  underlines the field.

**Alias**  
A=
Tips To specify more than one highlighting attribute, use the form

\[
\text{ATTR=}(\text{highlighting-attribute-1, \ldots})
\]

The highlighting attributes that are available depend on the type of
monitor that you use.

\[\text{AUTOSKIP=YES | NO}\]
controls whether the cursor moves to the next unprotected field of the current
window or group when you have entered data in all positions of a field.

\[\begin{array}{ll}
\text{YES} & \text{specifies that the cursor moves automatically to the next unprotected field.} \\
\text{NO} & \text{specifies that the cursor does not move automatically.}
\end{array}\]

Alias AUTO=
Default NO

\[\text{COLOR=\text{color}}\]
specifies a color for the variable or character string. You can specify one of the
following colors:

\[
\begin{array}{ll}
\text{BLACK} & \text{MAGENTA} \\
\text{BLUE} & \text{ORANGE} \\
\text{BROWN} & \text{PINK} \\
\text{CYAN} & \text{RED} \\
\text{GRAY} & \text{WHITE} \\
\text{GREEN} & \text{YELLOW}
\end{array}
\]

Alias C=
Default WHITE

Tips The representation of colors might vary, depending on the monitor that
you use.

\[\text{COLOR=}\text{ has no effect on monochrome monitors.}\]

\[\text{DISPLAY=YES | NO}\]
controls whether the contents of a field are displayed.

\[\begin{array}{ll}
\text{YES} & \text{specifies that SAS displays characters in a field as you enter them in.} \\
\text{NO} & \text{specifies that the entered characters are not displayed.}
\end{array}\]

Default YES

\[\text{PERSIST=YES | NO}\]
controls whether a field is displayed by all executions of a DISPLAY statement
in the same iteration of the DATA step until the DISPLAY statement contains the
BLANK option.

\[\begin{array}{ll}
\text{YES} & \text{specifies that each execution of the DISPLAY statement displays all} \\
& \text{previously displayed contents of the field as well as the contents that} \\
& \text{are scheduled for display by the current DISPLAY statement. If the}
\end{array}\]
new contents overlap persisting contents, the persisting contents are no longer displayed.

NO specifies that each execution of a DISPLAY statement displays only the current contents of the field.

Default NO

Tip PERSIST= is most useful when the position of a field changes in each execution of a DISPLAY statement.

Example “Example 3: Persisting and Nonpersisting Fields” on page 444

PROTECT=YES | NO
controls whether information can be entered into a field.

YES specifies that you cannot enter information.

NO specifies that you can enter information.

Alias P=

Default No

Tip Use PROTECT= only for fields that contain variables; fields that contain text are automatically protected.

REQUIRED=YES | NO
controls whether a field can be left blank.

NO specifies that you can leave the field blank.

YES specifies that you must enter a value in the field.

Default NO

Tip If you try to leave a field blank that was defined with REQUIRED=YES, SAS does not allow you to input values in any subsequent fields in the window.

Automatic Variables
The WINDOW statement creates two automatic SAS variables: _CMD_ and _MSG_.

_CMD_
contains the last command from the window's command line that was not recognized by the window.

Tip _CMD_ is a character variable with a length of 80 bytes; its value is set to ‘ ‘ (blank) before each execution of a DISPLAY statement.

Example “Example 4: Sending a Message” on page 445

_MSG_
contains a message that you specify to be displayed in the message area of the window.

Tip _MSG_ is a character variable with a length of 80 bytes; its value is set to ‘ ‘ (blank) after each execution of a DISPLAY statement.
Displaying Windows

The DISPLAY statement enables you to display windows. Once you display a window, the window remains visible until you display another window over it or until the end of the DATA step. When you display a window that contains fields into which you can enter values, either enter a value or press Enter at each unprotected field to cause SAS to proceed to the next display. While a window is being displayed, you can use commands and function keys to view other windows, change the size of the current window, and so on. The execution proceeds to the next display only after you have pressed Enter in all unprotected fields.

A DATA step that contains a DISPLAY statement continues execution until

- the last observation that is read by a SET, MERGE, MODIFY, UPDATE, or INPUT statement has been processed
- a STOP or ABORT statement is executed
- an END command executes.

Comparisons

- The WINDOW statement creates a window, and the DISPLAY statement displays it.
- The %WINDOW and %DISPLAY statements in the macro language create and display windows that are controlled by the macro facility.

Examples

Example 1: Creating a Single Window

This DATA step creates a window with a single group of fields:

data _null_; 
  window start
    #9 @26 'WELCOME TO THE SAS SYSTEM'
    color=black
    #12 @19 'THIS PROGRAM CREATES'
    #12 @40 'TWO SAS DATA SETS'
    #14 @26 'AND USES THREE PROCEDURES'
    #18 @27 'Press ENTER to continue';
  display start;
  stop;
run;
The START window fills the entire monitor. The first line of text is black. The other three lines are the default for your operating environment. The text begins in the column that you specified in your program. The START window does not require you to input any values. However, to exit the window do one of the following:

- Press Enter to cause DATA step execution to proceed to the STOP statement.
- Issue the END command.

If you omit the STOP statement from this program, the DATA step executes endlessly until you execute END from the window, either with a function key or from the command line. (Because this DATA step does not read any observations, SAS cannot detect an end-of-file to end DATA step execution.)

**Example 2: Displaying Two Windows Simultaneously**

The following statements assign news articles to reporters. The list of article topics is stored as variable art in SAS data set category.article. This application enables you to assign each topic to a writer and to view the accumulating assignments. The program creates a new SAS data set named Assignment.

```sas
libname category 'SAS-library';
data Assignment;
    set category.article end=final;
    drop a b j s o;
    window Assignment irow=1 rows=12 color=white
        #3 @10 'Article:' +1 art protect=yes
        'Name:' +1 name $14.;
    window Showtotal irow=20 rows=12 color=white
        group=subtotal
        #1 @10 'Adams has' +1 a
        #2 @10 'Brown has' +1 b
        #3 @10 'Johnson has' +1 j
        #4 @10 'Smith has' +1 s
        #5 @10 'Other has' +1 o
        group=lastmessage
        #8 @10
        'ALL ARTICLES ASSIGNED.
Press ENTER to stop processing.';
    display Assignment blank;
    if name='Adams' then a+1;
    else if name='Brown' then b+1;
```
When you execute the DATA step, the following windows appear.

In the Assignment window (located at the top of the monitor), you see the name of the article and a field into which you enter a reporter's name. After you enter a name and press Enter, SAS displays the Showtotal window (located at the bottom of the monitor) which shows the number of articles that are assigned to each reporter (including the assignment that you just made). As you continue to make assignments, the values in the Showtotal window are updated. During the last iteration of the DATA step, SAS displays the message that all articles are assigned, and instructs you to press Enter to stop processing.

**Example 3: Persisting and Nonpersisting Fields**

This example demonstrates the PERSIST= option. You move from one window to the other by positioning the cursor in the current window and pressing Enter.

```sas
data _null_;  
array row{3} r1-r3;  
array col{3} c1-c3;  
input row{*} col{*};  
window One  
   rows=20 columns=36  
   #1 @14 'PERSIST=YES' color=black  
   @(row{i}) @(col{i}) 'Hello'  
   color=black persist=yes;  
window Two  
   icolumn=43 rows=20 columns=36  
   #1 @14 'PERSIST=NO' color=black  
   @(row{i}) @(col{i}) 'Hello'  
   color=black persist=no;  
  do i=1 to 3;  
    display One;  
    display Two;  
  end;
```

444  Chapter 2 • Dictionary of SAS Statements
The following windows show the results of this DATA step after its third iteration.

![Windows showing results](image)

Note that window One shows **Hello** in all three positions in which it was displayed. Window Two shows only the third and final position in which **Hello** was displayed.

**Example 4: Sending a Message**

This example uses the _CMD_ and _MSG_ automatic variables to send a message when you execute an erroneous windowing command in a window that is defined with the WINDOW statement:

```plaintext
if _cmd_ ne ' ' then
   _msg_='CAUTION: UNRECOGNIZED COMMAND' || _cmd_;
```

When you enter a command that contains an error, SAS sets the value of _CMD_ to the text of the erroneous command. Because the value of _CMD_ is no longer blank, the IF statement is true. The THEN statement assigns to _MSG_ the value that is created by concatenating **CAUTION: UNRECOGNIZED COMMAND** and the value of _CMD_ (up to a total of 80 bytes). The next time a DISPLAY statement displays that window, the message line of the window displays the following:

**CAUTION: UNRECOGNIZED COMMAND**

`Command` is the erroneous windowing command.

**Example 5: Creating a SAS Data Set**

The following statements create a SAS data set by using input from the WINDOW statement.

```plaintext
data new;
   length name $20;
   window start
      #3 'Type the variable name'
      #4 'and press the Enter key.'
      #7 'Name:' +1 name attr=underline
      #11 'When you are finished entering variable names, type "end"'
      #12 'at the command line.';
   display start;
   run;
proc print;
run;
```
X Statement

Issues an operating-environment command from within a SAS session.

Valid in: Anywhere  
Category: Operating Environment  
See: X Statement under Windows, UNIX, or z/OS

Syntax

\[
\text{X} <\text{operating-environment-command}>;
\]

Without Arguments

Using X without arguments places you in your operating environment, where you can issue commands that are specific to your environment.

Arguments

\[\text{operating-environment-command}\]

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. See the SAS Macro Language: Reference for information.

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
Chapter 3
Dictionary of SAS 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.
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.

The following table includes examples of defining the SAS_FTP_AUTHTLS environment variable.

<table>
<thead>
<tr>
<th>Method</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS configuration file</td>
<td>-set SAS_FTP_AUTHTLS 1</td>
</tr>
<tr>
<td>SAS invocation</td>
<td>-set SAS_FTP_AUTHTLS 1</td>
</tr>
<tr>
<td>OPTIONS statement</td>
<td>options set=SAS_FTP_AUTHTLS 1;</td>
</tr>
</tbody>
</table>

For more information about TLS, see *Encryption in SAS*

**See Also**

**Statements:**

- “FILENAME Statement, FTP Access Method” on page 126
Recommended Reading

Here is the recommended reading list for this title:

• Base SAS Procedures Guide
• Base SAS Utilities: Reference
• Learning SAS by Example: A Programmer's Guide
• Output Deliver System: The Basics and Beyond
• SAS Companion for UNIX Environments
• SAS Companion for Windows
• SAS Companion for z/OS
• SAS Component Objects: Reference
• SAS Data Set Options: Reference
• SAS Formats and Informats: Reference
• SAS Functions and CALL Routines: Reference
• SAS Functions by Example
• SAS Language Interfaces to Metadata
• SAS Language Reference: Concepts
• SAS ODS Graphics: Procedures Guide
• SAS Output Delivery System: User's Guide
• Base SAS Procedures Guide
• SAS Scalable Performance Data Engine: Reference
• SAS System Options: Reference
• SAS XML LIBNAME Engine: User's Guide
• Step-by-Step Programming with Base SAS 9.4
• The Little SAS Book: A Primer, Fifth Edition

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

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books
## Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_ALL_</code></td>
<td>Argument</td>
</tr>
<tr>
<td><code>FILENAME</code></td>
<td>Statement 97</td>
</tr>
<tr>
<td><code>LIBNAME</code></td>
<td>Statement 276</td>
</tr>
<tr>
<td><code>_ALL_ CLEAR</code></td>
<td>Option</td>
</tr>
<tr>
<td><code>CATNAME</code></td>
<td>Statement 38</td>
</tr>
<tr>
<td><code>_ALL_ LIST</code></td>
<td>Option 38</td>
</tr>
<tr>
<td><code>BLANKPAGE_</code></td>
<td>Option, PUT statement 335</td>
</tr>
<tr>
<td><code>_CMD_</code></td>
<td>Automatic variable 441</td>
</tr>
<tr>
<td><code>CMD</code></td>
<td>Automatic variable 441</td>
</tr>
<tr>
<td><code>_CMD_ SAS</code></td>
<td>Variable, WINDOW statement 441</td>
</tr>
<tr>
<td><code>_ERROR_</code></td>
<td>Variable 71</td>
</tr>
<tr>
<td><code>FILE_</code></td>
<td>73</td>
</tr>
<tr>
<td><code>FILE</code></td>
<td>Variable 73</td>
</tr>
<tr>
<td><code>FILE_</code></td>
<td>Option 73</td>
</tr>
<tr>
<td><code>FILE=</code></td>
<td>Option 73</td>
</tr>
<tr>
<td><code>FILE</code></td>
<td>Statement 73</td>
</tr>
<tr>
<td><code>INFILE_</code></td>
<td>Automatic variable 198</td>
</tr>
<tr>
<td><code>INFILE_</code></td>
<td>Option 198</td>
</tr>
<tr>
<td><code>INFILE=</code></td>
<td>Option 198</td>
</tr>
<tr>
<td><code>INFILE</code></td>
<td>Statement 198, 226</td>
</tr>
<tr>
<td><code>_IORC_</code></td>
<td>Automatic variable 226</td>
</tr>
<tr>
<td><code>_IORC_</code></td>
<td>MODIFY statement and 315</td>
</tr>
<tr>
<td><code>_MSG_</code></td>
<td>434, 441</td>
</tr>
<tr>
<td><code>_MSG_</code></td>
<td>Automatic variable 441</td>
</tr>
<tr>
<td><code>_MSG_</code></td>
<td>SAS variable, WINDOW statement 441</td>
</tr>
<tr>
<td><code>_NULL_</code></td>
<td>Argument 44</td>
</tr>
<tr>
<td><code>DATA</code></td>
<td>Statement 44</td>
</tr>
<tr>
<td><code>_PAGE_</code></td>
<td>Option, PUT statement 335</td>
</tr>
<tr>
<td><code>;</code></td>
<td>(semicolon), in data lines 37, 54</td>
</tr>
<tr>
<td><code>:</code></td>
<td>(colon) format modifier 255</td>
</tr>
<tr>
<td><code>~</code></td>
<td>(tilde) format modifier, definition 258</td>
</tr>
</tbody>
</table>

## A

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ABEND</code></td>
<td></td>
</tr>
<tr>
<td><code>ABORT</code></td>
<td>14</td>
</tr>
<tr>
<td><code>ABORT</code></td>
<td>14 arguments 14 compared to STOP statement 409 comparisons 17 details 17 examples 17</td>
</tr>
</tbody>
</table>
without arguments 14
ACCESS= option
LIBNAME statement 276
ACCESS=READONLY option
CATNAME statement 38
aggregate storage location
filerefs for 100
ALTER passwords 48
ampersand (&) format modifier 255, 258
anonymous FTP login 138
array reference 23
array reference statement 23
array reference, explicit 23
compared to ARRAY statement 21
ARRAY statement 18
compared to array reference, explicit 24
arrays
defining elements in 18
describing elements to process 23
writing to 346
assignment statement 26
at sign (@) argument
INPUT statement 233
INPUT statement, column input 248
INPUT statement, formatted input 251
INPUT statement, named input 262
PUT statement 335
PUT statement, column output 354
PUT statement, formatted output 356
PUT statement, named output 365
at sign (@) column pointer control
INPUT statement 234
PUT statement 335
WINDOW statement 438
at sign (@) line-hold specifier
PUT statement 345
PUT statement, column output 345
at signs (@@) argument
INPUT statement 233
INPUT statement, column input 248
INPUT statement, formatted input 251
INPUT statement, named input 262
PUT statement 335
PUT statement, column output 354
PUT statement, formatted output 356
PUT statement, named output 365
at signs (@@) line-hold specifier, PUT statement 345
ATTACH= option
FILENAME statement, EMAIL access method 112
attachments to email 122
ATTRIB statement 27
compared to FORMAT statement 183
compared to INFORMAT statement 230
compared to LENGTH statement 273
AUTHDOMAIN option
FILENAME statement, WebDAV access method 166
AUTHDOMAIN= option
FILENAME statement, FTP access method 127
FILENAME statement, URL access method 161
AUTHTLS option
FILENAME statement, FTP access method 127
AUTO= option, WINDOW statement 440
autocall macro libraries
accessing lowercased members 172
WebDAV location as 172
autocall macros
executing from catalogs 105
AUTOSKIP= option, WINDOW statement 440
B
batch processing
checkpoint-restart mode and 41
BATCHFILE option
FILENAME statement, SFTP access method 149
BCC= option
FILENAME statement, EMAIL access method 113
BELL argument, DISPLAY statement 57
BINARY option
FILENAME statement, FTP access method 128, 134
BLANK argument, DISPLAY statement 57
BLKSIZE= option
FILE statement 73
INFILE statement 202
BLOCKSIZE= option
FILENAME statement, FTP access method 128
FILENAME statement, SOCKET access 156
FILENAME statement, URL access method 161
BUFFERLEN= option
FILENAME statement, Hadoop access method 142
buffers, allocating
SASFILE statement 385
BY groups
identifying beginning and end of 33
processing 34
BY processing 34
with nonsorted data 34
BY statement 31
arguments 31
details 33
examples 34
in DATA step 33
in PROC steps 33
specifying sort order 34
with SAS views 33
BY values
duplicates, MODIFY statement 314
BY variables
customizing titles with 417
specifying 34
BY-group processing
SET statement for 401
BYE command, compared to ENDSAS statement 71

C
C= option, WINDOW statement 440
CALL routines
calling 36
CALL statement 36
CANCEL argument
ABORT statement 14
CARDS argument
INFILE statement 199
CARDS statement 37, 53
CARDS4 statement 37, 54
CASE statement 393
CATALOG access method
See FILENAME statement, CATALOG access method
catalog entries
%INCLUDE with 105
catalogs
%INCLUDE statement with several entries in single catalog 198
concatenating 38
concatenating, implicitly 284, 286
executing autocall macros from 105
referencing as external files 103
CATAMS entries
reading and writing 105
CATNAME statement 38
arguments 38
comparisons 39
details 39
examples 39
options 38
catrefs 38
CC= option
FILENAME statement, EMAIL access method 114
CD= option
FILENAME statement, FTP access method 128
FILENAME statement, SFTP access method 149
CFG= option
FILENAME statement, Hadoop access method 143
character data
embedded blanks in 260
CHECKPOINT EXECUTE ALWAYS statement 41
checkpoint-restart mode 41
CLEAR argument
FILENAME statement 97, 99
LIBNAME statement 276
CLEAR option
CATNAME statement 38
CLIPBOARD access method 106
COL= option
INFILE statement 202
colon (:) format modifier 255, 258
COLOR= argument, WINDOW statement 434
COLOR= option, WINDOW statement 440
column input 237, 248
column output 342, 354
column pointer controls
INPUT statement 232
PUT statement 335
COLUMN= option
FILE statement 73
INFILE statement 202
columns
two-column page format 90
COLUMNS= argument, WINDOW statement 434
coma-delimited data 259, 260
Comment statement 41
comments 41
COMPRESS= option
LIBNAME statement 278
COMPRESSION= option
FILENAME statement, ZIP access method 175
CONCAT= option
FILENAME statement, Hadoop access method 143
concatenating catalogs
CATNAME statement 38
implicitly 284, 286
logically concatenated catalogs 39
nested catalog concatenation 40
rules for 39
concatenating data libraries 284
concatenating data sets
SET statement for 401, 402
conditional logic
for sending email 123
CONNECT option
FILENAME statement, URL access method 161
CONTENT_TYPE= option
FILENAME statement, EMAIL access method 114
CONTINUE argument, DM statement 59
CONTINUE statement 43
compared to LEAVE statement 271
copied records
truncating 224
CT= option
FILENAME statement, EMAIL access method 114
CVPBYTES= option
LIBNAME statement 279
CVPENGINE= option
LIBNAME statement 279
CVPMULTIPLIER= option
LIBNAME statement 279

D
data libraries
associating librefs with 283
concatenating 284
concatenating, logically 286
disassociating librefs from 284
writing attributes to log 284
data lines
including 191
reading 37, 54
DATA naming convention 44
data set list
MERGE statement 304
SET statement 394
data set options
MODIFY statement with 318
data sets
See also DATA statement
combining 401
concatenating 401, 402
interleaving 401, 402
one-to-one reading 402, 403
permanently storing, one-level names 287
reading observations 394, 402
reading observations, more than once 403
DATA statement 44
arguments 44
creating custom reports 51
creating DATA step views 49
creating input DATA step views 50
creating output data sets 48
creating stored compiled DATA step programs 49
describing DATA step views 49
details 48
displaying nesting levels 52
examples 50
executing stored compiled DATA step programs 50
when not creating data sets 49
without arguments 44
DATA step
aborting 14
BY statement in 33
MODIFY statement in 316
stopping 382, 409
DATA step programs
stored compiled, executing 73
DATA step programs, retrieving source code from 56
DATA step views
creating 49
describing 49
retrieving source code from 56
DATALINES argument
INFILE statement 199, 219
DATALINES statement 37, 53
compared to DATALINES4 statement 55
DATALINES4 54
DATALINES4 statement 37, 54
DEBUG option
FILENAME statement, FTP access method 128
FILENAME statement, Hadoop access method 143
FILENAME statement, SFTP access method 149
FILENAME statement, URL access method 162
FILENAME statement, WebDAV access method 167
FILENAME statement, ZIP access method 175
declarative DATA step statements 2
declarative statements 2
DEFAULT= argument
INFORMAT statement 228
LENGTH statement 271
DEL_ALL option
FILENAME statement, WebDAV access method 167
DELETE argument, DISPLAY statement 57
DELETE statement 55
compared to DROP statement 69
compared to IF statement, subsetting 188
delimited data 261
reading 214
reading from external file 99
delimiter sensitive data
FILE statement 73
DELIMITER= option
FILE statement 73
INFILE statement 202, 219
delimiters
INFILE statement 219
DESC= option
FILENAME statement, CATALOG access method 103
DESCENDING argument
BY statement 31
DESCRIBE statement 56
DIR option
FILENAME statement, FTP access method 128
FILENAME statement, SFTP access method 149
FILENAME statement, WebDAV access method 167
DIR= option
FILENAME statement, Hadoop access method 143
direct access
by indexed values 314
by observation number 315
directories
reading and writing from 140
reading from member of 172
writing to new member of 171
directory listings
retrieving 137
DISPLAY statement 57
compared to WINDOW statement 442
DISPLAY= option, WINDOW statement 440
DLM= option
INFILE statement 202
DLMSOPT= option
FILE statement 73
INFILE statement 204
DLMSTR= option
FILE statement 73
INFILE statement 203
DM statement 59
DO statement 61
compared to DO UNTIL statement 67
compared to DO WHILE statement 68
DO statement, iterative 62
compared to DO statement 61
compared to DO UNTIL statement 67
compared to DO WHILE statement 68
DO UNTIL statement 66
compared to DO statement 61
compared to DO statement, iterative 64
compared to DO WHILE statement 68
DO WHILE statement 67
compared to DO statement 61
compared to DO statement, iterative 64
compared to DO UNTIL statement 67
DO-loop processing termination value 404
DO-loops
DO statement 61
DO statement, iterative 62
DO UNTIL statement 66
DO WHILE statement 67
ending 70
GO TO statement 185
resuming 43, 270
stopping 43, 270
dollar sign ($) argument
INPUT statement 232
INPUT statement, column input 248
INPUT statement, named input 262
LENGTH statement 271
double trailing @
INPUT statement, list 255
DROP statement 68
compared to DELETE statement 56
compared to KEEP statement 266
DROP= data set option
compared to DROP statement 69
DROPOVER option
FILE statement 73
DSD option
FILE statement 73
INFILE statement 204, 220

E
email
attachments 122
creating and sending images 124
options for FILENAME statement, EMAIL access method 111
procedure output in 123
sending from SAS with SMTP 111
EMAIL (SMTP) access method
See FILENAME statement, EMAIL (SMTP) access method
embedded blanks
character data with 260
encoded passwords 138
encoding
for output files 93
ENCODING= option
FILE statement 73, 93
FILENAME statement 97, 101, 102
FILENAME statement, EMAIL access method 114
FILENAME statement, FTP access method 129
FILENAME statement, Hadoop access method 143
FILENAME statement, SOCKET access 156
FILENAME statement, WebDAV access method 167
FILENAME statement, ZIP access method 175
INFILE statement 204, 227
END statement 70
END= argument
MODIFY statement 309
UPDATE statement 422
END= option
INFILE statement 205
SET statement 394, 404
ENDSAS command, compared to
ENDSAS statement 71
FILENAME statement 71
environment variable
SAS_FTP_AUTHTLS 449
EOF= option
INFILE statement 205
EOV= option
INFILE statement 205
error messages
writing 71
ERROR statement 71
executable DATA step statements 2
executable statements 2
EXECUTE statement 73
EXPANDTABS option
INFILE statement 205
expressions, summing 410
EXTENDOBSCOUNTER= option
LIBNAME statement 279
external files
associating filerefs 99, 100
definition 98
dissociating filerefs 97, 99
encoding specification 97, 101, 102
identifying a file to read 198
including 196
reading delimited data from 99
referencing catalogs as 103
updating in place 88, 213, 224
writing attributes to log 97, 99

F
FILE argument
ABORT statement 14
file extensions
attaching automatically 173
FILENAME statement 73
arguments 73
arranging contents of entire page 90
comparisons 89
current output file 91
details 87
encoding for output file 93
examples 90
executing statements at new page 90
external files, updating in place 88
lockdown 73
operating environment options 73
options 73
output buffer, accessing contents 88
output line too long 92
page breaks 90
TCP/IP socket and 92
updating _FILE_ variable 88
FILEEXT option
FILENAME statement, FTP access method 129
FILENAME statement, WebDAV access method 167, 173
FILEEXT= option
FILENAME statement, Hadoop access method 144
FILENAME statement, ZIP access method 176
FILENAME statement 94
arguments 94
compared with REDIRECT statement 370
comparisons 99
definitions 98
details 98
dissociating filerefs 97, 99
encoding specification 97, 101, 102
examples 100
filerefs for aggregate storage location 100
filerefs for external files 99, 100
filerefs for output devices 99
LIBNAME statement and 100
lockdown 94
operating environment information 98
reading delimited data from external files 99
routing PUT statement output 101
writing file attributes to log 97, 99
FILENAME statement, CATALOG access method 103
%INCLUDE with catalog entries 105
catalog options 103
details 104
examples 105
executing autocall macros from catalogs 105
reading and writing CATAMS entries 105
FILENAME statement, CLIPBOARD access method 106
FILENAME statement, DATAURL access method 108
arguments 108
DATAURL options 108
FILENAME statement, EMAIL (SMTP) access method 111
attachments with email 122
conditional logic in DATA step 123
creating and emailing images 124
details 120
details 120
email options 111
examples 122
sending procedure output 123
FILENAME statement, FTP access method 126
arguments 126
comparisons 137
creating files on remote host 138
creating transport libraries with transport engine 139
encoded passwords 138
examples 137
FTP anonymous login 138
FTP options 126
importing transport data sets 139
proxy servers 140
reading and writing from directories 140
reading files from remote host 137
reading S370V files on z/OS 138
retrieving directory listings 137
transporting libraries 139
FILENAME statement, Hadoop access method 141
FILENAME statement, SFTP access method 148
arguments 148
comparisons 153
details 152
examples 153
prompts 153
SFTP options 149
FILENAME statement, SOCKET access method 155
client mode 158
details 158
details 158
examples 159
server mode 158
TCPIP options 155
FILENAME statement, URL access method 160, 174
accessing files at a website 164
arguments 160
details 164
examples 164
reading part of a URL file 165
URL options 160
user ID and password 165
ZIP options 174
FILENAME statement, WebDAV access method 166
accessing files at a website 171
accessing files with mixed-cased names 173
accessing lowercased autocall macro member 172
arguments 166
automatically attaching file extensions 173
deleting a directory that contains members 173
details 170
examples 171
proxy servers 171
reading from directory member 172
Sharepoint files 170
WebDAV location as autocall macro library 172
WebDAV options 166
writing to new directory member 171
FILENAME statement, ZIP access method
arguments 174
FILENAME= option
FILE statement 73
INFILE statement 205
filerefs
associating with aggregate storage location 100
associating with external files 99, 100
associating with output devices 99
   definition 98
disassociating from external files 97, 99
   FILENAME statement 94
files, master
   updating 422
FILEVAR= option
   FILE statement 73, 91
   INFILE statement 206, 223
FIRST. variable 33
FIRSTOBS= option
   INFILE statement 206
FLOWOVER option
   FILE statement 73
   INFILE statement 206, 221
FOOTNOTE statement 178
   comparisons 181
details 181
   examples 181
   without arguments 178
footnotes
   customizing with ODS 417
FOOTNOTES option
   FILE statement 73
FORMAT statement 182
   formats
   associating with variables 182
   formatted input 238, 251
   modified list input vs. 259
   formatted output 342, 356
FROM= option
   FILENAME statement, EMAIL access method 115
FTP
   anonymous login 138
FTP access method
   See FILENAME statement, FTP access method

G
   global DATA step statements
      definition 3
   GO TO statement 185
   GOTO statement 185
   GROUP= operator, WINDOW statement 434
   GROUPFORMAT argument
      BY statement 31
grouping observations 35
      with formatted values 35

H
   Hadoop access method
FILENAME statement, Hadoop access method 141
   HEADER= option
   FILE statement 73, 90
   HEADERS= option
FILENAME statement, URL access method 162
   HOST= option
   FILENAME statement, FTP access method 130
   FILENAME statement, SFTP access method 150
   HOSTRESPONSELEN= option
   FILENAME statement, FTP access method 130
I
   I/O control
      MODIFY statement 326
   ICOLUMN= argument, WINDOW statement 434
   IF statement, subsetting 187
      compared to DELETE statement 56
   IF, THEN/ELSE statements 189
      compared to IF statement, subsetting 188
   images
      sending in email 124
   IMPORTANCE= option
      FILENAME statement, EMAIL access method 115
importing
   transfer data sets 139
   including programming statements and data lines 191
indexed values
   direct access by 314
indexes
   duplicate values 314, 324
   INDNAME option
      SET statement 394
   INENCODING= option
      LIBNAME statement 280
INFILE statement 198
   arguments 198
   compared to INPUT statement 243
   comparisons 219
   DBMS specifications 198
delimited data, reading 214
delimiters 219
details 212
   encoding specification 227
examples 219
   input buffer, accessing contents 213
   input buffer, working with data 225
lockdown 198
missing values, list input 221
multiple input files 213, 223
operating environment options 198
options 198
pointer location 224
reading long instream data records 217
reading past the end of a line 217
short records 221
truncating copied records 224
updating external files in place 213, 224
variable-length records, reading 222
variable-length records, scanning 221
INFORMAT statement 228
informs
associating with variables 228
reading unaligned data with 260
input
assigning to variables 232
column 237, 248
describing format of 232
end-of-data indicator 329
formatted 238, 251
invalid data 242, 301
list 238
list input 255
listing for current session 295
logging 295
missing records 301
missing values 308
named 238, 262
resynchronizing 301
input buffer
accessing contents 213
accessing, for multiple files 226
working with data in 225
input column 251
input data
reading past the end of a line 217
input data sets
redirecting 369
input DATA step views
creating 50
input files
reading multiple files 213, 223
truncating copied records 224
INPUT statement 232
column 248
compared to PUT statement 346
formatted 251
identifying file to be read 198
named 262
INPUT statement, column 248
INPUT statement, formatted 251
INPUT statement, list 255
details 257
examples 259
INPUT statement, named 262
instream data
reading long records 217
interleaving data sets
SET statement for 401, 402
IOM clients
tracking submitted SAS programs 411
IROW= argument, WINDOW statement 434
J
JMP engine LIBNAME statement 288
K
KEEP statement 266
compared to DROP statement 69
compared to RETAIN statement 380
KEEP= data set option
compared to KEEP statement 266
KEY= argument
MODIFY statement 309
KEY= option
SET statement 394, 403
keyboard input
including 197
KEYS= argument, WINDOW statement 434
L
LABEL statement 267
compared to statement labels 269
label: statement 269
labels
statement labels 269
LAST. variable 33
LEAVE statement 43, 270
compared to CONTINUE statement 43
LENGTH statement 271
LENGTH= option
INFILE statement 207, 222
LIBNAME statement 274
arguments 274
assigning librefs 286
associating librefs with data libraries 283
comparisons 285
concatenating catalogs, implicitly 284, 286
concatenating data libraries 284
concatenating data libraries, logically 286
data library attributes, writing to log 284
details 283
disassociating librefs from data libraries 284
disassociating librefs from data libraries 284
gine-host-options 274
elements 286
FILENAME statement and 100
library concatenation rules 284
lockdown 274
options 274
permanently storing data sets, one-level names 287
LIBNAME statement, for JMP Engine 288
LIBNAME statement, for WebDAV Servers 289
libraries
transporting 139
library concatenation rules 284
librefs
assigning 286
associating with data libraries 283
disassociating from data libraries 284
line pointer controls
INPUT statement 232
PUT statement 335
line-hold specifiers
INPUT statement 240
PUT statement 344
LINE= option
FILE statement 73
INFILE statement 207
LINES statement 37, 53
LINES4 statement 37, 54
LINESIZE= option
FILE statement 73
INFILE statement 207
LINESLEFT= option
FILE statement 73, 90
LINK statement 293
compared to GO TO statement 186
LIST argument
FILENAME statement 97, 99
LIBNAME statement 276
list input 238, 255
character data with embedded blanks 260
comma-delimited data 260
data with quotation marks 259
missing values in 221
modified 258, 259, 261
reading delimited data 261
reading unaligned data 259
reading unaligned data with informats 260
simple 258, 259
when to use 257
LIST option
CATNAME statement 38
FILENAME statement, FTP access method 130
list output 342, 362
See also PUT statement, list
PUT statement, list 360
spacing 362
writing values with 363
LIST statement 295
LOCALCACHE= option
FILENAME statement, WebDAV access method 168
LOCK statement 298
lockdown
FILE statement 73
FILENAME statement 94
INFILE statement 198
LIBNAME statement 274
LOCKDURATION= option
FILENAME statement, WebDAV access method 168
log
writing data library attributes to 284
writing external file attributes to 97, 99
writing messages to 367
LOSTCARD statement 301
LOWCASE_MEMNAME option
FILENAME statement, FTP access method 130
FILENAME statement, WebDAV access method 168
LOWCASE_MEMNAME= option
FILENAME statement, Hadoop access method 144
FILENAME statement, ZIP access method 176
LRECL= option
FILE statement 73
FILENAME statement, CATALOG access method 103
FILENAME statement, EMAIL access method 113, 115
FILENAME statement, FTP access method 130
FILENAME statement, Hadoop access method 144
FILENAME statement, SFTP access method 150
FILENAME statement, SOCKET access 157
FILENAME statement, URL access method 162
FILENAME statement, WebDAV access method 168
FILENAME statement, ZIP access method 176
INFILE statement 207
LS option
FILENAME statement, FTP access method 130
FILENAME statement, SFTP access method 150
LS= option
FILENAME statement, FTP access method 130
FILENAME statement, SFTP access method 150
LSFILE= option
FILENAME statement, FTP access method 131
FILENAME statement, SFTP access method 150
FILENAME statement, Hadoop access method 144
MOD option
FILE statement 73
FILENAME statement, CATALOG access method 103
FILENAME statement, WebDAV access method 169
MOD= option
FILENAME statement, Hadoop access method 144
modified list input 258
formatted input vs. 259
reading delimited data 261
modified list output 362
vs. formatted output 362
writing values with : 364
writing values with ~ 364
MODIFY statement 309
_IORC_ automatic variable 315
collections 318
data set options with 318
details 314
direct access by indexed values 314
direct access by observation number 315
duplicate BY values 314
duplicate index values 314, 324
duplicates 319
I/O control 326
in DATA step 316
matching access 314
SAS/SHARE environment 318
sequential access 315
SYSRC autocall macro 315
MPROMPT option
FILENAME statement, FTP access method 131

M
master files, updating 422
match-merge 306
matching access 314
MAXWAIT= option
FILENAME statement, Hadoop access method 144
MEMBER= argument
FILENAME, ZIP Access Method 175
MENU= argument, WINDOW statement 434
MERGE statement 304
compared to UPDATE statement 424
merging observations 403
messages
writing to log 367
MGET option
FILENAME statement, FTP access method 131
FILENAME statement, SFTP access method 151
missing records, input 301
MISSING statement 308
missing values
input 308
list input 221
MISSING statement 308
reading external files 221
substitute characters for 308
MISSING= system option
compared to MISSING statement 308
MISSOVER option
INFILE statement 208, 221
MKDIR option

N
N= option
FILE statement 73, 90
INFILE statement 208
named input 238, 262
named output 343, 365
NBYTE= option
INFILE statement 92, 209
nested catalog concatenation 40
nesting levels
displaying 52
NEW option
FILENAME statement, FTP access method 131
FILENAME statement, SFTP access method 151
NEW= option
FILENAME statement, Hadoop access method 144
NOBS= option
MODIFY statement 309
SET statement 394, 404
NOINPUT argument, DISPLAY statement 57
NOLIST argument
ABORT statement 14
DATA statement 44
NOTSORTED argument
BY statement 31
Null statement 329

O
OBS= option
INFILE statement 209
observations combining multiple 35
deleting 55, 370
dropping 68
grouping 35
grouping with formatted values 35
merging 304, 403
modifying 309, 319
modifying, located by index 323
modifying, located by number 322
modifying, with transaction data set 320
modifying, writing to different data sets 328
multiple records for 241
reading subsets 404
reading with SET statement 394
replacing 374
writing 332
observations, selecting
IF, subsetting 187
IF, THEN/ELSE statement 189
WHERE statement 427
ODS (Output Delivery System)
customizing titles and footnotes 417
ODS option
FILE statement 73
OLD option
FILE statement 73
one-to-one merge 306
one-to-one reading 402, 403
OPEN= option
SET statement 394
operating environment
FILE statement options for 73
operating system commands
issuing from SAS sessions 446
OPTIONS statement 331
OPTIONS= option
FILENAME statement, SFTP access method 151
OPTIONSX= option
FILENAME statement, SFTP access method 151
OUTENCODING= option
LIBNAME statement 280
output
column 342, 354
footnotes 178
formatted 342, 356
list 342
named 343, 365
output buffer
accessing contents 88
output data sets 48
creating 48
redirecting 369
output devices
associating filerefs 99
output files
dynamically changing current file 91
encoding for 93
for PUT statements 73
identifying current file 91
output line too long 92
OUTPUT statement 332
compared to REMOVE statement 371
compared to REPLACE statement 375
OUTREP= option
LIBNAME statement 280
OVERPRINT option, PUT statement 335

P
P= option, WINDOW statement 441
PAD option
FILE statement 73
INFILE statement 209
page breaks
determined by lines left on current page 90
executing statements at 90
page size
two-column format 90
PAGE statement 335
PAGESIZE= option
FILE statement 73
PASS= option
FILENAME statement, FTP access method 131
FILENAME statement, Hadoop access method 145
FILENAME statement, URL access method 162
FILENAME statement, WebDAV access method 169
PASSWORD= option
FILENAME statement, WebDAV access method 169
passwords
ALTER 48
DATA step and 48
encoded 138
READ 48
stored compiled DATA step programs with 51
PATH option
FILENAME statement, SFTP access method 151
PBSZ= option
FILENAME statement, FTP access method 132
PERSIST= option, WINDOW statement 440
PGM= argument
DATA statement 44
plotters
filerefs for 99
plus sign (+) column pointer control
INPUT statement 235
PUT statement 335
WINDOW statement 438
POINT= option
MODIFY statement 309
SET statement 394, 404
pointer controls
INPUT statement 239
PUT statement 344
pointer location 224
POINTOBS= option
LIBNAME statement 282
PORT= option
FILENAME statement, FTP access method 132
pound sign (#) line pointer control
INPUT statement 236
PUT statement 335
PPASS= option
FILENAME statement, URL access method 162
PRINT option
FILE statement 73
INFILE statement 209
printers
filerefs for 99
PROC steps
BY statement in 33
procedure output
footnotes 178
sending in email 123
submitting as SAS statements 59
Program Editor commands, submitting as SAS statements 59
flow into main entry 59
programming statements
including 191
PROMPT option
FILENAME statement, FTP access method 132
FILENAME statement, URL access method 163
FILENAME statement, WebDAV access method 169
PROMPT= option
FILENAME statement, Hadoop access method 145
PROT= option
FILENAME statement, FTP access method 133
PROTECT= option, WINDOW statement 441
proxy servers 140, 171
PROXY= option
FILENAME statement, URL access method 163
PUT statement 335
compared to INPUT statement 243
compared to LIST statement 296
FILE statement and 73
output file for 73
routing output 101
PUT statement, column 354
PUT statement, formatted 356
PUT statement, list 360
arguments 360
comparisons 362
details 362
examples 363
list output 362
list output, spacing 362
list output, writing values with 363
modified list output vs. formatted output 362
modified list output, writing values 364
writing character strings 363
writing variable values 363
PUT statement, named 365
PUTLOG statement 367
PUTTY client
connecting to SSHD server 154
PW= option
FILENAME statement, WebDAV access method 169
PWD= option
FILENAME statement, WebDAV access method 169

Q
question mark (?) format modifier
INPUT statement 236
question marks (??) format modifier
INPUT statement 237

R
RCFM= option
FILENAME statement, ZIP access method 176
RCMD= option
FILENAME statement, FTP access method 133
READ passwords 48
reading
from directories 140
reading past the end of a line 217
RECFM= option
FILE statement 73
FILENAME statement 98
FILENAME statement, CATALOG access method 103
FILENAME statement, FTP access method 134
FILENAME statement, Hadoop access method 145
FILENAME statement, SFTP access method 151
FILENAME statement, SOCKET access 157
FILENAME statement, URL access method 163
FILENAME statement, WebDAV access method 169
INFILE statement 209
RECFM=F option
FILENAME statement, FTP access method 128
RECONN= option
FILENAME statement, SOCKET access 157
REDIRECT statement 369
arguments 369
examples 370
redirecting data sets 369
remote files
DATAURL access method 108
FTP access method 126
SFTP access method 126
URL access method 148
WebDAV access method 160
ZIP access method 174
remote host
creating files on 138
reading files from 137
reading files from a directory 154
REMOVE statement 370
compared to OUTPUT statement 333
compared to REPLACE statement 370, 375
RENAME statement 372
RENAME= data set option compared to RENAME statement 373
REPEMPTY= option
LIBNAME statement 283
REPLACE statement 374
compared to OUTPUT statement 333
compared to REMOVE statement 371
REPLYTO= option
FILENAME statement, EMAIL access method 116
reports
creating with DATA statement 51
REQUIRED= option, WINDOW statement 441
RESETLINE statement 376
RETAIN statement 378
compared to KEEP statement 266
compared to SUM statement 411
RETURN argument
ABORT statement 14
RETURN statement 382
compared to GO TO statement 186
RHELP option
FILENAME statement, FTP access method 135
ROWS= argument, WINDOW statement 434
RSTAT option
FILENAME statement, FTP access method 135
RUN statement 383

S
S2= argument
%INCLUDE statement 194
S370V files
reading on z/OS 138
S370V option
FILENAME statement, FTP access method 135
S370VS option
  FILENAME statement, FTP access method 135
SAS data sets
  deleting observations 370
  redirecting 369
  writing to 332
SAS jobs
  aborting 14
SAS jobs, terminating 71
SAS log
  logging input 295
  skipping to new page 335
SAS OPTIONS window, compared to OPTIONS statement 332
SAS programs
  including statements or data lines 191
  tracking, for IOM clients 411
SAS sessions
  aborting 14
  issuing operating-system commands 446
  terminating 71
SAS statements
  aborting 14
  semicolon (;), in data lines 37, 54
SAS views
  BY statement with 33
  SOURCE entries writing to 105
SOURCE= argument
  DATA statement 44
SOURCE2 argument
  %INCLUDE statement 194
SSHD server
  connecting at non-standard port 153
  connecting at standard port 153
  connecting Windows PUTTY client to 154
START= option
  BY-group processing with 401
  combining data sets 401
  compared to INPUT statement 243
  compared to MERGE statement 306
  comparisons 402
  concatenating data sets 401, 402
details 400
  examples 402
  interleaving data sets 401, 402
  merging observations 403
  one-to-one reading 402, 403
  options 394
  reading subsets 404
table-lookup 403
SFTP access method
  See FILENAME statement, SFTP access method
SFTP argument
  FILENAME statement, SFTP access method 148
SHAREBUFFERS option
  INFILE statement 210, 224
SHAREBUFS option
  INFILE statement 210
SHAREPOINT_COMP_MODE
  environment variable 170
short records 221
Simple Mail Transfer Protocol
  See FILENAME statement, EMAIL (SMTP) access method
SKIP statement 408
slash (/) line pointer control
  INPUT statement 236
  PUT statement 335
SMTP access method
  See FILENAME statement, EMAIL (SMTP) access method
SOCKET access method
  FILENAME statement 155
SOCKET argument
  FILENAME statement, SOCKET access 156
sort order
  specifying with BY statement 34
SOURCE entries
  writing to 105
SOURCE= argument
  DATA statement 44
SOURCE2 argument
  %INCLUDE statement 194
INDEX

INDEX

INFILE statement 210
statement labels 269
statement labels, jumping to 293
statements 1
  DATA step statements 2
  declarative 2
  executable 2
  executing at page break 90
STOP statement 409
STOPOVER option
  FILE statement 73
INFILE statement 210, 221
stored compiled DATA step programs
  creating 49
  executing 50, 73
  passwords with 51
  retrieving source code from 56
SUBJECT= option
  FILENAME statement, EMAIL access
  method 116
SUM function
  compared to SUM statement 411
Sum statement 410
summing expressions 410
SYSECHO statement 411
SYSRC autocall macro
  MODIFY statement and 315

T
  table-lookup
    duplicate observations in master file
    403
TCP/IP socket
  reading and writing text through 92
TCP/IP socket access
  FILENAME statement 155
TCPIP-options
  FILENAME statement, SOCKET
  access 155
terminals
  filerefs for 99
TERMSTR= option
  FILENAME statement, ZIP access
  method 177
FILENAME statement, SOCKET
  access 158
FILENAME statement, URL access
  method 164
text editor commands, submitting as SAS
  statements 59
  flow into main entry 59
  tilde (~) format modifier 255, 258
TITLE statement 412
titles
  customizing with BY variables 417
  customizing with ODS 417
TITLES option
  FILE statement 73
TLS protocol
  FILENAME statement, FTP access
    method 136
FILENAME statement, URL access
    method 164
FILENAME statement, WebDAV access
    method 170
TO statement, compared to LINK
    statement 294
TO= option
  FILENAME statement, EMAIL access
    method 116
trailing @
  INPUT statement, list 255
transaction data sets
  modifying observations 320
transport data sets
  importing 139
transport engine
  creating transport libraries with 139
Transport Layer Security (TLS) protocol
  FILENAME statement, FTP access
    method 136
FILENAME statement, URL access
    method 164
FILENAME statement, WebDAV access
    method 170
transport libraries
  creating with transport engine 139
transporting libraries 139
truncating
  copied records 224
TRUNCOVER option
  FILENAME statement 211, 221
two-column format 90
TYPE= option
  FILENAME statement, EMAIL access
    method 114

U
UID= option
  FILENAME statement, WebDAV access
    method 170
unaligned data 259
UNBUF option
  FILENAME statement 211
UNBUFFERED option
  FILENAME statement 211
UNIQUE option
  SET statement 394
UNIQUE= option
  MODIFY statement 309

468 Index
Universal Printers
filerefs for 99
UPDATE statement 422
compared to MERGE statement 306
UPDATEMODE= argument
UPDATE statement 422
UPDATEMODE= option
MODIFY statement 309
URL access method
See FILENAME statement, DATAURL access method
See FILENAME statement, URL access method
USER= option
FILENAME statement, FTP access method 136
FILENAME statement, Hadoop access method 146
FILENAME statement, SFTP access method 152
FILENAME statement, URL access method 164
FILENAME statement, WebDAV access method 170

V
variable-length records
reading 222
scanning for character string 221
variables
_ERROR_, setting 71
assigning input to 232
associating formats with 182
associating informats with 228
BY variables 34
FIRST. 33
labeling 267
LAST. 33
length, specifying 271
renaming 372
retaining values 378
VIEW= argument
DATA statement 44

W
WAIT_MILLISECONDS= option
FILENAME statement, SFTP access method 152
WEBDAV 289
WebDAV access method
See FILENAME statement, WebDAV access method
websites
accessing files at 164, 171
WHEN statement 391
in SELECT groups 392
WHERE statement 427
compared to IF statement, subsetting 188
WINDOW statement 434
Windows PUTTY client
connecting to SSHD server 154
windows, displaying 57, 434
writing from directories 140

X
X command, compared to X statement 447
X statement 446

Z
z/OS
reading S370V files 138
ZIP access method
See FILENAME statement, ZIP access method
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.